Formal Definitions and Proofs for Partial (Co)Recursive Functions

Horatiu Cheval, David Nowak, Vlad Rusu

Formal Definitions and Proofs for Partial (Co)Recursive Functions

Horatiu Cheval[1]

Research Center for Logic, Optimization and Security (LOS), Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Romania

David Nowak

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Vlad Rusu

Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

Partial functions are a key concept in programming. Without partiality a programming language has limited expressiveness - it is not Turing-complete, hence, some programs cannot be written. In *functional* programming languages, partiality mostly originates from the non-termination of recursive functions. *Corecursive* functions are another source of partiality: here, the issue is not non-termination, but the inability to produce arbitrary large, finite approximations of a theoretically infinite output.

Partial functions have been formally studied in the branch of theoretical computer science called *domain theory*. In this paper we propose to step up the level of formality by using the Coq proof assistant. The main difficulty is that Coq requires all functions to be total, since partiality would break the soundness of its underlying logic. We propose practical solutions for this issue, and others, which appear when one attempts to define and reason about partial (co)recursive functions in a total functional language.

1. Introduction

Partiality is a key concept in programming and in particular in functional programming. In practice, functional programmers encounter partiality by running recursive functions and noting that they appear to be non-terminating on some inputs. In certain functional languages such as Haskell [1] one can also write corecursive functions, whose expected output is, in theory, infinite; e.g., the sieve of Eratosthenes computing the infinite sequence of prime numbers. Corecursive functions can also be partial: for

[1]Partially supported by COST Action EuroProofNet CA20111, funded by COST (European Cooperation in Science and Technology).

example, a buggy implementation of the sieve of Eratosthenes may, at some point, stop producing primes: it does not terminate, but does not produce any more output either.

Undersirable as it is, partiality is essential for expressiveness: without it a language is not Turing-complete. It is therefore important to formally define and reason about partiality. This has been done in the discipline known as domain theory [2, 3] and its application, the denotational semantics of programming languages [4, 5]. Mostly, recursive functions and the *inductive* types that constitute their inputs have been studied. Two exceptions are Kahn networks [6, 7], which are corecursive and do produce output in *coinductive* (i.e., infinite) streams (although they were introduced before the notions of corecursion/coinduction had emerged). More recently, the denotational semantics of Haskell sketched in [8] touches on corecursive functions and coinductive types.

Proof assistants such as Coq [9], Isabelle/HOL [10], Agda [11] and Lean [12] also favor recursive functions and inductive types over the dual notions - recursive functions and coinductive types. Basic support for the dual notions in the above-mentioned proof assistants typically requires that corecursive functions satisfy a strong syntactical criterion of *guardedness*: each corecursive call must occur, up to standard reductions, directly under a call to a constructor of the coinductive type being produced. This is a sufficient condition ensuring the productiveness of each corecursive call and, by way of consequence, totality. It guarantees the soundness of their underlying logics, but severely limits the class of corecursive functions accepted by the tools. Agda and Isabelle/HOL also offer advanced support, enabling them to accept broader classes of corecursive functions [13, 14]. The corecursive functions in question, like their recursive counterparts, are total - although partiality can, to some extent, be simulated.

Contributions. We use, adapt and occasionally contribute to elements of domain theory that enable the formal definition and reasoning about partial (co)recursive functions, and implement the results as a library in the Coq proof assistant. Our main contribution is practical; all theory is in the service of practice. In the following order:

1. We first define an encoding of coinductive types, distinct from the existing one in our target proof assistant. One starts with the built-in *inductive* types and their constructors, additionally endowed with a so-called *definition order* and a special constructor for "undefined" terms. A *completion* operation - a variant of the so-called *ideal completion* in domain theory - transforms inductive types and their constructors into an encoding of coinductive types and of their corresponding constructors;

2. Then, we build proof tools for reasoning terms on the resulting types. A coinductively defined predicate characterizes *total* terms - those without "undefined" subterms. A coinductive notion of *bisimulation* is introduced and is proved equivalent with equality. These notions come with dedicated coinductive proof principles:

3. Next, we provide techniques for defining encodings of possibly-partial (co)recursive functions that produce outputs in the coinductive types defined at the previous step[2]. Like in classical approaches [2, 3, 4], each function is defined as the *least fixpoint*

[2]Recursive functions produce values in a type defined without self-reference (but with an added "undefined" value). In this sense recursive functions are just a particular case of corecursive functions.

of its corresponding *functional*, provided the functional satisfies a *continuity* requirement. But, unlike the above, we do not use *Kleene's fixpoint theorem* for this purpose: proving continuity for higher-order functionals is not practically feasible because it involves computing *least upper bounds* (*lub*s) in complex orders. We use instead a theorem that we call *Haddock's fixpoint theorem*, which uses a condition that is logically equivalent to continuity but involves *lub*s in simpler orders. As a consequence, Haddock's theorem can be used in practice for defining (co)recursive functions (possibly, after several iterations of simplification) in situations when direct applications of Kleene's fixpoint theorem fail[3]. We also extend the notion of *totality* and its coinductive proof principle from terms to (co)recursive functions.

4. Finally, we apply the various ingredients of the approach to defining and reasoning about concrete examples of possibly-partial (co)recursive functions:

   - a corecursive *filter* function on streams, which outputs the subsequence of values in its input that satisfy a given predicate. This function is partial: if, after some point, no more value in the input satisfy the predicate, the output is not a stream;
   - a corecursive *mirror* function on Rose trees (coinductive trees, with finite breadth and possibly infinite depth), whose output mirrors its input. This function does not satisfy the guardedness-by-constructors totality criterion, but is proved total;
   - a recursive *collatz* function, which we define as a partial function because its termination is an unsolved conjecture in mathematics;
   - a recursive *while* function, which is a monadic encoding of while-loops. Like the loops it encodes, this function is possibly non-terminating, hence, it is partial.

   For each example, some specific properties are proved using the techniques introduced in the paper, with details demonstrating the adequacy of the techniques.

Outline. After some preliminaries in Section 2, we present our construction of coinductive types and of their associated proof techniques in Section 3. Section 4 shows how one can effectively define possibly-partial (co)recursive functions as fixpoints of their so-called "Haddock-continuous" functionals. Section 5 illustrates the proposed techniques on several concrete examples of (co)recursive functions. We discuss our implementation in Section 6, before concluding and presenting related and future work in Section 7. An Appendix contains proofs from Section 2 that we could not locate in the literature. The Coq development corresponding to the paper can be found at `https://github.com/vladmgrusu/haddock`.

2. Preliminaries

This section introduces elements of domain theory used throughout the paper. We follow the books [2, 3, 4]. Definitions and theorems are fully spelled out. Proofs are given (in the Appendix) if they are nontrivial and/or not present in the books. Almost everything is covered in the cited books. One key result is, to our best knowledge, new.

[3] A natural question that arises is: how does one use Kleene's theorem for defining language constructs in denotational semantics? This is discussed and compared with our approach in related works (Section 7.2).

2.1. Orders

In this section we present a series of increasingly expressive orders, starting from partially-ordered sets and ending with the key notion of algebraic CPO. We also present several construction techniques for building more complex orders from simpler ones.

2.1.1. Basic Definitions

Definition 1. *A partially-ordered set (poset) is a pair* $(C, \preceq)$ *consisting of a set* C *and a partial order* $\preceq$ *on* C*.*

Example 1. *Any set with equality as order (a.k.a. the* discrete *order) is a poset. The natural numbers with their usual order* $(\mathbb{N}, \leq)$ *form a poset, which, additionally, is* total*.*

Posets are given additional structure in several ways. One may identify a least element:

Definition 2. *A Pointed Partial Order (PPO) is a triple* $(C, \preceq, \bot)$ *where* $(C, \preceq)$ *is a poset and* $\bot \in C$ *satisfies* $\bot \preceq c$ *for all* $c \in C$*.*

Example 2. *If* $(A, \preceq)$ *is a poset and* $\bot \notin A$*, then* $(A_\bot, \preceq_\bot, \bot)$ *with* $A_\bot = A \cup \{\bot\}$ *and* $\preceq_\bot = \preceq \cup \{(\bot, a) \mid a \in A_\bot\}$ *is a PPO. If* $\preceq$ *is the discrete order on A, i.e., equality, then* $(A_\bot, \preceq_\bot, \bot)$ *is a PPO called the* flat *PPO of A. Another example of PPO is* $(\mathbb{N}, \leq, 0)$*.*

Remark. In this paper, the order in a PPO is – with the notable exception of $(\mathbb{N}, \leq, 0)$ – interpreted as a definition order: $\bot$ is interpreted as *undefined*, and $x \preceq y$ means that *x is at most as defined as y*. For example in the flat PPO $(A_\bot, \preceq_\bot, \bot)$, $\bot$ is undefined and all the other elements $a \in A$ are completely defined since one cannot go further in the definition order. The exception is $(\mathbb{N}, \leq, 0)$: $\bot$, i.e., 0, is not naturally interpreted as being undefined, and $1 \leq 42$ does not naturally mean that 1 is at most as defined as 42.

Another manner in which posets can be enriched is by identifying a notion of "limit":

Definition 3. *Given a poset* $(C, \preceq)$ *and a set* $S \subseteq C$*, the* least upper bound *of* S*, denoted by lub* S*, is an element* $c \in C$ *such that c is an* upper bound *of* S*: for all* $s \in S$*,* $s \preceq c$*; and c is* minimal *with that property: for all upper bounds* c' *of* S*, it holds that* $c \preceq c'$*.*

Example 3. *In the discrete order* $(A, =)$ *only singletons* $\{a\}$ *have least upper bounds, and lub* $\{a\} = a$*. In* $(\mathbb{N}, \leq)$ *any nonempty finite set S has a lub, which coincides with the maximum of S. The set* $\mathbb{N}$ *itself does not have any upper bound, least or other.*

The limits make sense only for sets that, in the sense defined below, do not "diverge":

Definition 4. *Given a poset* $(C, \preceq)$*, a set* $S \subseteq C$ *is* directed *if* $S \neq \emptyset$ *and for all* $x, y \in S$ *there exists* $z \in S$ *such that* $x, y \preceq z$*.*

Example 4. *In* $(A, =)$ *only singletons* $\{a\}$ *are directed. In the flat PPO* $(A_\bot, \preceq_\bot, \bot)$ *the directed sets are singletons and pairs of the form* $\{\bot, a\}$ *with* $a \in A$*. In* $(\mathbb{N}, \leq)$ *all nonempty subsets are directed since* $\leq$ *is total: for* $x, y \in \mathbb{N}$*, max x y is an upper bound.*

Remark. Directed sets generalize nonempty total orders. The intuition is that two elements x and y in the set may have defined different features, e.g., they have developed two different branches in a tree; but they will eventually evolve into an element that has at least all the defined features of x and y. Hence the set as a whole does not "diverge".

Definition 5. *A Directed Complete Partial Order (DCPO) is a poset* $(C, \leq)$ *with the additional property that each directed set* $S \subseteq C$ *has a least upper bound.*

Example 5. $(A, =)$ *is a DCPO, the* discrete DCPO of A. $(\mathbb{N}, \leq)$ *is* not *a DCPO because* $\mathbb{N}$ *is directed but does not have a least upper bound. One can obtain a DCPO* $(\mathbb{N} \cup \{\infty\}, \leq^\infty)$ *by adding the element* ∞ *to* $\mathbb{N}$ *and setting* $\leq^\infty = \leq \cup \{(n, \infty) \mid n \in \mathbb{N} \cup \{\infty\}\}$.

One can combine the two manners in which posets have been given additional structure.

Definition 6. *A Complete Partial Order (CPO) is a DCPO which is also a PPO.*

Example 6. *In the above examples* $(A_\perp, \leq_\perp, \perp)$*, and* $(\mathbb{N} \cup \{\infty\}, \leq^\infty, 0)$ *are CPOs. For the latter, all directed sets* $\emptyset \neq S \subseteq \mathbb{N} \cup \{\infty\}$ *either have a maximum* $m \in S$*, in which case lub* $S = m$*, or they do not have a maximum in* S*, in which case lub* $S = \infty$*.*

As illustrated by the above example, least upper bounds of directed sets sometimes denote "infinite" elements. The opposite notion of "finiteness" is defined as follows.

Definition 7. *In a DCPO* $(C, \leq)$*, an element* c° *is* compact *(or* finite*) if for all directed sets* $S \subseteq C$*, if* $c^\circ \leq$ *lub* S *then there exists* $c \in S$ *such that* $c^\circ \leq c$*.*

That is, if *lub* S is at least as defined as c°, then some $c \in S$ is at least as defined as c°.

Example 7. *The compact elements in* $(\mathbb{N} \cup \{\infty\}, \leq^\infty)$ *are exactly the (finite) natural numbers* $\mathbb{N}$*. The other examples seen so far only have compact elements.*

Notation. In a DCPO $(C, \leq)$, we denote by C° the whole set of compacts of the DCPO and by C°_c the set of compacts less or equal to c, i.e., $C^\circ_c := \{c^\circ \in C^\circ \mid c^\circ \leq c\}$.

Compact elements play an important role in the key notion of *algebraic* DCPO below: they form a "basis", from which all other elements are built using least upper bounds.

Definition 8. *A DCPO* $(C, \leq)$ *is* algebraic *if for all* $c \in C$*, the set* C°_c *is directed, and* $c =$ *lub* C°_c*. A CPO is algebraic if it is algebraic as a DCPO.*

All DCPOs seen so far are algebraic. A non-algebraic DCPO is given in [2], Ex. 1.1.13. Algebraic (D)CPOs can be built by certain operations described in subsequent sections.

2.1.2. Completion

Algebraic DCPOs can be obtained from posets, and algebraic CPOs can be obtained from PPOs, by an operation called *ideal completion*. First, an intermediate definition:

Definition 9. *Given a poset* $(C, \leq)$*, the* downward closure $\downarrow S$ *of a subset* S *of* C *is defined as* $\downarrow S := \{c \in C \mid \exists s \in S.\ c \leq s\}$*. A set* S *is* downwards closed *if* $S = \downarrow S$.

The following definition and lemma are adapted from [2] (Def. 1.1.20, Prop. 1.1.21)[4].

Definition 10. *In a poset* $(C, \leq)$*, an* ideal *is a directed, downwards-closed subset of* C*. Let* $\mathcal{I}_C$ *denote the set of ideals of* C*. An ideal* I *is* principal *if* $I = \downarrow \{x\}$ *for some* $x \in C$ *(which, trivially, is unique). We denote by* $\mathcal{P}_C$ *the set of principal ideals of* C*.*

[4] Up to details: in [2] $K(C)$ is used instead of C°, they use more general *preoders* instead of posets, etc.

Proposition 1. *If $(C, \preceq)$ is a poset then $(\mathcal{I}_C, \subseteq)$ is an algebraic DCPO, called the* ideal completion *of $(C, \preceq)$, whose compact elements are the principal ideals $\mathcal{P}_C$. Moreover, any algebraic DCPO $(C, \preceq)$ is isomorphic with the ideal completion $(\mathcal{I}_{C^\circ}, \subseteq)$.*

The proof, with minor changes corresponding to notations, is that of Prop. 1.1.21 [2].

Remark. Above, $\mathcal{I}_{C^\circ}$ is the set of ideals of the poset of compacts of $(C, \preceq)$, i.e., of the poset $(C^\circ, \preceq^\circ)$ where $\preceq^\circ$ is the restriction of $\preceq$ to C°. Ideal completion applies to PPOs as well, and has the same properties as that for posets: if $(C, \preceq, \perp)$ is a PPO then $(\mathcal{I}_C, \subseteq, \{\perp\})$ is an algebraic CPO whose compacts are exactly the principal ideals $\mathcal{P}_C$; and any algebraic CPO $(C, \preceq, \perp)$ is isomorphic with the ideal completion $(\mathcal{I}_{C^\circ}, \subseteq, \{\perp\})$.

Despite its name ideal completion is not ideal because it produces convoluted (D)CPOs.

Example 8. *Completion by ideals of $(\mathbb{N}, \leq)$ gives $((\bigcup_{n\in\mathbb{N}}\{m \in \mathbb{N} \mid m \leq n\}) \cup \{\mathbb{N}\}, \subseteq)$.*

To obtain more "natural" structures we introduce a notion called *natural completion*.

Definition 11. *For a poset $(C^\circ, \preceq^\circ)$, an algebraic DCPO $(C, \preceq)$ such that $(C^\circ, \preceq^\circ)$ is the poset of compacts of $(C, \preceq)$ is called a* natural completion of $(C^\circ, \preceq^\circ)$. *The notion extends to PPOs: for a PPO $(C^\circ, \preceq^\circ, \perp)$ an algebraic CPO $(C, \preceq, \perp)$ such that $(C^\circ, \preceq^\circ, \perp)$ is the PPO of compacts of $(C, \preceq, \perp)$ is called a* natural completion of $(C^\circ, \preceq^\circ, \perp)$.

Example 9. *The algebraic DCPO $(\mathbb{N} \cup \{\infty\}, \leq^\infty)$ is a natural completion of $(\mathbb{N}, \leq)$. The natural completion of the flat PPO $(A_\perp, \preceq_\perp, \perp)$ is itself, seen as an algebraic CPO.*

Proposition 2. *Any poset has a natural completion, unique up to isomorphism.*

The proof of this proposition is sketched in the Appendix. Natural-completion algebraic CPOs exist (and are unique up to isomorphism) for PPOs as well. Hereafter in the paper we systematically use natural completions instead of ideal completions, We sometimes refer to natural completions simply as *completions*.

2.2. Continuity and Kleene's Fixpoint Theorem

Another interesting feature of completions, which we will be using hereafter, is that they also apply to *morphisms*: morphisms of posets are "completed" to morphisms of DCPOs in a unique way. Morphisms of posets preserve poset structure, i.e., they are *monotonic*. Morphisms of DCPOs preserve DCPO structure, i.e., they are *continuous*:

Definition 12. *If $(D, \precsim)$ and $(C, \preceq)$ are DCPOs, a function $f : D \to C$ is continuous when it is monotonic and for all directed sets $S \subseteq D$, $f\ (lub\ S) = lub\ (f\ S)$.*

Remark. The above definition is sound: we have $f\ S = \{f\ x \mid x \in S\}$ and, since f is monotonic and S is directed in $(D, \precsim)$, $f\ S$ is directed in $(C, \preceq)$, hence, $lub\ (f\ S)$ exists.

Proposition 3. *If $(D^\circ, \precsim^\circ)$ and $(C^\circ, \preceq^\circ)$ are posets having respective natural completions $(D, \precsim)$ and $(C, \preceq)$, then for any monotonic function $f^\circ : D^\circ \to C^\circ$ there exists a unique continuous function $f : D \to C$ such that $f\ d^\circ = f^\circ\ d^\circ$ for all compacts $d^\circ \in D^\circ$. We refer to the function $f : D \to C$ as the (natural)* completion *of $f^\circ : D^\circ \to C^\circ$.*

Proposition 1.1.22 [2] states this result for ideal completion but can readily be adapted to natural completion. Alternatively, adapt Corollary 1.6 from [3, Ch. 3] to DCPOs.

Example 10. *The successor function* $suc^\circ : \mathbb{N} \to \mathbb{N}$ *is monotonic. Hence, its completion* $suc : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ *coincides with* suc° *on* $\mathbb{N}$ *and, by continuity,* $suc\ \infty = \infty$.

Continuous functions play an essential role in *Kleene's fixpoint theorem*, which is used in denotational semantics for defining the semantics of program constructions. A few preliminary notions and observations are required. Consider a function $F : C \to C$. Any $x \in C$ such that $F\ x = x$ is called a *fixpoint* of F. If F is monotonic and $(C, \leq, \bot)$ is a CPO then the sequence $(F^n)_{n\in\mathbb{N}}$ inductively defined by $F^0 = \bot$ and for all $m \in \mathbb{N}$, $F^{m+1} = F(F^m)$ is monotonic. It forms a directed set $\{F^n \mid n \in \mathbb{N}\}$, which has a *lub*.

Proposition 4 (Prop. 1.1.7 in [2])**.** *If* $(C, \leq, \bot)$ *is a CPO and* $F : C \to C$ *is continuous then* F *has the least fixpoint* $\mu F = lub\,\{F^n \mid n \in \mathbb{N}\}$.

Example 11. *We have seen that* $suc : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ *is continuous and that* $(\mathbb{N} \cup \{\infty\}, \leq^\infty, 0)$ *is a CPO. Hence* suc *has the least fixpoint* $lub\,\{n + 1 \mid n \in \mathbb{N}\} = \infty$.

In practice it is tedious to prove continuity from Def. 12 for non-trivial functions. One technique is *continuity-by-construction*, like what we did for *suc* function, which is continuous as the completion of a monotonic function on the compacts of its domain. Continuity by several other constructions is widely used in denotational semantics [4].

2.3. Closure Properties of Algebraic (D)CPOs

Algebraic CPOS are closed under possibly infinite product; algebraic DCPOs are also closed under possibly infinite sums. We define the sum and product operations and illustrate them by defining an algebraic DCPO of lists.

2.3.1. Product

We start by recalling the Cartesian product of an indexed set of sets.

Definition 13. *Given a set* J *of indices and a* J*-indexed set of sets* $\{C_j \mid j \in J\}$*, their Cartesian product* $\Pi_{j\in J} C_j$ *is the set of functions* $\{c : J \to \bigcup_{j\in C} C_j \mid \forall j.\,(c\ j) \in C_j\}$.

Remark. If all the sets C_j are identical copies of some set C, then $\Pi_{j\in J} C_j$ coincides with $J \to C$. In this case the product is simply an *exponentiation* In the corner case $J = \emptyset$, $J \to C$ has exactly one element, the unique function from $\emptyset$ to C.

Example 12. *For all* $n \in \mathbb{N}$*, let* $\{< n\}$ *denote the set* $\{m \in \mathbb{N} \mid m < n\}$*. In particular,* $\{<0\} = \emptyset$*. The set* $list_n\ A$ *of lists of length* $n \in \mathbb{N}$ *over a set* A *is in bijection with* $A^{\{<n\}}$*. In particular, the empty list nil corresponds to the unique function from* $\emptyset$ *to* A.

Definition 14. *In the context of Definition 13, assume that each* C_j *has a distinguished element* $\bot_j$*. The* carrier *of* $c \in \Pi_{j\in J} C_j$*, denoted by* $[\![c]\!]$*, is the set* $\{j \in J \mid c\ j \neq \bot_j\}$.

The result regarding the product of a J-indexed set of algebraic CPOs follows.

Proposition 5. *In the context of Definition 13, if for all* $j \in J$*,* C_j *is organized as an algebraic CPO* $(C_j, \leq_j, \bot_j)$*, then the structure* $(C, \leq, \bot)$ *defined as follows*

- $C = \Pi_{j\in J} C_j$;
- *for all* $c, c' \in C$, $c \preceq c'$ *iff for all* $j \in J, c\ j \preceq_j c'\ j$*:*
- $\bot = (\bot_j)_{j\in J}$

is an algebraic CPO denoted $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$*. The set of compacts of* $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ *is the set* $\{c \in \Pi_{j\in J} C_j^\circ \mid [\![c]\!] \text{ is finite}\}$*, of compact-valued functions with finite carriers.*

Remark. To our best knowledge this result is new. It can be summarized as the statement *"*$\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ *is a natural completion of the PPO* $\{c \in \Pi_{j\in J} C_j^\circ \mid [\![c]\!] \text{ is finite}\}$*"*. It is a key result because it occurs in Section 3 in our construction of coinductive types.

Example 13. *If* $(A, \preceq, \bot)$ *is an algebraic CPO then, for all* $n > 0$*, by Prop. 5* $A^{\{<n\}}$ *is an algebraic CPO. Its compacts are the functions from* $\{< n\}$ *to* A° *(which do have finite carriers). The bijection between* $A^{\{<n\}}$ *and* $\text{list}_n A$ *noted in Example 12 induces an algebraic CPO structure on* $\text{list}_n A$*: the order is pointwise, the bottom is a list of length n of* $\bot$*, and the compact elements are lists of length n over* A°*. For* $n = 0$*, the one-element* $A^{\{<0\}}$ *in not naturally an algebraic CPO. The unique function from* $\emptyset$ *to A is a defined object, hence, it cannot also play the role of the bottom element, which is interpreted as undefined.* $A^{\{<0\}}$ *is more naturally interpreted as an algebraic DCPO.*

The proof of Prop. 5 uses the following definition and lemma, which are also used elsewhere in the paper; hence, we state them here and leave proofs in the Appendix.

Definition 15. *In the context of Prop. 13, for any* $S \subseteq C = \Pi_{j\in J} C_j$ *and* $j \in J$ *we define the* projection of S on j *to be the set* $S \Downarrow j = \{c\ j \in C_j \mid c \in S\}$.

Remark. The projection $S \Downarrow j$ is the value at j of functions in the set S. Alternatively, $S \Downarrow j = \{c_j \in C_j \mid \forall i \in J \setminus \{j\}.\ \exists c_i \in C_i.\ \lambda\, i \to c_i \in S\}$: that is, $c_j \in C_j$ and together with other values $c_i \in C_i$, for $i \in J \setminus \{j\}$, c_j creates a function $\lambda\, i \to c_i \in S$.

The following lemma builds the *lub* of a directed set of functions as the function that to each index associates the *lub* of the projection of the directed set on that index.

Lemma 1. *For any directed set* $S \subseteq C = \Pi_{j\in J} C_j$ *and* $j \in J$*,* $S \Downarrow j$ *is directed, and* $lub\ S = \lambda\, j \to (lub\ (S \Downarrow j))$.

The lemma (especially, the version for exponentials $J \to C$) is a key helper in Section 4 for our construction of (co)recursive functions, because it expresses *lub*s in $J \to C$, which is a CPO of functions, in terms of *lub*s in the simpler CPO C.

2.3.2. Sum

We define a possibly infinite sum. Unlike the product, the sum has no natural $\bot$ element. Hence both the summands and the result shall be algebraic DCPOs.

Definition 16. *Given a set J of indices and a J-indexed set of mutually-disjoint sets* $\{C_j \mid j \in J\}$ *the* sum $\Sigma_{j\in J} C_j$ *is the union of sets* $\bigcup_{j\in J} C_j$.

Example 14. *The set of lists over a set A, denoted by list A, is in bijection with* $\Sigma_{n\in\mathbb{N}} A^{\{<n\}}$.

Proposition 6. *In the context of Definition 16, if for all $j \in J$, C_j is organized as an algebraic DCPO $(C_j, \preceq_j)$, then the structure $(C, \preceq)$ defined as follows*

- $C = \Sigma^{\perp}_{j\in J} C_j$;
- $\preceq = \bigcup_{j\in J} (\preceq_j)$

is an algebraic DCPO denoted by $\Sigma_{j\in J}(C_j, \preceq_j)$. Its set of compacts is $(\bigcup_{j\in J} C_j^{\circ})$.

The proof is trivial, since $\Sigma_{j\in J}(C_j, \preceq_j)$ is essentially $|J|$ separate algebraic DCPOs.

Example 15. *if $(A, \preceq)$ is an algebraic DCPO then, by Prop. 5 and 6, $\Sigma_{n\in\mathbb{N}} A^{\{<n\}}$ is arganized an algebraic DCPO. The bijection noted in Example 14 between $\Sigma_{n\in\mathbb{N}} A^{\{<n\}}$ and list A organizes the latter as an algebraic DCPO isomorphic to $\Sigma_{n\in\mathbb{N}} A^{\{<n\}}$, whose compact elements are $\{nil\} \cup \bigcup_{n\geq 1} list_n\, A^{\circ}$.*

2.4. Knaster-Tarski's Theorem and Coinduction

Knaster-Tarski's fixpoint theorem gives different conditions than Kleene's fixpoint theorem for the existence of fixpoints. We are interested in greatest fixpoints in order to obtain coinductive predicates and coinductive proofs. The following material is partially adapted from [4]. For a set Q we denote by 2^Q the set of subsets of Q.

Definition 17. *A* complete lattice *is a poset $(L, \preceq)$ such that lub S exists for all $S \subseteq L$.*

Example 16. *If Q is a set then $(2^Q, \subseteq)$ with lub $S = \bigcup_{T\in S} T$ is a complete lattice.*

Knaster-Tarski's fixpoint theorem has a broader scope. We focus on greatest fixpoints.

Proposition 7 (Th. 5.16 in [4])**.** *Let $(L, \preceq)$ be a complete lattice. Let $F : L \to L$ be a monotonic function. Define $\nu F = lub\,\{x \in L \mid x \preceq f\ x\}$. Then νF is a fixpoint of F and the greatest post-fixpoint of f. (A post-fixpoint is an element x such that $x \preceq F\ x$.)*

Remark. It follows that νF in the above theorem is actually the greatest fixpoint of F.

The following elaborations on Prop. 7 are used for coinductive definitions and proofs. Consider the complete lattice $(2^Q, \subseteq)$ from Example 16, a monotonic function $F : 2^Q \to 2^Q$, and its greatest fixpoint $\nu F \subseteq Q$. We say that F is the *functional* for νF, and that νF is *coinductively defined* by F. The equation $\nu F = F(\nu F)$ is called the *unfolding equation of νF*. The fact that νF is the greatest post-fixpoint of F, rewritten as: *for all $P \subseteq Q$, $P \subseteq F\ P$ implies $P \subseteq \nu F$* is called the *coinduction principle* for νF.

Remark. By the natural identification of sets and their characteristic predicates, F can be seen as a predicate transformer and νF can be seen as a predicate. Such predicates can have arbitary arities. Predicates of arity greater than 1 are usually called "relations". Hence Knaster-Tarski's theorem enables us to define coinductive relations and proofs.

3. Constructing Coinductive Types

Corecursive functions produce values in coinductive types. In this section we show how coinductive types can be constructed from inductive types endowed with definition orders, using the completion and closure operations from the previous section.

The resulting types are kin to (unary) *containers* [15], a rich class of types that include all *strictly positive* types built with constants, sums, products, exponentiation by sets, and fixpoints. There are, however, differences: the main one is that our types are inhabited by possibly *partially-defined* terms, ordered by a definition order. Such partial terms are, in particular, results of partial corecursive functions.

For partial *recursive* functions the situation is simpler: they produce values in flat algebraic CPOs, which, as seen in Example 9, are natural completion of themselves seen as PPOs; the results in this section are redudant for partial recursive functions.

For the sake of simplicity we present the constructions in a set-theoretical setting.

3.1. Basic Definitions and Examples

Definition 18. *Given a set A of* shapes *and B a function that for each $a \in A$ produces a set $(B\ a)$ of* positions*, the* finite partial container *(FPC) C° parameterized by A and B is the set inductively defined by the rules: $\bot \in C^\circ$, and, for all $a \in A$ and $f : (B\ a) \to C^\circ$ such that the carrier $\llbracket f \rrbracket$ is finite, $\mathrm{node}^\circ_a\ f \in C^\circ$.*

Elements of FPCs can be seen as finite trees: in depth due to their inductive nature, and in breadth because finitely many subtrees (generated by node°_a) are different from $\bot$.

Definition 19. *The definition order $\preceq^\circ$ on C° is inductively defined by the rules $\bot \preceq^\circ$ for all $c^\circ \in C^\circ$, and $\mathrm{node}^\circ_a\ f \preceq^\circ \mathrm{node}^\circ_a\ f'$ whenever for all $b \in (B\ a)$, $f\ b \preceq^\circ f'\ b$.*

Remark. The reflexivity, transitivity, and antisymmetry of $\preceq^\circ$ are proved by induction. Since $\bot$ is least with respect to $\preceq^\circ$, we obtain that the triple $(C^\circ, \preceq^\circ, \bot)$ is a PPO.

Two examples of PPOs isomorphic to finite-partial-container PPOs are given below. The symbols $\preceq^\circ$ and $\bot$ are overloaded; their meaning can be inferred from the context.

Example 17. *Consider the set $\mathrm{stream}^\circ\ A$ of finite approximations of streams over a set A, inductively defined by $\bot \in \mathrm{stream}^\circ\ A$ and $\mathrm{cons}^\circ\ a\ s^\circ \in \mathrm{stream}^\circ\ A$ whenever $a \in A$ and $s^\circ \in \mathrm{stream}^\circ\ A$. An order $\preceq^\circ$ is inductively defined by $\bot \preceq^\circ s^\circ$ for all $s^\circ \in \mathrm{stream}^\circ\ A$ and $\mathrm{cons}^\circ\ a\ s^\circ \preceq^\circ \mathrm{cons}^\circ\ a\ s'^\circ$ whenever $s^\circ \preceq^\circ s'^\circ$. Hence $(\mathrm{stream}^\circ\ A, \preceq^\circ, \bot)$ is a PPO.*

We present a finite partial container isomorphic to $\mathrm{stream}^\circ\ A$: Let C° be defined by the set of shapes A (same as the one in $\mathrm{stream}^\circ\ A$), and, for all $a \in A$, the set $B\ a$ equals $\{*\}$, the singleton set. Hence C° has the constructors $\bot$ and $\mathrm{node}^\circ_a\ f$, for all $a \in A$ and functions $f : \{*\} \to C^\circ$ (which, obviously, have finite carriers). We use the implicit value "$\_$", when it can be inferred from the context, e.g., we write $(f\ \_)$ for $(f\ *)$.

We then define $\mathrm{s2c}^\circ : \mathrm{stream}^\circ\ A \to C^\circ$ by $\mathrm{s2c}^\circ\ \bot = \bot$ and $\mathrm{s2c}^\circ\ (\mathrm{cons}^\circ\ a\ s^\circ) = \mathrm{node}^\circ_a\ (\lambda\ \_ \to \mathrm{s2c}^\circ\ s^\circ)$; and $\mathrm{c2s}^\circ : C^\circ \to \mathrm{stream}^\circ\ A$ by $\mathrm{c2s}^\circ\ \bot = \bot$ and $\mathrm{c2s}^\circ\ (\mathrm{node}^\circ_a\ f) = \mathrm{cons}^\circ\ a\ (\mathrm{c2s}^\circ\ (f\ \_))$. We prove by induction that $\mathrm{s2c}^\circ$ and $\mathrm{c2s}^\circ$ are monotonic and inverse to each other. Since they also preserve $\bot$, $\mathrm{s2c}^\circ : \mathrm{stream}^\circ\ A \to C^\circ$ and $\mathrm{c2s}^\circ : C^\circ \to \mathrm{stream}^\circ\ A$ define an isomorphism of PPOs between $\mathrm{stream}^\circ\ A$ and C°.

Example 18. *Let $\mathrm{len}\ l$ denote the length of a list l and l_i, for $i < \mathrm{len}\ l$, be the i-th element of l. The set T° of finite Rose trees is inductively defined by $\bot \in T^\circ$ and, for all $l \in \mathrm{list}\ T^\circ$, $\mathrm{tree}^\circ\ l \in T^\circ$. The order $\preceq^\circ$ is inductively defined by $\bot \preceq^\circ$ for all $t^\circ \in T^\circ$, and $\mathrm{tree}^\circ\ l \preceq^\circ \mathrm{tree}^\circ l'$ whenever $\mathrm{len}\ l = \mathrm{len}\ l'$ and for all $i < \mathrm{len}\ l$, $l_i \preceq^\circ l'_i$,*

Consider the finite partial container C° having set of shapes $A = \mathbb{N}$ and position-function B that maps $n \in \mathbb{N}$ to the set $\{< n\}$ of natural numbers less than n.

A PPO isomorphism between $(T^\circ, \preceq^\circ, \bot)$ and $(C^\circ, \preceq^\circ, \bot)$ is given by the functions:

$t2c^\circ : T^\circ \to C^\circ$ defined by
$t2c^\circ \bot = \bot$, and $t2c^\circ\ (tree^\circ\ l) = node^\circ_{(len\ l)}\ (\lambda\, i : \{< (len\ l)\} \to (t2c^\circ\ l_i))$

and (using the map function on lists and $[0, \ldots,\ n-1]$ the list of the first n naturals):

$c2t^\circ : C^\circ \to T^\circ$ defined by
$c2t^\circ \bot = \bot$ and $c2t^\circ\ (node^\circ_{(len\ l)}\ f) = tree^\circ\ (map\ c2t^\circ\ (map\ f\ [0, \ldots,\ (len\ l) - 1\ ]))$.

3.2. Using Natural Completion

The natural completion operation (Definition 11) eliminates, in some sense, finiteness from FPCs. Partiality remains because $\bot$ is not eliminated.

Definition 20. *Assume $(C^\circ, \preceq^\circ, \bot)$ is a Finite Partial Container organized as a PPO. We call any natural completion $(C, \preceq, \bot)$ of $(C^\circ, \preceq^\circ, \bot)$ a* Partial Container *(PC).*

Natural completion also applies to certain functions (cf. Prop. 3). We are here interested in the constructors $node^\circ_a : ((B\ a) \to C^\circ) \to C^\circ$ with $a \in A$ (cf . Def. 18 of FPCs).

Notation. Given an FPC C° parameterized by shapes A and positions B, and $a \in A$, we denote by F°_a the set $\{f^\circ : (B\ a) \to C^\circ \mid \|f^\circ\| \text{ is finite}\}$. A relation $\sqsubseteq^\circ$ on F°_a is defined by $f^\circ \sqsubseteq^\circ f'^\circ$ iff for all $b \in (B\ a)$, $f\ b \preceq^\circ f'\ b$, with $\preceq^\circ \subseteq C^\circ \times C^\circ$ from Def. 19. Let $\bot_a$ be $(\lambda\,(\_ : (B\ a)) \to \bot) \in F^\circ_a$ where $\bot \in C^\circ$ is the constructor of C° from Def.18.

Remark. With the above notations, for all $a \in A$, $(F^\circ_a, \sqsubseteq_a, \bot_a)$ is a PPO. Going back to the constructor $node^\circ_a$ of C°, with the above notations it holds that for all $a \in A$, $node^\circ_a : F^\circ_a \to C^\circ$ is a monotonic function between the posets $(F^\circ_a, \sqsubseteq_a)$ and $(C^\circ, \preceq^\circ)$.

Notation. If C° is an FPC with shapes A and positions B, and the PC $(C, \preceq, \bot)$ is the natural completion of $(C^\circ, \preceq^\circ, \bot)$, cf. Def. 20, for all $a \in A$, we let F_a denote the set $(B\ a) \to C$. A relation $\sqsubseteq_a$ on F_a is defined by $f \sqsubseteq f'$ iff for all $b \in (B\ a)$, $f\ b \preceq f'\ b$.

Remark. By Prop. 5, $(F_a, \sqsubseteq_a, \bot_a)$ is an algebraic CPO whose PPO of compacts is $(F^\circ_a, \sqsubseteq^\circ_a, \bot_a)$. According to Def. 11 $(F_a, \sqsubseteq_a, \bot_a)$ is a natural completion of $(F^\circ_a, \sqsubseteq^\circ_a, \bot_a)$.

Definition 21. *In the context of the above notations and remarks: for all $a \in A$, we define $node_a : F_a \to C$ as the natural completion (cf. Prop. 3) of $node^\circ_a : F^\circ_a \to C^\circ$.*

In other words $node_a : F_a \to C$ is the unique continuous extension of $node^\circ_a : F^\circ_a \to C^\circ$.

3.3. Completion of Constructors are Constructors of Completions

Next, we show that if $(C, \preceq, \bot)$ is a completion of $(C^\circ, \preceq^\circ, \bot)$, then $\bot$ and the completions $node_a : F_a \to C$ of the constructors $node^\circ_a : F^\circ_a \to C^\circ$ of C° (cf. Def. 18) behave like constructors for C. We emphasize that, being a defined function, $node_a$ is not an *actual* constructor for C; it only behaves like one: that is, for each $c \in C$, either $c = \bot$ or (exclusively) there exist unique $a \in A$ and $f \in F_a$ such that $c = node_a\ f$.

This is achieved in several steps. The first step is a lemma about *lub*s, which is used in many situtations below in the paper. Its proof follows directly from definitions.

Lemma 2. *Assume a poset $(D, \preceq)$ and sets $S, S' \subseteq D$ such that lub S and lub S' exist.*
(i) if for all $s \in S$ there exists $s' \in S'$ such that $s \preceq s'$ then lub $S \preceq$ lub S';

(ii) if every $s \in S$ is compact (Def. 7) then the reciprocal also holds: lub $S \preceq$ lub S' implies that for all $s \in S$ there exists $s' \in S'$ such that $s \preceq s'$.

Below, $(C^\circ, \preceq^\circ, \bot)$ is an FPC, $(C, \preceq, \bot)$ is a PC that completes $(C^\circ, \preceq^\circ, \bot)$, and $node_a : F_a \to C$ is the natural completion of the constructor $node_a^\circ : F_a^\circ \to C^\circ$ of $(C^\circ, \preceq^\circ, \bot)$.

Remark. Hereafter in proofs, instead of "$f^\circ \in F_a^\circ$ *such that* $f^\circ \sqsubseteq_a f$" we simply write "$f^\circ \sqsubseteq_a f$", The implicit fact $f^\circ \in F_a^\circ$ is inferred from the $^\circ$ exponents on f° and F_a°, indicating compactness, and the $\sqsubseteq_a$ relation, indicating which $a \in A$ is involved in F_a°.

Lemma 3. *For all $a, a' \in A$, $f \in F_a$ and $f' \in F_{a'}$, $node_a\ f \preceq node_{a'}\ f'$ if and only if $a = a'$ and $f \sqsubseteq_a f'$.*

Proof. ($\Rightarrow$) Since $(F_a, \sqsubseteq_a, \bot_a)$ is an algebraic CPO, $f = lub\{f^\circ \in F_a^\circ \mid f^\circ \sqsubseteq_a f\}$. Similarly, $f' = lub\{f'^\circ \in F_{a'}^\circ \mid f'^\circ \sqsubseteq_{a'} f'\}$. Since $node_a$ and $node_{a'}$ are continuous and are completions of, respectively, $node_a^\circ : F_a^\circ \to C^\circ$ and $node_{a'}^\circ : F_{a'}^\circ \to C^\circ$, we obtain that $S := \{node_a^\circ\ f^\circ \mid f^\circ \sqsubseteq_a f\}$ and $S' := \{node_{a'}^\circ\ f'^\circ \mid f'^\circ \sqsubseteq_{a'} f\}$ are directed, and $node_a\ f = lub\ S$ and $node_{a'}\ f' = lub\ S'$. From the hypothesis $node_a\ f \preceq node_{a'}\ f'$ and Lemma 2 (ii) we obtain that for all $f^\circ \sqsubseteq_a f$, there exists $f'^\circ \sqsubseteq_{a'} f'$ such that $node_a^\circ\ f^\circ \preceq^\circ node_{a'}^\circ\ f'^\circ$. (Here we have used the fact that $\preceq$ is $\preceq^\circ$ on compacts.) But by Def. 19 this implies $a = a'$ and for all $b \in B\ a$, $f\ b \preceq f'\ b$, i.e., $f \sqsubseteq_a f'$.

($\Leftarrow$) This implication holds by the continuity, hence, the monotonicity of $node_a$. □

As a corollary to Lemma 3 we obtain that the function $node : \forall(a : A), F_a \to C$, where F_a depends on a, is injective in both arguments:

Corollary 1. *$node_a\ f\ = node_{a'}\ f'$ implies $a = a'$ and $f = f'$.*

We next show that the function $node : \forall(a : A), F_a \to C$ is surjective on $C \setminus \{\bot\}$.

Lemma 4. *For all $c \in C \setminus \{\bot\}$, there exist $a \in A$ and $f \in F_a$ such that $c = node_a\ f$.*

Proof. Let $S_c^\circ := \{c^\circ \in C^\circ \mid c^\circ \preceq c\}$. By algebraicity of C, S_c° is directed and $c = lub\ S_c^\circ$. Since $c \neq \bot$, S_c° contains at least a compact different from $\bot$, and $S_c'^\circ := S_c^\circ \setminus \{\bot\}$ is still directed, and $c = lub\ S_c'^\circ$ still holds because $\bot$ does not matter when computing the least upper bound. Since $\bot \notin S_c'^\circ$, for each $c^\circ \in S_c'^\circ$, by Def. 18 of FPCs, there exists $a \in A$ and $f^\circ \in F_a^\circ$ such that $c^\circ = node_a^\circ\ f^\circ$. Now, assuming there exist $c^\circ, c'^\circ \in S_c'^\circ$ with $c^\circ = node_a^\circ\ f^\circ$, $c'^\circ = node_{a'}^\circ\ f'^\circ$ and $a \neq a'$: from the directedness of $S_c'^\circ$ there exists $c''^\circ = node_{a''}^\circ\ f^\circ$ such that $node_a^\circ\ f^\circ \preceq^\circ node_{a''}^\circ\ f''^\circ$ and $node_{a'}^\circ\ f'^\circ \preceq^\circ node_{a''}^\circ\ f''^\circ$, which by Def. 19 implies $a'' = a' = a$, in contradiction with the assumed $a \neq a'$.

Hence, for all $c^\circ \in S_c'^\circ$ there exists a *unique* $a \in A$ and some $f^\circ \in F_a^\circ$ such that $c^\circ = node_a^\circ\ f^\circ$. Let now $F^\circ := \{f^\circ \in F_a^\circ \mid \exists c^\circ \in S_c'^\circ\ s.t.\ c^\circ = node_a^\circ\ f^\circ\}$ be the projection of $S_c'^\circ$ on F_a°. The set F° is directed, due to the directedness of $S_c'^\circ$ and monotonicity of $node_a^\circ$. Moreover by construction of F° as the projection of $S_c'^\circ$ on F_a°, we have $S_c'^\circ = \{node_a^\circ\ f^\circ \mid f^\circ \in F^\circ\}$, hence, $c = lub\ S_c'^\circ = lub\ \{node_a^\circ\ f^\circ \mid f^\circ \in$

$F^\circ\} = node_a\ (lub\ \{f^\circ \mid f^\circ \in F^\circ\}) = node_a\ (lub\ F^\circ)$, where we have used the continuity of $node_a$. Overall, we have obtained that there do exist $a \in A$ and $f := lub\ F^\circ$ with $F^\circ \subseteq F_a^\circ$ (thus, $f \in F_a$) such that $c = node_a\ f$; which proves the lemma. □

Hence, $node : \forall(a : A), f : F_a \to C$, or, equivalently, the set of functions $node_a : F_a \to C$ where a ranges over A, act as constructors of $C \setminus \{\bot\}$. The other constructor is $\bot$.

Lemma 5. *For all $a \in A$ and $f \in F_a$, $node_a\ f \neq \bot$.*

Proof. Assuming $node_a\ f = \bot$ we obain $node_a\ f \preceq \bot$, hence, by continuity of $node_a$ and the fact that on compacts it equals $node_a^\circ$, $node_a\ f = lub\,\{node_a^\circ\ f^\circ \mid f^\circ \sqsubseteq_a f\} \preceq \bot$, hence, for all $f^\circ \sqsubseteq_a f$, $node_a^\circ\ f^\circ \preceq^\circ \bot$, in contradiction to Def. 19 of the order $\preceq^\circ$. □

By combining Lemmas 3, 4 and 5 we obtain a characterization of the elements of C:

Theorem 1. *For all $c \in C$, $c = \bot$ or (exclusively) there exist unique $a \in A$ and $f \in F_a$ such that $c = node_a\ f$.*

This shows that elements of C are trees of possibly infinite depth and arbitrary breadth.

Example 19. *Consider $(stream\ A, \preceq, \bot)$ that naturally completes $(stream^\circ\ A, \preceq^\circ, \bot)$ from Example 17[5]. Then, using Theorem 1 we obtain that for all $s \in stream\ A$, either $s = \bot$ or (exclusively) there exist unique $a \in A$ and $s' \in A$ such that $s = cons\ a\ s'$, where $(cons\ a) : stream\ A \to stream\ A$ is the continuous natural completion of the monotonic $(cons^\circ\ a) : stream^\circ\ A \to stream^\circ\ A$. The reasoning can be repeated for s', etc,... leading to possibly infinite (i.e., coinductive) streams over A.*

Example 20. *Let $(T, \preceq, \bot)$ be a natural completion of $(T^\circ, \preceq^\circ, \bot)$ from Example 18. Using Th. 1 we obtain that for all $t \in T$, either $t = \bot$ or (exclusively) there is a unique list $l \in list\ T$ such that $t = tree\ l$, where $tree : list\ T \to T$ is the continuous natural completion of the monotonic $tree^\circ : list\ T^\circ \to T^\circ$. The reasoning can be repeated for all elements of l,... leading to possibly infinite-depth (i.e., coinductive) Rose trees.*

3.4. Coinduction: Total Elements and Equivalence of Bisimulation and Equality

Additional evidence regarding the coinductive nature of Partial Containers is provided by the ability, formalized below, to reason by coinduction about their elements.

We use the Knaster-Tarski theorem (cf. Section 2.4) to coinductively define a predicate characterizing *total* elements in PCs, meaning terms that do not contain $\bot$ as a subterm. The associated coinduction principle is then used for proving the totality of corecursive functions. We also define a bisimulation relation of PCs and prove that it is equivalent to equality. Bisimulation is easier to prove than equality thanks to its coinduction principle; we later exploit this fact when proving equalities.

[5]The actual completion is performed on an FPC isomorphic to $(stream^\circ\ A, \preceq^\circ, \bot)$, Including explicit isomorphisms ruins readability; we shall tacitly be using definitions, theorems and proofs up to isomorphisms.

3.4.1. Totality

Definition 22. *Assume the PC* $(C, \preceq, \bot)$ *is a completion of the FPC* $(C^\circ, \preceq^\circ, \bot)$ *having shapes A and positions B. The* functional *Total* $: 2^C \to 2^C$ *of totality is defined by: for all* $S \subseteq C$, *Total* $S = \{node_a\ f \in C \mid \forall b \in (B\ a), f\ b \in S\}$.

Remark. *Total* $: 2^C \to 2^C$ is monotonic with respect to $\subseteq$. By the Knaster-Tarski theorem it has a greatest fixpoint $\nu T\, total$, which we denote by *total*. Hence (cf. Section 2.4):

- unfolding equation: $total = \{node_a\ f \in C \mid \forall b \in (B\ a), f\ b \in total\}$;
- coinduction principle: for all $T \subseteq C$, if $T \subseteq (Total\ T)$ then $T \subseteq total$. Equivalently: we say $t \in T$ is total if $t \in total$. To prove that t is total, find $T \subseteq C$ with $t \in T$ and prove $\forall t' \in T, \exists a \in A, \exists f : (B\ a) \to C, t' = node_a\ f \wedge \forall b \in (B\ a), f\ b \in T$.

The given definition of totality captures the intuition that a term does not contain $\bot$.

Example 21. *Assume* $(stream\ A, \preceq, \bot)$ *completes* $(stream^\circ\ A, \preceq^\circ, \bot)$ *as in Example 19. Then, using the above technique it can be proved that the total streams are exactly those in* $(stream\ A) \setminus (stream^\circ\ A)$. *Indeed, those are the streams that do not contain* $\bot$.

By contrast, if $(T, \preceq, \bot)$ *completes* $(T^\circ, \preceq^\circ, \bot)$ *as in Example 20, then there are total Rose trees in* T° *- for example,* $tree^\circ$ [] *where* [] *is the empty list; and there are non-total, i.e, partial Rose trees in* $T \setminus T^\circ$ *- e.g., tree* $[t, \bot]$ *where t is any tree in* $T \setminus T^\circ$.

The notion of totality is lifted from terms to functions between partial containers:

Definition 23. *Assume PCs* $(C, \preceq_C, \bot_C)$ *and* $(D, \preceq_C, \bot_C)$. *A function* $f : C \to D$ *is total it maps any total element of C a total element of D.*

3.4.2. Bisimulation

We start by defining a coinductive version $\precsim$ of the order $\preceq$ in the PC $(C, \preceq, \bot)$, which, as before, is a completion of the FPC $(C^\circ, \preceq^\circ, \bot)$ having shapes A and positions B.

Definition 24. $F^\precsim : 2^{C\times C} \to 2^{C\times C}$ *is defined by: for all* $R \subseteq C \times C$, $F^\precsim\ R := \{(c, c') \mid c = \bot \vee (\exists a\ f\ f',\ c = node_a\ f \wedge c' = node_a\ f' \wedge \forall b \in (B\ a),\ ((f\ b), (f'\ b)) \in R)\}$.

Remark. The functional $F^\precsim$ is monotonic with respect to $\subseteq$ in $C \times C$. By the Knaster-Tarski theorem it has a greatest fixpoint $\nu F^\precsim$, denoted by $\precsim$. Hence (cf. Section 2.4):

- unfolding equation (we sometimes write $c \precsim c'$ instead of $(c, c') \in \precsim$):
  $\precsim = \{(c, c') \mid c = \bot \vee \exists a\ f\ f', c = node_a\ f \wedge c' = node_a\ f' \wedge \forall b \in (B\ a), (f\ b) \precsim (f'\ b)\}$.
- coinduction principle: for all $R \subseteq C \times C$, if $R \subseteq (F^\precsim\ R)$ then $R \subseteq \precsim$. Or, equivalently: to prove $c \precsim c'$, find $R \subseteq C \times C$ with $(c, c') \in R$ and prove that for all $(u, v) \in R$, $u = \bot$ or $u = node_a\ f, v = node_a\ f'$ such that for all $b \in (B\ a), ((f\ b), (f'\ b)) \in R$.

One application of the coinduction principle for $\precsim$ is that the order $\preceq$ on C implies $\precsim$:

Lemma 6. *For all* $c, c' \in C$, $c \preceq c'$ *implies* $c \precsim c'$.

Proof. In the coinduction principle for $\precsim$ we choose $R := \preceq$ and prove that for all $c, c' \in C$, $c \preceq c'$ implies $c = \bot$ or there exist $a \in A$, $f, f' \in F_a$ such that $c = node_a\ f$, $c' = node_a\ f'$, and for all $b \in (B\ a)$, $f\ b \preceq f'\ b$. But this is just a a consequence of Theorem 1 on the structure of C and Lemma 3 on the order $\preceq$. □

For the reverse implication, which is he hard part, we use a proof by induction on elements of the FPC C°.

Lemma 7. *For all $c, c' \in C$, if $c \precsim c'$ then $c \preceq c'$.*

Proof. We first prove the following intermediary statement:

$(*)$: for all $c^\circ \in C^\circ$,
(for all $c, c' \in C$, if (i) $c \precsim c'$ and (ii) $c^\circ \preceq c$ then there exists $c'^\circ \in C^\circ$ with
(iii) $c'^\circ \preceq c'$ and (iv) $c^\circ \preceq^\circ c'^\circ$).

The proof is by induction on $c^\circ \in C^\circ$. In the base case, $c^\circ = \bot$ and this case is settled by choosing $c'^\circ = \bot$.

For the inductive step, $c^\circ = node^\circ_a\ f^\circ$ for some $f^\circ \in F^\circ_a$, and then the hypothesis (ii) $c^\circ \preceq c$ implies $c = node_a\ f$ such that for all $b \in (B\ a)$, (ii') $(f^\circ\ b) \preceq (f\ b)$. In particular $(f^\circ\ b) \in C^\circ$. Next, the hypothesis (i) $c \precsim c'$ implies that $c' = node\ a\ f'$ such that for all $b \in (B\ a)$, (i') $(f\ b) \precsim (f'\ b)$. Fix an arbitrary $b \in (B\ a)$. We can now apply the inductive hypothesis to $c^\circ := (f^\circ\ b)$, with $c := (f\ b)$ and $c' := (f'\ b)$; then, (ii') $(f^\circ\ b) \preceq (f\ b)$ corresponds to the hypothesis (ii) and (i') $(f\ b) \precsim (f'\ b)$ corresponds to the hypothesis (i). Hence using the inductive hypothesis, there exists $c'^\circ_b \in C^\circ$ such that (iii') $c'^\circ_b \preceq (f'\ b)$ and (iv') $(f^\circ\ b) \preceq^\circ c'^\circ_b$. Since $b \in (B\ a)$ was chosen in an arbitrary way, from properties of $\preceq$ we obtain $(node^\circ_a\ (\lambda\, b \to c'^\circ_b)) \preceq node_a\ f' = c'$, and from properties of $\preceq^\circ$ we obtain $c^\circ = (node^\circ_a\ f^\circ) \preceq^\circ (node^\circ_a\ (\lambda\, b \to c'^\circ_b))$. Setting $c'^\circ := node^\circ_a\ (\lambda\, b \to c'^\circ_b) \in C^\circ$, we have just obtained (iii) $c'^\circ \preceq c'$ and (iv) $c^\circ \preceq^\circ c'^\circ$, which proves the inductive step of $(*)$ and $(*)$ as a whole.

Now $(*)$ is equivalently reformulated by moving the top *for all* $c^\circ \in C^\circ$ next to the first occurence of c°:

For all $c, c' \in C$, if $c \precsim c'$ then
(for all $c^\circ \in C^\circ$, if $c^\circ \preceq c$ then there exists $c'^\circ \in C^\circ$ with $c'^\circ \preceq c'$ and $c^\circ \preceq^\circ c'^\circ$).

The statement between parentheses is equivalent, by Lemma 2, to the $\preceq$ ordering $lub\,\{c^\circ \in C^\circ \mid c^\circ \preceq c\} \preceq lub\{c'^\circ \in C^\circ \mid c'^\circ \preceq c'\}$. Since C is an algebraic CPO the latter amounts to $c \preceq c'$; which proves our lemma. □

By combining Lemmas 6 and 7 we obtain the equivalence between $\precsim$ and $\preceq$:

Lemma 8. *For all $c, c' \in C$, $c \precsim c'$ if and only if $c \preceq c'$.*

On to bisimulation: it can be defined using $\precsim$ and the opposite $\succsim = \{(c, c') \mid (c', c) \in \precsim\}$:

Definition 25. *The bisimulation relation $\approx \subseteq C \times C$ is defined by $\approx := \precsim \cap \succsim$.*

Remark. Bisimulation $\approx$ has not been defined directly by coinduction, but, based on the unfolding equation and coinduction principle of $\precsim$, we obtain:

- unfolding equation for bisimulation: $\approx = \{(c, c') \mid c = c' = \bot \vee \exists a \in A, \exists f, f' \in F_a, c = node_a\ f \wedge c' = node_a\ f' \wedge \forall b \in (B\ a), (f\ b) \approx (f'\ b)\}$.
- coinduction principle for bisimulation: to prove $c \approx c'$, find $R \subseteq C \times C$ with $(c, c') \in R$ and prove that for all $(u, v) \in R$, $u = v = \bot$ or $u = node_a\ f$, $v = node_a\ f'$ for some $a \in A$ and $f, f' \in F_a$ such that for all $b \in (B\ a)$, $((f\ b), (f'\ b)) \in R$.

By using Lemma 8, we prove the equivalence between bisimulation and equality:

Theorem 2. *For all $c, c' \in C$, $c \approx c'$ if and only if $c = c'$.*

4. Defining (Co)Recursive Functions as Fixpoints

We now provide tools for defining (co)recursive functions operating on the types and using the constructors introduced in the previous section.
Assume one wants to define a function $f : A \to B$. A tentative approach for doing this is by using the *functional*: $F : (A \to B) \to (A \to B)$[6], a description of the "body" of the function f of interest, in which all self-calls to f are replaced by calls to the argument of F. Then, f could tentatively be defined as a solution of the fixpoint equation $f = Ff$.

However, this does not qualify as an actual definition. The fixpoint equation may have zero, or more than one solutions, and some solutions may be partial functions.

Dealing with these issues requires (at least) to organize the codomain of f as a PPO $(B, \leq, \bot)$ in order to encode partial values. Hence $A \to B$ with the pointwise order and bottom value $\lambda\_ \to \bot$ is also a PPO. And we can now *uniquely* define f as the *least* solution (if such a solution exists) of the equation $f = Ff$, which intuitively means that f is defined exactly as much as F "intends" to define it. Not less, and not more, because then f would not be the least solution to the equation, or would be no solution.

Conditions under which $f = F\ f$ has a least solution are given by the following corollary to Kleene's theorem (Prop. 4 pg. 7): B needs to be a CPO, and F needs to be continuous as a function from the poset (actually, a CPO) $(A \to B)$ to $(A \to B)$ itself.

Corollary 2. *If A is a set and B is a CPO, and $F : (A \to B) \to A \to B$ is continuous, then F has the least fixpoint $\mu F = lub\,\{F^n \mid n \in \mathbb{N}\}$, where $F^0 = \lambda\_ \to \bot$.*

Let us examine the condition for applying Corollary 2 of Kleene's theorem. First, the intended function's codomain B must be a CPO. This is not a problem in our setting: the completion operation produces *algebraic* CPOs. For the particular case of *recursive* functions we use $(B_\bot, \leq_\bot, \bot)$ (cf. Example 2), which is a PPO and, as an algebraic CPO, is its own completion. Overall, codomains being CPOs is not an issue.

The other condition is the continuity of $F : (A \to B) \to (A \to B)$. Let us examine what it entails. According to Definition 12, F must be monotonic - this is usually trivial to prove - and for all directed sets $S \subseteq (A \to B)$, $F\ (lub\ S) = lub\ \{F\ g \mid g \in S\}$.

Checking the latter condition is typically difficult, because, due to the higher-order nature of F, the *lub*s are taken in the nontrivial CPO of functions $A \to B$. We now

[6] As usual in functional programming $\to$ is right-associative, so $F : (A \to B) \to (A \to B)$ is often simply written as $F : (A \to B) \to A \to B$.

state a condition where *lub*s are taken in the simpler CPO B and show its equivalence to continuity, for the functionals $F : (A \to B) \to (A \to B)$ of interest.

Definition 26. *If A is a set and B is a CPO, we say that $F : (A \to B) \to (A \to B)$ is* Haddock-continuous *(or H-*continuous*) if F is monotonic and, for all directed sets $S \subseteq (A \to B)$, $F(\lambda\,(a : A) \to lub\,\{g\ a \mid g \in S\}) = \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\}$.*

Remark. Note that the *lub*s in Def. 26 exist: $lub\,\{g\ a \mid g \in S\}$ because, by Lemma 1, if S is directed the so are the $\{g\ a \mid g \in S\} \subseteq B$, for all $a \in A$; and $lub\{F\ g\ a \mid g \in S\}$ because, due to the monotonicity of F and the directedness of S, $\{F\ g\ a \mid g \in S\} \subseteq B$ is directed as well. The "Haddock" name for this version of continuity is tongue-in-cheek: it is ad hoc, i.e., for functionals only, unlike the standard notion of continuity that applies to all functions. Also unlike continuity, it is actually checkable in practice because in involves *lub*s in the relatively simple CPO B instead of the complex $A \to B$.

Before we prove the equivalence of H-continuity and continuity we prove a helper lemma, which also serves hereafter in function definitions.

Lemma 9. *If A is a set, B is a CPO, $F : (A \to B) \to (A \to B)$ is monotonic, and $S \subseteq A \to B$ is directed then for all $a \in A$, $(lub\,\{F\ g \mid g \in S\})\ a = lub\,\{F\ g\ a \mid g \in S\}$.*

Proof. This is an equality in the CPO $(B, \preceq, \bot)$, and it involves comparing functions in $A \to B$ according to the pointwise order $\sqsubseteq$. We have to prove:

($\preceq$): let $h : A \to B := \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\}$. Then, for all $g \in S$, $F\ g = \lambda\,(a : A) \to F\ g\ a \sqsubseteq \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\} = h$. It follows that h is an upper bound for the set $\{F\ g \mid g \in S\}$. Hence, for the least upper bound of that set, $lub\,\{F\ g \mid g \in S\} \sqsubseteq h = \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\}$. On any given $a \in A$ the latter inequality becomes $(lub\,\{F\ g \mid g \in S\})\ a \preceq lub\,\{F\ g\ a \mid g \in S\}$, which proves ($\preceq$).

($\succeq$): for all $g \in S$, $F\ g \sqsubseteq lub\,\{F\ g \mid g \in S\}$, and by definition of $\sqsubseteq$, for all $a \in A$, $F\ g\ a \preceq (lub\,\{F\ g \mid g \in S\})\ a$. Fix an abitrary $a \in A$. The last inequality shows that $(lub\,\{F\ g \mid g \in S\})\ a$ is an upper bound for $\{F\ g\ a \mid g \in S\}$. Hence, for the least upper bound of that set, $lub\,\{F\ g\ a \mid g \in S\} \preceq (lub\,\{F\ g \mid g \in S\})\ a$, which proves ($\succeq$). □

Theorem 3. *If A is a set and B is a CPO, then any functional $F : (A \to B) \to (A \to B)$ is continuous if and only if it is H-continuous.*

Proof. By Def. 12, F is continuous if and only if F is monotonic and for all directed $S \subseteq A \to B$, $F\ (lub\ S) = lub\,\{F\ g \mid g \in S\}$. By Lemma 1, $lub\ S = \lambda\,(a : A) \to lub\,\{g\ a \mid g \in S\}$ and, by using Lemma 9, $lub\,\{F\ g \mid g \in S\} = \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\}$. Hence, F is continuous iff *F is monotonic and for all directed $S \subseteq A \to B$, $F\ (\lambda\,(a : A) \to lub\,\{g\ a \mid g \in S\}) = \lambda\,(a : A) \to lub\,\{F\ g\ a \mid g \in S\}$.* But the statement in italics is by Def. 26 the H-continuity of F: the theorem is proved. □

A consequence of Corollary 2 and Th. 3 is the so-called **Haddock's fixpoint theorem**:

Corollary 3. *If A is a set and B is a CPO, and $F : (A \to B) \to A \to B$ is Haddock-continuous, then F has the least fixpoint $\mu F = lub\,\{F^n \mid n \in \mathbb{N}\}$, where $F^0 = \lambda\ \_ \to \bot$.*

Remark. The function being defined here is μF. Using Lemma 9 we obtain a "pointwise" version of the description of the fixpoint: for all $a \in A$, $\mu F\ a = lub\,\{F^n\ a \mid n \in \mathbb{N}\}$, which is sometimes more convenient when reasoning about the defined function μF.

5. Examples

We use Haddock's theorem to define several functions: two corecursive ones and two recursive ones, some of which are partial. We use coinduction to prove some properties of the defined functions; in particular, their totality, where it applies.

5.1. Two Corecursive Functions

We define, and prove properties of, a filter function on streams and a mirror function on Rose trees. The first one is partial, the second one is total.

5.1.1. Filter

Consider the set *stream A* from Example 19. We define a function $filter_p$ on *stream A*, parameterized by a Boolean function $p : A \to bool$, which takes a stream as input and computes the sub-stream of its input containing the values for which p evaluates to *true*. With the constructor *cons* defined as in Example 19, we define the accessors by $head\ (cons\ a\ s) = a$, $tail\ (cons\ a\ s) = s$, to be only used on non-$\bot$ streams.

Definition 27. *The functional* $Filter_p : (stream\ A \to stream\ A) \to (stream\ A \to stream\ A)$ *is given by*

$$Filter_p\ (f : stream\ A \to stream\ A)(s : stream\ A) :=$$
$$\quad if\ s = \bot\ then\ \bot\ else\ if\ p\ (head\ s) = true\ then\ cons\ (head\ s)\ (f\ (tail\ s)).\ else\ f\ (tail\ s)$$

We want to apply Haddock's theorem and define $filter_p$ as the least fixpoint of $Filter_p$.

Lemma 10. *$Filter_p$ is H-continuous.*

Proof. By Def. 26 we have to prove that $Filter_p$ is monotonic, which is easy, and, for al directed $S \subseteq stream\ A \to stream\ A$,

$$Filter_p\ (\lambda\, s \to lub\,\{g\ s \mid g \in S\}) = \lambda\, s \to lub\,\{Filter_p\ g\ s \mid g \in S\}$$

Fix an arbitrary S as above. Hence we have to prove that for an arbitrary $s \in stream\ A$,

$$(*)\quad Filter_p(\lambda\, s \to lub\,\{g\ s \mid g \in S\})\ s = lub\,\{Filter_p\ g\ s \mid g \in S\}$$

which reduces to a case analysis on s, taking into account the definition of $Filter_p$:

- if $s = \bot$ then $(*)$ amounts to $\bot = lub\ \{\bot\}$, which is trivial;
- if $s \neq \bot$, $p\ (head\ s) = true$: the *lhs* of $(*)$ is $cons\ (head\ s)\ (lub\,\{g\ s \mid g \in S\})$. Since $cons\ (head\ s) : stream\ A \to stream\ A$ is continuous (as the completion of the monotonic $cons^\circ\ (head\ s) : stream^\circ\ A \to stream^\circ\ A$, cf. Example 19), the *lhs* of $(*)$ is equal to $lub\,\{cons\ (head\ s)\ (g\ s) \mid g \in S\}$; which is precisely the *rhs* of $(*)$;
- if $s \neq \bot$, $p\ (head\ s) = false$: both *lhs* and *rhs* are $lub\,\{cons\ (head\ s)\ (g\ s) \mid g \in S\}$. Hence, $(*)$ holds in this case as well, and the H-continuity of $Filter_p$ is proved. □

Using Haddock's theorem (Corollary 3) we obtain the function $filter_p : stream\ A \to stream\ A$ as the least fixpoint of $Filter_p$, and, moreover, $filter_p = lub\,\{Filter_p^n \mid n \in \mathbb{N}\}$. By the remark following Corollary 3: $\forall s \in stream\ A, filter_p\ s = lub\,\{Filter_p^n\ s \mid n \in \mathbb{N}\}$.

Remark. In the above example the H-continuity of the second-order functional $Filter_p : (stream\ A \to stream\ A) \to (stream\ A \to stream\ A)$ has been reduced to the continuity of the first-order $cons\ (head\ s) : stream\ A \to stream\ A$, which was easily established.

For the function $filter_p$, parameterized by $p : A \to bool$, we prove that it is, in general, partial (except in the case where $p\ a = true$ for all $a \in A$, in which case $filter_p$ is total). We also prove that the restriction of $filter_p$ to a certain subset of $stream\ A$ is total.

Partiality. Assume some $a \in A$ with $p\ a = false$, and consider the stream a^∞ which, informally, is an infinite repetition of a. We prove that a^∞ is total but $filter_p\ a^\infty = \bot$, which implies that $filter_p$ is not total under the given assumptions:

- first, a^∞ is formally defined as the least fixpoint of $\lambda\, s \to cons\ a\ s$, which, as noted above, is continuous; by Kleene's theorem, $a^\infty = cons\ a\ a^\infty$, hence, $tail\ a^\infty = a^\infty$;
- second, we prove that a^∞ is total, i.e. $a^\infty \in \nu Total$ where $Total$ is the instance on $stream\ A$ of the homonymous general, monotonic function from Section 2.4; here, for all $S \subseteq stream\ A$, $Total\ S = \{s \in stream\ A \mid \exists a'\ s',\ s = cons\ a'\ s' \wedge s' \in S\}$. By the coinduction principle for the instance of $Total$ on streams, in order to prove $a^\infty \in \nu Total$, i.e., $\{a^\infty\} \subseteq \nu Total$, it is enough that $\{a^\infty\} \subseteq Total\,\{a^\infty\}$, which holds because there do exists $a' := a$ and $s' := a^\infty$ such that $a^\infty = cons\ a'\ s'$, and $s' = a^\infty \in \{a^\infty\}$;
- third, we prove by induction on n that for all $n \in \mathbb{N}$, $Filter_p^n\ a^\infty = \bot$;
- last, using $\forall s \in stream\ A, filter_p\ s = lub\,\{Filter_p^n\ s \mid n \in \mathbb{N}\}$ we obtain $filter_p\ a^\infty = \bot$.

Totality of a Restriction. We now prove that the restriction of $filter_p$ to the set $\Box\Diamond_p$ of streams on which, informally speaking, p is $true$ on infinitely many positions, is total.

Formally, let $\Diamond_p$ the subset of $stream\ A$ be inductively defined by the rules (now): $(cons\ a\ s) \in \Diamond_p$ if $p\ a = true$ and (later): $(cons\ a\ s) \in \Diamond_p$ if $p\ a = false$ and $s \in \Diamond_p$. That is, $\Diamond_p$ is the set of streams that have at least one position on which p is $true$.

Next, we define the set of streams $\Box_q$ such that $q : A \to bool$ is $true$ on all positions. Formally, let $F^{\Box_q} : 2^{stream\ A} \to 2^{stream\ A}$ be defined by: for all $S \subseteq stream\ A$, $F^{\Box_q}\ S = \{s \in stream\ A \mid \exists a'\ s',\ s = cons\ a'\ s' \wedge q\ a' = true \wedge s' \in S\}$. We prove that $F^{\Box_q}$ is monotonic with respect to $\subseteq$. By the Knaster-Tarski theorem, $F^{\Box_q}$ has a greatest fixpoint, denoted by $\Box_q$, which satisfies the following:

- unfolding equation: $\Box_q = \{s \in stream\ A \mid \exists a'\ s',\ s = cons\ a'\ s',\ q\ a' = true, s' \in \Box_q\}$;
- coinduction principle: for all $S \subseteq stream\ A$, $S \subseteq F^{\Box_q}\ S$ implies $S \subseteq \Box_q$. By expanding this definition one gets: *to prove* $s \in \Box_q$, *find* $S \subseteq stream\ A$ *with* $s \in S$ *and for all* $x \in S$, $x = cons\ a'\ s'$ *for some* $a' \in A$ *with* $q\ a' = true$ *and* $s' \in S$.

Let us now defne $\Box\Diamond_p := \Box_{(\lambda\, s \to s \in \Diamond_p)}$. Informally, $\Box\Diamond_p$ is the set of streams on which p is $true$ on infinitely many positions. We prove that $filter_p$ restricted to $\Box\Diamond_p$ is total:

Proposition 8. *For all* $s \in \Box\Diamond_p$ *it holds that* $filter_p\ s \in \nu Total$.

Proof. If $\Box\Diamond_p = \emptyset$ the proposition holds vacuously. Otherwise, fix $s \in \Box\Diamond_p$. The coinduction principle for totality of streams is equivalently reformulated as: to prove $\mathit{filter}_p\ s \in \nu\mathit{Total}$, find $S \subseteq \mathit{stream}\ A$ s.t. $\mathit{filter}_p\ s \in S$ and for all $x \in S$, $\mathit{tail}\ x \in S$. We choose $S := \mathit{filter}_p(\Box\Diamond_p) = \{s' \in \mathit{stream}\ A \mid \exists y \in \Box\Diamond_p,\ s' = \mathit{filter}_p\ y\}$:

- $\mathit{filter}_p\ s \in S$: this is trivial since $s \in \Box\Diamond_p$;
- for all $x \in S$, $\mathit{tail}\ x \in S$; that is, if $x = \mathit{filter}_p\ u$ for some $u \in \Box\Diamond_p$, then $\mathit{tail}\ x = \mathit{filter}_p\ v$ for some $v \in \Box\Diamond_p$. Choose v to be suffix of u starting exactly *after* the first position on u where p is *true*. The fact that u is well-defined, that it belongs to $\Box\Diamond_p$, and that $\mathit{tail}\ x = \mathit{filter}_p\ u$, are established using the following ingredients: the induction principle for $\Diamond_p$; the coinduction principle for $\Box\Diamond_p$; the fixpoint equation of filter_p; and the fact that $\bot$ and cons_a behave like constructors for $\mathit{stream}\ A$. □

5.1.2. Mirror

Consider the set T of Rose trees from Example 20. We define a function *mirror* on T, which is an example of nested corecursive function. With the constructor *tree* defined as in Example 20, we define an accessor *forest* by $\mathit{forest}\ (\mathit{tree}\ l) = l$, to be only used on non-$\bot$ trees. We shall also be using the usual functions *rev* and *map* on lists.

Definition 28. *The functional* $\mathit{Mirror} : (T \to T) \to (T \to T)$ *is defined by*
$\mathit{Mirror}\ (f : T \to T \to T)\ (t : T) := \mathit{if}\ t = \bot\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{tree}\ (\mathit{rev}\ (\mathit{map}\ f\ (\mathit{forest}\ t)))$.

Before we prove that *Mirror* is H-continuous we need two other continuity results.

Lemma 11. *The function* $\mathit{rev} : \mathit{list}\ T \to \mathit{list}\ T$ *is continuous.*

Proof. In Example 15 it is noted that $\mathit{list}\ T$ is an algebraic DCPO isomorphic to $\Sigma_{n\in\mathbb{N}} A^{\{<n\}}$ Hence $\mathit{rev} : \mathit{list}\ T \to \mathit{list}\ T$ can identified with $\mathit{rev}' : \Sigma_{n\in\mathbb{N}} T^{\{<n\}} \to \Sigma_{n\in\mathbb{N}} T^{\{<n\}}$ defined as follows: for any $f \in \Sigma_{n\in\mathbb{N}} T^{\{<n\}}$, consider the unique $n \in \mathbb{N}$ such that $f \in T^{\{<n\}}$, and let $\mathit{rev}'\ f = \lambda(i : \{< n\}) \to f\ (n - 1 - i)$. The continuity of *rev* is equivalent to that of rev'; we prove the continuity of rev' because is is easier.

Proving that rev' is monotonic is trivial. Next, consider a directed set $S \subseteq \Sigma_{n\in\mathbb{N}} T^{\{<n\}}$. We only have to prove $(*)$: $\mathit{rev}'\ (\mathit{lub}\ S) = \mathit{lub}\{\mathit{rev}'\ f \mid f \in S\}$. There is a unique $n \in \mathbb{N}$ such that $S \subseteq T^{\{<n\}}$, and, by using Lemma 1, $\mathit{lub}\ S = \lambda(i : \{< n\}) \to \mathit{lub}\{f\ i \mid f \in S\}$. Then by definition of rev', for the *lhs* of $(*)$ we have $\mathit{rev}'\ (\mathit{lub}\ S) = \lambda(i : \{< n\}) \to \mathit{lub}\{f\ (n - 1 - i) \mid f \in S\}$. For the *rhs* of $(*)$, $\mathit{lub}\{\mathit{rev}'\ f \mid f \in S\} = \mathit{lub}\{\lambda(i : \{< n\}) \to f\ (n - 1 - i) \mid f \in S\}$, i.e., the *lub* of a directed set of functions; using Lemma 1 again, the last expression becomes $\lambda(i : \{< n\}) \to \mathit{lub}\{f\ (n - 1 - i) \mid f \in S\}$. Both the *lhs* and *rhs* of $(*)$ are equal to $\lambda(i : \{< n\}) \to \mathit{lub}\{f\ (n - 1 - i) \mid f \in S\}$; which proves $(*)$. □

Lemma 1 was helpful twice in the proof. It is also helpful in the proof of the next result.

Lemma 12. *For all* $l \in \mathit{list}\ T$, *the function* $\lambda(g : T \to T) \to \mathit{map}\ g\ l$ *is continuous.*

Proof. Consider a directed set $S \subseteq T \to T$ and let $\mathit{map}'_l := \lambda g \to \mathit{map}\ g\ l$. To prove that $\mathit{map}'_l : (T \to T) \to \mathit{list}\ T$ is continuous we prove that it is monotonic, which is trivial, and $(*)$: $\mathit{map}'_l(\mathit{lub}\ S) = \mathit{lub}\{\mathit{map}'_l\ g \mid g \in S\}$. By using Lemma 1 in the *lhs* of $(*)$, $\mathit{map}'_l(\mathit{lub}\ S) = \mathit{map}'_l(\lambda(t : T) \to \mathit{lub}\{g\ t \mid g \in S\}) = \mathit{map}\ (\mathit{lub}\{g\ t \mid g \in S\})\ l$;

and, by expanding the definition of map'_l in the *rhs* of $(*)$: $lub\,\{map'_l\ g \mid g \in S\} = lub\,\{map\ g\ l \mid g \in S\}$. Hence in order to prove $(*)$ we only have to prove the equality

$(**)$: $map\ (lub\,\{g\ t \mid g \in S\})\ l = lub\,\{map\ g\ l \mid g \in S\}$

in the pointwise-ordered poset $(list\ T, \sqsubseteq)$. Proving it will also involve the poset $(T, \preceq)$.

The first thing to prove is that both sides of $(**)$ are lists of the same length. Due to properties of *map*, the length of the *lhs* of $(**)$ is that of l, i.e., $len\ l$. Regarding the *rhs*, we have $map\ g\ l \sqsubseteq lub\,\{map\ g\ l \mid g \in S\}$ for all $g \in S$, which implies $len\ (lub\,\{map\ g\ l \mid g \in S\}) = len\ (map\ g\ l) = len\ l$. This being settled, we next prove:

$(\sqsubseteq)$: we denote by l_i the i-th element of $l \in list\ T$ and prove that for all $i < len\ l$: $(map\ (lub\,\{g\ t \mid g \in S\})\ l)_i \preceq (lub\,\{map\ g\ l \mid g \in S)_i$, which, thanks to properties of *map*, becomes $lub\,\{g\ l_i \mid g \in S\} \preceq (lub\,\{map\ g\ l \mid g \in S)_i$. Now, by properties of *lub*, for all $g \in S$, $map\ g\ l \sqsubseteq lub\,\{map\ g\ l \mid g \in S\}$, which implies, using again properties of *map* that for all $i < len\ l$, $g\ l_i = (map\ g\ l)_i \preceq (lub\,\{map\ g\ l \mid g \in S)_i$. Again by properties of *lub*: $lub\,\{g\ l_i \mid g \in S\} \preceq (lub\,\{map\ g\ l \mid g \in S)_i$, which is what we had to prove for $(\sqsubseteq)$.

$(\sqsupseteq)$: what we have to prove is $lub\,\{map\ g\ l \mid g \in S\} \sqsubseteq map\ (lub\,\{g\ t \mid g \in S\})\ l$. Due to the monotoncity of $map'_l = \lambda\, g \to map\ g\ l$ and by properties of *lub*, for all $g \in S$, $map\ g\ l \sqsubseteq map\ (lub\,\{g\ t \mid g \in S\})\ l$. Again by properties of *lub*, we obtain the desired $lub\,\{map\ g\ l \mid g \in S\} \sqsubseteq map\ (lub\,\{g\ t \mid g \in S\})\ l$; which proves $(\sqsupseteq)$ and the lemma. □

We now prove the lemma that enables the application of Haddock's fixpoint theorem:

Lemma 13. *The functional Mirror is H-continuous.*

Proof. By Def. 26 we have to prove that *Mirror* is monotonic, which is trivial, and that for all directed sets $S \subseteq T \to T$ and all $t \in T$:

$Mirror(\lambda\,(x : T) \to lub\,\{g\ x \mid g \in S\})\ t = lub\,\{Mirror\ g\ t \mid g \in S\}$

If $t = \bot$, by Def. 28 of *Mirror*, the above equality becomes $\bot = lub\{\bot\}$, which is trivial. Otherwise, $t = tree\ l$, for some $l \in list\ T$, and, the above equality becomes:

$(*)$: $tree\ (rev\ (map\ (\lambda\,(x : T) \to lub\,\{g\ x \mid g \in S\})\ l)) = lub\,\{tree\ (rev\ (map\ g\ l)) \mid g \in S\}$.

Since *tree* is continuous (as natural extension of the monotonic $tree^\circ$, cf. Example 20),

$lub\,\{tree\ (rev\ (map\ g\ l)) \mid g \in S\} = tree\ (lub\,\{rev\ (map\ g\ l) \mid g \in S\})$.

Hence, in order to prove our target equality $(*)$ it is enough to prove the simpler equality

$rev\ (map\ (\lambda\,(x : T) \to lub\,\{g\ x \mid g \in S\})\ l) = lub\,\{rev\ (map\ g\ l) \mid g \in S\}$.

By Lemma 11 *rev* is continuous; hence, what we have to prove further simplifies to

$map\ (\lambda\,(x : T) \to lub\,\{g\ x \mid g \in S\})\ l = lub\,\{map\ g\ l \mid g \in S\}$

Using Lemma 1, what we have to prove becomes $map\ (lub\,S)\ l = lub\,\{map\ g\ l \mid g \in S\}$, which is implied by the continuity of $\lambda\,(g : T \to T) \to map\ g\ l$, i.e., by Lemma 12. □

Using Haddock's theorem with *Mirror* we define $mirror := \mu Mirror$ and know that $mirror = lub\,\{Mirror^n \mid n \in \mathbb{N}\}$. By the remark following Corollary 3, the latter equality also holds "pointwise": for all $t \in T$, $mirror\ t = lub\,\{Mirror^n\ t \mid n \in \mathbb{N}\}$.

Remark. In the above example the H-continuity of the functional $Mirror : (T \to T) \to (T \to T)$ has been reduced to the continuity of *tree*, which holds by construction since it is a natural completion; and of $rev : list\ T \to list\ T$ and of $\lambda\, g \to map\ g\ l : (T \to T) \to list\ T$ parameterized by $l \in list\ T$. Establishing those continuities took several applications of Lemma 1 to reduce the complexity of CPOs where *lub*s are taken.

Totality. We now prove that *mirror* is a total function. The functional for totality $Total : 2^T \to 2^T$ in this case is defined, for all $S \subseteq T$, by $Total\ S = \{t \in T \mid \exists\, l \in list\ T,\ t = tree\ l \wedge \forall i < len\ l,\ t_i \in S\}$. It is monotonic, and its greatest fixpoint *vTotal* satisfies

- unfolding equation: $vTotal = \{t \in T \mid \exists\, l \in list\ T,\ t = tree\ l \wedge \forall i < len\ l,\ t_i \in vTotal\}$;
- coinduction principle: to prove $t \in vTotal$, find $S \subseteq T$ with $t \in S$ and $S \subseteq Total\ S$: for all $x \in S$, there exists $l \in list\ T$ such that $x = tree\ l$ and for all $i < len\ l$, $x_i \in S$.

Proposition 9. *for all $t \in T$, $t \in vTotal$ implies $mirror\ t \in vTotal$.*

Proof. Apply the above coinduction principle with $S = \{mirror\ y \mid y \in vTotal\}$. Since by hypothesis $t \in vTotal$, clearly, $mirror\ t \in S$. Assuming that some $mirror\ y \in S$ equals $\bot$: from $mirror\ y = lub\,\{Mirror^n\ y \mid n \in \mathbb{N}\}$ we obtain that for all $n \in \mathbb{N}$, $Mirror^n\ y = \bot$, from which we derive that $y = \bot$, in contradiction with $y \in vTotal$.

Hence, there exists $l' \in list\ T$ such that $mirror\ y = tree\ l'$, and in order to conclude the proof we need to establish that for all $i < len\ l'$, $l'_i \in S$. Now, $mirror\ y \in S$ implies $y \in vTotal$, thus, there exists $l \in list\ T$ such that $y = tree\ l$, and using the unfolding equation above, for all $i < len\ l, l_i \in vTotal$, which we equivalently rephrase as "for all $i < len\ l, l_{(len\ l)-i-1} \in vTotal$". Then, $mirror\ (tree\ l) = tree\ l'$, and using the fixpoint equation for *mirror*, we obtain $rev\ (map\ mirror\ l) = l'$, which, using properties of *map* and *rev*, implies $len\ l = len\ l'$ and for all $i < len\ l$, $l'_i = mirror(l_{(len\ l)-1-i})$. From $l_{(len\ l)-1-i} \in vTotal$ we obtain $l'_i \in S$; which is all that remained to be proved. □

Involutivity. The second property that we prove on *mirror* is an equation. We use the equivalence between equality and bisimulation - the instance of Th. 2 for Rose trees.

Adapting bisimulation (Def. 25) for Rose trees entails the following:

- unfolding equation : $\approx\ =\ \{(t, t') \in T \times T \mid t = t' = \bot \vee \exists\, l, l' \in list\ T,\ len\ l = len\ l' \wedge t = tree\ l \wedge t' = tree\ l' \wedge \forall\, i < len\ l,\ l_i \approx l'_i\}$;
- coinduction principle: to prove $t \approx t'$, find $R \subseteq T \times T$ satisfying $(t, t') \in R$ and for all $(x, x') \in R$, either $x = x' = \bot$ or there are $l, l' \in list\ T$ such that $len\ l = len\ l'$, $x = tree\ l$, $x' = tree\ l'$ and for all $i < len\ l$, $(l_i, l'_i) \in R$.

Proposition 10. *mirror is involutive, i.e. for all $t \in T$, $mirror\ (mirror\ t) = t$.*

Proof. Fix $t \in T$. Thanks to the equivalence between equality and bisimulation we shall prove $mirror\ (mirror\ t) \approx t$. In the coinduction principle above, we choose

$$R = \{(mirror\ (mirror\ y),\ y) \mid y \in T\}.$$

Obviously, $(mirror\ (mirror\ t),\ t) \in R$. Moreover, for all $mirror\ (mirror\ x),\ x) \in R$: either $x = \bot$, in which case, using the fixpoint equation of *mirror*, $mirror\ (mirror\ x) =$

$\perp$. Or $x = \mathit{tree}\ l$, for some $l \in \mathit{list}\ T$. In this case, using the fixpoint equation of *mirror* and properties of *map* and *rev*, $\mathit{mirror}\ (\mathit{mirror}\ x) = \mathit{tree}\ (\mathit{map}\ (\mathit{mirror} \circ \mathit{mirror})\ l)$, where $\circ$ denotes function composition. Again, by properties of *len* and *map*, $\mathit{len}\ (\mathit{map}\ (\mathit{mirror} \circ \mathit{mirror})\ l) = \mathit{len}\ l$; and we only have to prove that for all $i < \mathit{len}\ l$, $((\mathit{map}\ (\mathit{mirror} \circ \mathit{mirror})\ l)_i,\ l_i) \in R$. But $(\mathit{map}\ (\mathit{mirror} \circ \mathit{mirror})\ l)_i = \mathit{mirror}\ (\mathit{mirror}\ l_i)$, and by definition of R, $(\mathit{mirror}\ (\mathit{mirror}\ l_i),\ l_i) \in R$; which proves the result. □

5.2. Two Partial Recursive Functions

Partial recursive functions of codomain B are encoded as total functions to $B_\perp = B \cup \{\perp\}$ where $\perp \notin B$ encodes undefinedness, usually due to nontermination. The flat order $(B_\perp, \preceq, \perp)$ is a PPO and is its own natural completion when seen as an algebraic CPO. Hence, partial recursive functions are just particular cases of corecursive functions.

Below we define two functions. The first one computes the number of steps taken by the Collatz sequence, starting from an input $n \geq 1$, to reach 1. It its not known whether this happens for all inputs - the answer is a conjecture in number theory - so, our function is partial. The second example is a function modeling *while* loops in a monadic imperative language shallowly embedded in a functional language. Like the loops themselves our function may not terminate, hence, it is partial as well.

5.2.1. Collatz

We start by defining a successor function for $\mathbb{N}_\perp$ and proving its continuity.

Definition 29. *The successor function* $\mathit{Succ} : \mathbb{N}_\perp \to \mathbb{N}_\perp$ *is defined by* $\mathit{Succ}\ \perp = \perp$ *and* $\mathit{Succ}\ n = 1 + n$ *if* $n \in \mathbb{N}$.

Lemma 14. *The function Succ is continuous.*

Proof. Monotonicity is trivial. Next, we consider a directed set $S \subseteq \mathbb{N}_\perp$ and show $\mathit{Succ}(\mathit{lub}\ S) = \mathit{lub}\ \{\mathit{Succ}\ x \mid x \in S\}$. Now, directed sets in the flat order are either singletons $\{x\}$ with $x \in \mathbb{N}_\perp$ or of the form $\{\perp, n\}$ for some $n \in \mathbb{N}$. If $S = \{x\}$ then what we have to prove amounts to the trivial $\perp = \mathit{lub}\ \{\perp\}$. If $S = \{\perp, n\}$ then it amounts to $\mathit{Succ}\ n = \mathit{lub}\{\mathit{Succ}\ \perp, \mathit{Succ}\ n\}$, which is also trivial. □

Definition 30. *The functional* $\mathit{Collatz} : (\mathbb{N}_\perp \to \mathbb{N}_\perp) \to \mathbb{N}_\perp \to \mathbb{N}_\perp$ *is defined by*

$\mathit{Collatz}\ f\ x =$
  $\mathit{if}\ x = \perp\ \mathit{then}\ \perp$
  $\mathit{else\ if}\ x\ \mathit{mod}\ 2 = 0\ \mathit{then}\ \mathit{Succ}\ (f\ (x \div 2))$
  $\mathit{else\ if}\ x = 1\ \mathit{then}\ 0$
  $\mathit{else}\ \mathit{Succ}\ (f\ (3 * x + 1))$.

Proposition 11. *Collatz is H-continuous*

Proof. We need to prove that for all directed $S \subseteq \mathbb{N}_\perp \to \mathbb{N}_\perp$ and all $x \in \mathbb{N}_\perp$,

$(*)$: $\mathit{Collatz}\ (\lambda\, (y : \mathbb{N}_\perp) \to \mathit{lub}\ \{g\ y \mid g \in S\})\ x = \mathit{lub}\ \{\mathit{Collatz}\ g\ x \mid g \in S\}$

- if $x = \perp$, $(*)$ reduces to $\perp = \mathit{lub}\ \{\perp\}$;

- if $x \neq \bot$, $x \; mod \; 2 = 0$: the *lhs* of $(*)$ becomes $Succ \; (lub \, \{g \; (x \; \div \; 2) \mid g \in S\})$ and the *lhs* of $(*)$ is $lub \, \{Succ \; (g \; (x \; \div \; 2)) \mid g \in S\}$. The two are equal by Lemma 14;
- if $x = 1$: $(*)$ reduces to $0 = lub \, \{0\}$;
- if $x \neq \bot$, $x \; mod \; 2 \neq 0$, $x \neq 1$: the *lhs* of $(*)$ becomes $Succ(lub \, \{g \; (3 * x + 1) \mid g \in S\})$ and the *lhs* of $(*)$ is $lub \, \{Succ(g \; (3 * x + 1)) \mid g \in S\}$. The two are equal by Lemma 14. □

Using Haddock's theorem with *Collatz* we define $collatz := \mu Collatz$ and know that $collatz = lub \, \{Collatz^n \mid n \in \mathbb{N}\}$. By the remark following Corollary 3, the latter equality also holds "pointwise": for all $x \in \mathbb{N}_\bot$, $collatz \; x = lub \, \{Collatz^n \; x \mid n \in \mathbb{N}\}$.

Using the latter equality one can prove, for example, that $collatz \; 0 = \bot$.

5.2.2. While

Our last example is a partial recursive function modeling *while* loops in a monadic language. We define it and prove a dedicated Hoare-logic rule for it, which together with rules for the other language constructs enables proofs of program in the language[7].

A Termination State Monad. This monad enables shallow embeddings of imperative languages with possibly nonterminating programs in total functional languages. Its ingredients are listed below. Among them, program instructions in the guest imperative language are writen in **boldface**. They should not be confused with the sometimes homonymous statements of the host functional language; those are written in *italic*.

For compatibility with the rest of the paper we choose a set-theoretical presentation.

Definition 31. *A* termination state monad *consists of the following ingredients:*

- *a set* S *of* states*, and a set* Ω *of sets of* outputs*;*
- *for every set* $O \in \Omega$*, the set of* programs *over states* $\in S$ *emitting outputs* $\in O$ *:* $prog \; S \; O := S \to (O \times S)_\bot$
- *basic program builders: returning a value, and sequencing:*
  - $\mathbf{ret}(o : O) : prog \; S \; O := \lambda \, s \to (o, s)$ *;*
  - $bind(p : prog \; S \; A)(f : A \to \; prog \; S \; B) : prog \; \; S \; B :=$
    $\lambda \, s \to if \; p \; s = \bot \; then \; \bot \; else \; let \; (a, s') := p \; s \; in \; f \; a \; s'$
- *notations, for imperative look-and-feel:*
  - **do** $x \leftarrow p$; q *stands for bind* $p \; (\lambda \, x \to q)$ *(i.e., the output x of p is passed to q);*
  - p ; q *stands for* **do** $\_ \leftarrow p$; q *(i.e., the output of p is not passed to q - it is ignored).*

Additional *primitive* instructions are defined once a concrete set of states S is chosen, e.g., a pair consisting of a tuple of registers and of an array modeling a memory; primitives typically read and write in components of the state; we are not interested in them here. What we are interested in is two composite instructions: conditional and loop.

[7] Verification examples of monadic programs with an earlier version of *while* loops is presented in [16].

Definition 32. *The conditional instruction* **if_then_else_** : $prog\ S\ bool \to prog\ S\ \{*\} \to prog\ S\ \{*\} \to prog\ S\ \{*\}$ *is defined by:* **if** c **then** p **else** $q := \mathbf{do}\ x \leftarrow c\,;(\mathit{if}\ x\ \mathit{then}\ p\ \mathit{else}\ q)$.

Defining **while** *Loops.* As usual, we define the functional of the function of interest, then prove that the functional is H-continuous, and finally apply Haddock's theorem.

Definition 33. *For all* $c \in prog\ S\ bool$*, we define* $While_c : (prog\ S\ \{*\} \to prog\ S\ \{*\}) \to (prog\ S\ \{*\} \to prog\ S\ \{*\})$ *by* $While_c\ f\ p :=$ **if** c **then** p ; $(f\ p)$ **else ret** $*$.

Proposition 12. *For all* $c \in prog\ S\ bool$*,* $While_c$ *is H-continuous.*

Proof. We have to prove that $While_c$ is monotonic, which is trivial, and for all directed $F \subseteq (prog\ S\ \{*\} \to prog\ S\ \{*\})$ and $p \in prog\ S\ \{*\}$,

$$While_c\ (\lambda\,(q : prog\ S\ \{*\}) \to lub\,\{f\ q \mid f \in F\})\ p = lub\,\{While_c\ f\ p \mid f \in S\}$$

By Def. 33 this is equivalent to the following equation, which we shall refer to as (♯):

if c **then** $(p\ ;\ lub\,\{f\ p \mid f \in F\})$ **else ret** $* = lub\,\{$**if** c **then** p ; $(f\ p)$ **else ret** $* \mid f \in S\}$

Fix $p \in prog\ S\ \{*\}$ and let $G := (\lambda\,(q : prog\ S\ \{*\}) \to$ **if** c **then** p ; q **else ret** $*$. Then,

- the *lhs* of (♯) is $G\ (lub\,\{f\ p \mid f \in F\})$;
- the *rhs* of (♯) is $lub\,\{G\ (f\ p) \mid f \in F\}$;
- hence, (♯) is equivalent to $G\ (lub\,\{f\ p \mid f \in F\}) = lub\,\{G\ (f\ p) \mid f \in F\}$; that is, (♯) is implied by the continuity of G; thus, all what is left to prove is the continuity of G.

Now, remembering that $prog\ S\ \{*\} = S \to (\{*\} \times S)$, we have $G : (S \to (\{*\} \times S) \to (S \to (\{*\} \times S)$, which is the right form for using Theorem 3 about equivalence of continuity and H-continuity ; hence, what we have to prove is the H-continuity of G.

For this, we prove that G is monotonic, which is trivial, and that for all directed $P \subseteq S \to (\{*\} \times S)$ and $s \in S$,

$$G\ (\lambda\,(t : S) \to lub\,\{q\ t \mid q \in P\})\ s = lub\,\{G\ q\ s \mid q \in P\}$$

which by expanding the definition of G, amounts to the equality hereafter called (♭):

(**if** c **then** p ; $(\lambda\,(t : S) \to lub\,\{q\ t \mid q \in P\})$ **else ret** $*$) s =
$lub\,\{$(**if** c **then** p ; q **else ret** $*$) $s \mid q \in P\}$

Using Definition 32 of the **if_then_else_** instruction we distinguish the following cases:

- either $c\ s = \bot$, in which case (♭) amounts to $\bot = lub\,\{\bot\}$, which is trivial;
- or $c\ s = (b, s')$ for some $b \in \{true, false\}$ and $s' \in S$. By case analysis on b:
  - if $b = false$ then (♭) amounts to **ret** $* = lub\,\{$**ret** $*\}$, which is trivial;
  - if $b = true$ then (♭) amounts to $(p\ ;\ (\lambda\,(t : S) \to lub\,\{q\ t \mid q \in P\}))\ s' = lub\{(p\ ;\ q)\ s' \mid p \in P\}$, known below as (†). Using the notation ";" as *bind* and the definition of *bind* from Def. 31 of the termination state monad, two cases appear:
    - if $p\ s' = \bot$ then (†) amounts to $\bot = lub\,\{\bot\}$, which is trivial;

- $*$ if $p\ s' = (*, s'')$ for some $s'' \in S$, then (†) amounts to $lub\,\{q\ s'' \mid q \in P\} = lub\,\{q\ s'' \mid q \in P\}$, which is trivial; which proves this case and the proposition. □

Using Haddock's theorem with $While_c$ we define **while** $c := \mu While_c$ and know that **while** $c = lub\,\{While_c^n \mid n \in \mathbb{N}\}$. By the remark following Corollary 3, the latter equality also holds "program-wise": for all $p \in prog\ S\,\{*\}$, **while** $c\ p = lub\,\{While_c^n\ p \mid n \in \mathbb{N}\}$. And the latter holds "state-wise": for all $s \in S$, **while** $c\ p\ s = lub\,\{While_c^n\ p\ s \mid n \in \mathbb{N}\}$.

A Hoare-logic rule for **while** *loops.* We briefly recap some elements of Hoare logic and adapt them to the particular setting of programs in a a termination state monad.

Definition 34. *In the context of a termination state monad with states S and set of output sets Ω (cf. Definition 31), a* Hoare triple *is an expression of the form $\{P\}\,q\,\{R\}$ for some $A \in \Omega$, $P \subseteq S$, $q \in prog\ S\ A$, and $R \subseteq (A \times S)$. In a triple $\{P\}\,q\,\{R\}$, P is called the* precondition *and Q is called the* postcondition*. A Hoare triple $\{P\}\,q\,\{\ R\}$ is* valid *if for all $s, s' \in S$ and $a \in A$, $P\ s$ and $q\ s = (a, s')$ imply $R\ a\ s'$. A Hoare* rule *is an entailment of the form $H_1, \ldots H_n \vdash H$ such that for all $1 \leq i \leq n$, H_i is a Hoare triple; when $n = 0$, the Hoare rule $\vdash H$ is identified with the Hoare triple H. The Hoare rule $H_1, \ldots H_n \vdash H$ is* valid *if the validity of the triples $H_1, \ldots, H_n$ implies the validity of H.*

For example, the Hoare rule for the conditional statement is defined as follows:

Definition 35. *In the context of Definition 34, let $P \subseteq S$, $Q \subseteq (bool \times S)$, $R \subseteq (A \times S)$, $c \in prog\ S\ bool$ and $p, q \in prog\ S\ A$. Then, the Hoare rule for* **if_then_else_** *is:*

$$\{P\}\,c\,\{Q\},\ \{Q\ true\}\,p\,\{R\},\ \{Q\ false\}\,q\,\{R\} \vdash \{P\}\,\textbf{if}\ c\ \textbf{then}\ p\ \textbf{else}\ q\,\{R\}.$$

Remark. We freely identify sets with their characteristic predicates and use λ notation for predicates. Above, when $Q \subseteq (bool \times S)$, $Q\ true$ is the projection of Q on $true$; as predicates, $Q\ true = \lambda\,(s : S) \to Q\ true\ s$. These conventions are hereafter assumed.

Lemma 15. *The Hoare rule for* **if_then_else_** *is valid.*

Proof. Assume that the hypotheses $\{P\}\,c\,\{Q\},\ \{Q\ true\}\,p\,\{R\},\ \{Q\ false\}\,q\,\{R\}$ of our rule are valid. We prove that the conclusion $\{P\}\,\textbf{if}\ c\ \textbf{then}\ p\ \textbf{else}\ q\,\{R\}$ is valid as well.

Using Definitions 32, 34 and 35, what we have to prove amounts to:

- for all $s, s', s'' \in S$ and $a \in A$, if $c\ s = (true, s')$ and $p\ s' = (a, s'')$ then $R\ a\ s''$, and
- for all $s, s', s'' \in S$ and $a \in A$, if $c\ s = (false, s')$ and $q\ s' = (a, s'')$ then $R\ a\ s''$.

The first of these proof obligations is a consequence of $\{P\}\,c\,\{Q\},\ \{Q\ true\}\,p\,\{R\}$ being valid; while the second one is a consequence of $\{P\}\,c\,\{Q\}, \{Q\ false\}\,q\,\{R\}$ being valid. □

There is also a rule for sequencing. Its validity follows directly from the definitions:

Lemma 16. *In the context of Def. 34, let $P, R \subseteq S$, $Q \subseteq (\{*\} \times S)$, $p, q \in prog\ S\ \{*\}$. Then, the following Hoare rule for sequences is valid:*

$$\{P\}\,p\,\{\lambda\ \_\ s \to R\ s\},\ \ \{R\}\,q\,\{Q\} \vdash \{P\}\,(p\ ;\ q)\,\{Q\}.$$

We now focus on the rule that mainly interests us: the rule for the **while** loops.

Definition 36. *In the context Definition 34, let* $c \in \mathit{prog}\ S\ \mathit{bool}$, $p \in \mathit{prog}\ S\ \{*\}$, *and* $I \subseteq (\mathit{bool} \times S)$. *The Hoare rule for* **while** *loops is defined as follows:*

$$\{I\ \mathit{true}\}\, c\, \{I\},\ \ \{I\ \mathit{true}\}\, p\, \{\lambda\_\ s \to I\ \mathit{true}\ s\} \vdash \{I\ \mathit{true}\}\ \mathbf{while}\ c\ p\, \{\lambda\_\ s \to I\ \mathit{false}\ s\}.$$

The last result in the main paper is the validity of the last defined rule:

Proposition 13. *The Hoare rule for* **while** *is valid.*

Proof. From Haddock's theorem we know (♯): for all $c \in \mathit{prog}\ S\ \mathit{bool}$, $p \in \mathit{prog}\ S\ \{*\}$, $s \in S$: **while** $c\ p\ s = \mathit{lub}\, \{\mathit{While}_c^n\ p\ s \mid n \in \mathbb{N}\}$. The *lub* in the *rhs* of (♯) is computed in the flat CPO of $(\{*\} \times S)_\perp$. As a consequence (♯) amounts to (♭): for all $c \in \mathit{prog}\ S\ \mathit{bool}$, $p \in \mathit{prog}\ S\ \{*\}$, and $s, s' \in S$: **while** $c\ p\ s = (*, s') \leftrightarrow \exists n,\ \mathit{While}_c^n\ p\ s = (*, s')$.

From (♭) and Definition 34 we obtain moreover that for all $P \subseteq S$, $Q \subseteq (\{*\} \times S)$, $\{P\}$ **while** $c\ p\, \{Q\}$ is valid iff there exists $n \in \mathbb{N}$ such that $\{P\}\ \mathit{While}_c^n\ p\, \{Q\}$ is valid.

Hence, to prove the validity of the Hoare rule for **while** it suffices to prove that (†): for all $n \in \mathbb{N}$, $c \in \mathit{prog}\ S\ \mathit{bool}$, $p \in \mathit{prog}\ S\ \{*\}$, and $I \subseteq (\mathit{bool} \times S)$, it holds that

$$\{I\ \mathit{true}\}\, c\, \{I\},\ \ \{I\ \mathit{true}\}\, p\, \{\lambda\_\ s \to I\ \mathit{true}\ s\} \vdash \{I\ \mathit{true}\}\ \mathit{While}_c^n\ p\, \{\lambda\_\ s \to I\ \mathit{false}\ s\}.$$

where $\mathit{While}_c^n\ p$ has taken the place of **while** $c\ p$ in the Hoare rule for **while**.

To prove the proposition there only remains to prove (†) above by induction on n.

- in the base case $n = 0$, by definition, $\mathit{While}_c^0 = \lambda(\_ : S) \to \perp$, hence, the triple $\{I\ \mathit{true}\}\ \mathit{While}_c^n\ p\, \{\lambda\_\ s \to I\ \mathit{false}\ s\}$ is valid, since for no $s \in (I\ \mathit{true})$ does there exist s' such that $\mathit{While}_c^0\ s = (*, s')$; vacuously, all such $(*, s')$ satisfy the postcondition;
- for the induction step: assume the statement holds for $n = m$; we prove it for $n = m + 1$. By definition, $\mathit{While}_c^{m+1}\ p =$ **if** c **then** $(p\ ;\ (\mathit{While}_c^m\ p))$ **else ret** $*$. Hence, in order to prove the induction step, i.e., for arbitrary c, p, I and m of appropriate types,

  (‡) $\{I\ \mathit{true}\}\, c\, \{I\},\ \ \{I\ \mathit{true}\}\, p\, \{\lambda\_\ s \to I\ \mathit{true}\ s\} \vdash$

  $\{I\ \mathit{true}\}\ \mathbf{if}\ c\ \mathbf{then}\ (p\ ;\ (\mathit{While}_c^m\ p))\ \mathbf{else\ ret}\ *\ \{\lambda\_\ s \to I\ \mathit{false}\ s\}$

  we use on the conclusion of the (‡) entailement the rule for **if_then_else**$\_$, cf. Def.35, which was proved valid in Lemma 15. We apply the rule specialized with $P := I\ \mathit{true}$, $Q := I$, $R := \lambda\_\ s \to I\ \mathit{false}\ s$, $c := c$, $p := p\ ;\ (\mathit{While}_c^m\ p)$ and $q :=$ **ret** $*$.

  What remains to be proved now becomes:

  – $\{I\ \mathit{true}\}\, c\, \{I\}$ is valid: this is a hypothesis of the entailment (‡);
  – $\{I\ \mathit{true}\}\, (p\ ;\ (\mathit{While}_c^m\ p))\, \{\lambda\_\ s \to I\ \mathit{false}\ s\}$ is valid: by applying the Hoare rule for sequencing, cf. Lemma 16 specialized with $P, R := I\ \mathit{true}$, $Q = \lambda\_\ s \to I\ \mathit{false}\ s$, $p := p$ and $q := \mathit{While}_c^m\ p$ what we need to prove reduces to the validity of:
    * $\{I\ \mathit{true}\}\, p\, \{\lambda\_\ s \to I\ \mathit{true}\ s\}$, which is a hypothesis of the entailment (‡);
    * $\{I\ \mathit{true}\}\ \mathit{While}_c^m\ p\, \{\lambda\_\ s \to I\ \mathit{false}\ s\}$, the conclusion of our induction hypothesis, which we obtain from the induction hypothesis and both hypotheses of (‡).
  – $\{I\ \mathit{false}\}\ \mathbf{ret} *\ \{\lambda\_\ s \to I\ \mathit{false}\ s\}$ is valid: this follows from **ret** $* = \lambda\, s \to (*, s)$. □

6. About the Implementation

We have implemented the theory presented earlier in the paper as a library of the Coq proof assistant, and have applied the results to the examples from the previous section. Hence we obtain strong guarantees about the overall correctness of the paper. We next briefly present the structure of the library and some design choices we made. We also highlight some differences between theory and implementation, and present some lessons we learned, which will help in an upcoming re-engineering of the library.

6.1. Structure

The implementation is, mainly, a Coq library consisting of files that form several layers:

- A first layer corresponds to the theory presented in the Preliminaries (Section 2): definitions and results on sets and orders (posets, PPOs, CPOs, and algebraic CPOs with their completion and closure results); continuity with Kleene's least fixpoint theorem; and coinduction with the Knaster-Tarski theorem for greatest fixpoints.

- A second layer contains two coinductive types that are isomorphic to instances of Partial Containers from Section 3: streams and Rose trees. The coinductive types are obtained, like the Partial Containers, as completions of their respective finite approximations, with constructor-like functions obtained as completions of the respective constructors of finite approximations. Each instance comes with its own coinductive notion of totality and of bisimulation, and with the associated proof techniques.

- A third layer corresponds to the theory presented in Section 4: Haddock continuity and its equivalence with continuity (but based on *lub*s in simpler CPOs); and Haddock's theorem, which uses Haddock continuity as the condition to be met by the functional of a function under definition (instead of continuity in Kleene's theorem).

The Coq development contains the four use cases described in Section 5 as well.

6.2. Design Choices and Differences with Theory

The Coq development generally follows the theory in the paper. Of course, there are also differences. The main difference is the one between set theory, which is used in the paper, and Coq's type theory; this difference has induced some design choices, some of which are described below. Other design choices are induced by limitations of Coq (or by our own limitations as Coq developers) and by pragmatic considerations.

Set Theory vs. Type Theory. In the paper we have defined natural completion of a poset as adding certain new elements to the base set and extending the order to the new elements. Then, natural completion of a monotonic function between posets just extends the said function to the new elements. None of this is possible in type theory: functions are between types, *not* between sets, and adding inhabitants to a type is impossible.

What one can do (and we did) is to define new types, with injections between old and new types and properties ensuring that the injections are bijections between the old types and the subtypes of compact elements of the new types. Moreover, the completed functions are between the new types, and are not, strictly speaking, extensions of the

functions they complete, since the latter are between the old types. The relation between the completed functions and the ones they complete involves compositions with the injections between old and new types; which complicates matters, in practice.

The set theory that we use in the paper includes classical logic, Hilbert's choice operator, and other axioms implicitly accepted in standard mathematics. In our development these axioms have to be explicitly imported from the Coq's standard library.

Limitations of Coq (or of our proficiency therein). Other complications are induced by limitations of Coq and could perhaps have been avoided had we known more. For example, the series of increasingly complex orders in the paper is not linear: both PPOs and DCPOS extend posets, in different ways; the two extensions are "joined" in CPOs. This is not a problem in a light, set-theoretical setting. By contrast, in type theory, we encode posets as a record structure consisting of a *carrier type* and logical properties defining what an order is; then, an order with additional structure includes an order with less structure plus fields for the additional structure. In this way, one can obtain DCPOs and PPOs from posets. But for CPOs one needs a sort of union of fields of the records for DCPOs and PPOs, respectively, and, to our best knowledge, there is no simple mechanism to do that with records in Coq[8]. Hence we took the pragmatic decision *not* to implement DCPOs, which has consequences: the sum of (algebraic) CPOs is now an (algebraic) CPO, called "separated sum" in the literature; hence, lists are algebraic CPOs, with a $\perp$ element; and all functions on list (*map*, *rev*, taking the *i*-th element...) that we use in the examples need to take $\perp$ into account. As a result, defining the *mirror* function (as, on paper, in Section 5.1.2) is more involved in Coq because of technical difficulties that arise from working in a CPO instead of a DCPO.

Pragmatic considerations. We have taken the decision not to implement Partial Containers (cf. Section 2) in general, because this would have taken a lot of time and is not directly usable in examples. We have instead implemented two concrete examples of coinductive types that are isomorphic to instances of Partial Containers: streams and Rose trees. Hence, we have proved twice that completed constructors act as constructors of the respective completions, we have defined totality and bisimulation twice, and have proved twice that bisimulation is equivalent to equality. An upcoming reengineering of the library will implement Partial Containers and transport all their related notions and results to their isomorphic instances, via the isomorphism in question.

6.3. Lessons Learned

- Dependent types should be used scarcely: it is tempting to use dependent types, to closely mimic mathematical definitions; but eventually, the complications they induce may become unmanageable. This happened to us, for example, in the definition of *lub* in a CPO, which took as argument a proof that its main argument is directed. When *lub*s started occurring everywhere, and in different CPOs per Haddock's theorem, the complications became overwhelming. The proof argument was removed and the changes propagated in the whole development of several thousand lines.

[8] In a previous implementation we have used *coercions*. But coercions come with their own complications.

- Hiding details only works for a while: implicit arguments, which hide some of the details in a development, worked until we had, per Haddock's theorem, to deal with several CPOs at once. From there on implicit arguments had to be filled in by hand.
- Existing libraries should be used to save work: in the current development we have redefined everything from sets up. A better solution would have been to use (and to enrich) a library of mathematical notions and results, preferably one that already has, e.g., basic notions on CPOs, continuity, etc. Coq has the *mathematical components* library, but we could not find anything about CPOs in it. This is one of the reasons for our decision to reimplement our development in Lean, a proof assistant similar to Coq, whose extensive mathematical library covers the basics of what we need.

7. Conclusion, Related, and Future Work

Defining and reasoning about partial (co)recursive functions requires several ingredients. First, a notion of possibly partial coinductive types for the outputs of corecursive functions (which also work for recursive functions as particular cases), together with constructors making it possible to build terms for the the types in question. Second, tools for coinductive reasoning about totality (or total definedness) of terms, about equality of terms by means of bisimulation, and about other, user-defined coinductive relations involving the terms in question. Third, a technique for defining corecursive functions as fixpoints of their functionals built, among others, using constructors of the above-mentioned coinductive types, and tools for proving, among others, the partiality or totality of the resulting functions. In this paper we propose practical solutions for all these problems, present applications on concrete examples, and discuss the overall implementation of the approach and of the examples in the Coq proof assistant.

7.1. Limitations

Mutually coinductive types and mutually inductive-coinductive types are not supported. Neither are mutual corecursive functions, nor mutually recursive-corecursive functions. Interestingly enough, nested coinductive types and nested corecursive functions are supported: an example are Rose trees and their *mirror* function. However, mutually inductive types can be equivalently transformed into non-mutual indexed inductive types [17]; and it is well-known that mutually recursive functions can be reduced to non-mutually recursive ones [5] [Sec. 6.6.4]. Inspiration from these works might enable us to enlarge the class of types and functions supported by our approach.

7.2. Related Work

We group related work into several categories: classical results in domain theory and denotational semantics; the support for partial (co)recursive functions in proof assistants; and, finally, *coalgebra*, a whole other approach to coinduction.

Classical Results. We use a subset of domain theory corresponding to the initial chapters of [2, 3] - roughly, the same subset used in denotational semantics [4, 5]. There are, however, significant differences between denotational semantics and our approach.

First, denotational semantics starts from syntax and defines everything, including inductive types and total recursive functions. By contrast, we use existing inductive types (e.g., lists) and total recursive functions (e.g., *map*, *rev*) provided to us by Coq; redefining them makes no sense since by doing so we lose Coq's support for them.

Second, there are s differences in the manner of establishing *continuity* in order to use a fixpoint theorem for definition purposes. Denotational semantics is concerned with defining a fixed number of language constructs. To this end it starts from very elementary functions and proves their continuity; e.g., the function that returns the least fixpoint of a given continuous function is continuous. Then, by using continuity-preserving compositions, the functions defining the language constructs under definition are continuous; the definitions themselves are obtained using Kleene's fixpoint theorem. This is adequate when defining known-in-advance constructs of a language.

Our situation is different, since we don't know in advance which partial (co)recursive functions a user may want to define. Hence we provide users with support: Haddock's fixpoint theorem and an easier-to-check notion of continuity, based on least upper bounds in simpler CPOs. Using them one can reduce the complexity of CPOs where least upper bounds are taken, perhaps in several iterations, until one obtains simple enough CPOs where least upper bounds are simple enough to be manageable.

Tool-wise, a domain-theoretical framework called HOLCF [18] has been implemented on top of Isabelle/HOL and used in the denotational semantics of a significant fragment of Haskell [19]. Continuity is proved compositionaliy as explained above.

Partial (co)recursive functions in proof assistants. Partial *recursive* functions in Coq have been explored in [20]) and, with several different aproaches, in [21, Chapter 7.2] . Both rely of Kleene's theorem, and both note the difficulty of proving continuity; to our best knowledge they do not propose solutions for this issue, In another approach [22] a partial recursive function's codomain is a *thunk* - a parameterized coinductive type that "promises" an answer as a value of its parameter, but may postpone this answer forever, yielding nontermination. However, the functions being defined now become *corecursive* functions, which are very restricted in the Coq proof assistant. As a result, only *tail-recursive* functions can be defined with this approach (cf. [21, Chapter 7.3]).

Another way of encoding partial functions into total ones consists in adding an auxiliary *proof argument* to the partial function, which restricts the main argument to be in a subset of the function's domain where the function is total. With *this* definition of partial functions, all proof assistants based on dependent-type theory (Coq, Agda, Lean, ...) allow partial recursive functions. However, carrying the proof argument when composing such partial functions is cumbersome. Isabelle/HOL's type system does not allow proof arguments, but allows *domain predicates* playing a similar role.

On corecursive functions the proof-argument approach has been experimented in Coq on the *filter* function on streams in [23] and generalized in [24]. They transform unguarded corecursive calls into guarded ones, by skipping unproductive recursive calls and going directly to the next productive call, whose existence is guaranteed

by the proof argument. However, they do not handle the case where corecursive calls are guarded by non-constructor functions, such as our mirror function for Rose trees.

Corecursive functions are present in all the above-mentioned proof assistants, and others. In principle all the functions they define are total, but encodings of partial functions (using proof arguments, or domain predicates) somewhat bridge the gap.

Agda offer a basic support for corecursive functions similar to that of Coq, if somewhat more liberal, enabling for example mutually inductive-coinductive types and, based on those, a direct definition of the *mirror* function on Rose trees. Extensions of Agda include sized types [13] that provide users with a uniform, automatic way of handling termination and productiveness. The current implementation of sized types in Agda is unsound (cf. `https://github.com/agda/agda/issues/3026`). An implementation of sized types in Coq is proposed in [25].

Isabelle/HOL also offers basic and advanced support for defining corecursive functions. The basic support enables defining functions in the guarded-by-construction fragment. Advanced support [14] accepts functions beyond that fragment: corecursive calls can also be guarded by functions other than constructors, provided the functions are *friendly* (a friendly function needs to destruct at most one constructor of input to produce one constructor of output). Unguarded corecursive calls are also accepted, provided they eventually produce a constructor of output. Specific proof techniques for establishing friendliness are provided. The resulting corecursive functions are total.

Support for coinduction has recently been added to Lean [26]. The underlying theory is related to *coalgebra*, described below. To our best understanding the end result is that Lean only accepts, in principle, guarded-by-constructor (hence, total) corecursive functions. The extension enabling this is only partially implemented [27].

In a previous paper [28] we propose a method for defining total corecursive functions in Coq by replacing syntactical guardedness with a semantical notion of *productiveness* : for each input, an arbitrarily close approximation of the corresponding output is eventually produced. A fixpoint theorem is used for defining such functions as unique fixpoints of their productive functionals operating on CPOs. However, the CPO construction in [28] is ad hoc, which limits the manner in which corecursive functions are defined. Moreover no support for coinductive proofs is available.

The comparison of all the above with our approach can be summarized as follows: we do not attempt to relax the guardedness-by-construction *totality* criterion because we are not trying to define total functions. We define (an encoding of) partial functions using a version of Kleene's theorem with a condition equivalent to continuity but easier to check in practice. The totality of the resulting function can be proved later, if needed.

Coalgebra. This is a category-theoretical framework where coinductive types, corecursive functions and coinductive proofs are first-class citizens [29] . The domain is vast; we mainly focus on one article from this field, which has many similarities but also essential differences with our work. In [30] ideal completion of the *initial algebras* of any *polynomial functor* is shown isomorphic to the *final coalgebra* of the same functor. Our approach exhibits some similarities with the above: our partial containers are kin to polynomial functors (where $\perp$ becomes a constant term) and, like in a final coalgebra, bisimulation coincides with equality. However, there are also differences:

the partial order subject to ideal completion in [30] is *not* the definition order[9]. The practical consequence of an order other than the definition order for us is that constructors are not monotonic any more, hence, they cannot be completed, and anything in our approach that relies on completed constructors is broken. This includes essential features such as functionals for the functions of interest, and the structure of elements in partial containers; without these feature there is no corecursion/coinduction left. This no criticism of [30]: we have different objectives, and their choices fit their objectives.

Still in the coalgebraic setting, in [31] it is proved that, in the presence of inductive types in homotopy type theory, coinductive types are derivable. This has been improved in [32] by having the computation rule for corecursion hold judgmentally.

7.3. Future Work

As future work we plan to develop a framework for defining and using coinductive types and (partial) corecursive functions, based on the theory presented herein, in the Lean proof assistant, which lacks native support for coinduction. The framework will also deal with partial recursive functions as a particular case [10]. We stress that we intend this framework to be not just a new formalization of the results in the paper, but also a user-friendly system, with convenient syntax that hides many of its implementation details, and various forms of automation. To this end, the system will include:

- A formalization of the required domain theory results from Section 2. Rather than starting from scratch, we will make heavy use of Lean's extensive mathematical library, mathlib [33], which provides a good starting point on order theory, including definitions and theorems about (pointed) partial orders, lubs of directed sets, and order ideals. However, at the time of writing, the library lacks some notions that are critical to us. For example, complete partial orders have only recently been added, and don't have many associated results and definitions. Other notions, like algebraic CPOs, compact elements of a CPO, or completions are not yet present, and will need to be done by us. As such, as a by-product of our project, we will be able to contribute these to mathlib, thus extending the Lean ecosystem.

- An implementation of partial containers in their full generality and their associated theorems, as described in Section 3. Based on it, we will obtain different particular examples of coinductive types, like (types isomorphic to) streams or Rose trees, as instantiations thereof. The approach carries the important benefit that all theorems will be stated and proved only once, in the general version, and they will follow easily for every user-defined example of a coinductive type.

- Proofs of the fixed point theorems from Section 4: Kleene's fixed point theorem, the equivalence between continuity and H-continuity, and Haddock's theorem.

[9] Specifically, the order, say, $\lesssim$, used in [30] for trees says that two trees are ordered whenever the first one is equal to a "cut" of the second one at a given height. This is not the definition order, because constructors are not monotonic: e.g., for all trees t, $\bot \lesssim t$ and $t \lesssim t$, but $tree\,[\bot, t] \not\lesssim tree\,[t, t]$ whenever $t \neq \bot$.

[10] This is different from Lean's built-in *partial def* command, which allows for potentially nonterminating recursion. Definitions given in that way are opaque and, in particular, one cannot prove theorems about them.

- A user-friendly interface, built using Lean's metaprogramming capabilities [34], that hides the domain theoretical details of our implementation. Instead, a user would be able to specify coinductive types via an intuitive syntax, similar, for example, to Coq's *CoInductive* command, or to that from [27], which will then be elaborated into instantiations of partial containers. Similarly, a special syntax can be added for (partial) (co)recursive functions, where the user writes a simple self-referential definition, which is then elaborated into the lub of an inferred functional. Other automations can be set up, like tactics that try to prove that a functional is H-continuous, with proof obligations assigned to the user in cases where the default tactics fail. Such a system is crucial if we want other users to adopt our framework.

Another future work direction is exploiting our shallowly embedded, monadic imperative language (including the *while* loops defined in the paper) and its Hoare logic for specifying, programming, and verifying algorithms with pointer-based structures (i.e., linked lists, linked trees, etc). We anticipate that imperative procedures over linked structures will be specified by Hoare triples involving coinductive types that are abstract descriptions of possibly-infinite linked structures, where infinity may occur from an unintended cycle created by self-links. Then, a coinductive relation would connect the concrete linked structure and the coinductive type describing it, and a partial corecursive function operating on the coinductive type would specify the imperative procedure working on the concrete linked structure. Such an approach brings together all the elements we introduced in this paper in a common framework, which we intend to apply to system-level code, building on our earlier experiences in this area [35, 36].

References

[1] The Haskell Programming Language, `https://www.haskell.org/`.

[2] R. M. Amadio, P. Curien, Domains and lambda-calculi, Vol. 46 of Cambridge tracts in theoretical computer science, Cambridge University Press, 1998.

[3] V. Stoltenberg-Hansen, I. Lindström, E. R. Griffor, Mathematical theory of domains, Vol. 22 of Cambridge tracts in theoretical computer science, Cambridge University Press, 1994.

[4] G. Winskel, The formal semantics of programming languages - an introduction, Foundation of computing series, MIT Press, 1993.

[5] D. A. Schmidt, Denotational semantics: a methodology for language development., Allyn & Bacon, 1997.

[6] G. Kahn, The semantics of a simple language for parallel programming, in: J. L. Rosenfeld (Ed.), Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974, North-Holland, 1974, pp. 471–475.

[7] C. Paulin-Mohring, A constructive denotational semantics for Kahn networks in Coq, `https://www.lri.fr/~paulin/PUBLIS/paulin07kahn.pdf`.

[8] Haskell/denotational semantics, https://en.wikibooks.org/wiki/Haskell/Denotational_semantics.

[9] The Coq Proof Assistant, https://coq.inria.fr/.

[10] The Isabelle/HOL Proof Assistant, https://isabelle.in.tum.de/.

[11] The Agda Proof Assistant, https://agda.readthedocs.io.

[12] The Lean Proof Assistant, https://leanprover.github.io/.

[13] N. Veltri, N. van der Weide, Guarded recursion in agda via sized types, in: 4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany, Vol. 131 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 32:1–32:19.

[14] J. C. Blanchette, A. Bouzy, A. Lochbihler, A. Popescu, D. Traytel, Defining Non-primitively (Co)recursive Functions in Isabelle/HOL, https://isabelle.in.tum.de/dist/Isabelle2021/doc/corec.pdf.

[15] M. G. Abbott, T. Altenkirch, N. Ghani, Containers: Constructing strictly positive types, Theor. Comput. Sci. 342 (1) (2005) 3–27. doi:10.1016/j.tcs.2005.06.002.

[16] D. Nowak, V. Rusu, While loops in Coq, in: Proceedings 7th Symposium on Working Formal Methods, FROM 2023, Bucharest, Romania, 21-22 September 2023, Vol. 389 of EPTCS, 2023, pp. 96–109.

[17] A. Kaposi, J. von Raumer, A Syntax for Mutual Inductive Families, in: Z. M. Ariola (Ed.), 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), Vol. 167 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 23:1–23:21. doi:10.4230/LIPIcs.FSCD.2020.23.

[18] O. Müller, T. Nipkow, D. von Oheimb, O. Slotosch, Holcf=hol+lcf, J. Funct. Program. 9 (2) (1999) 191–223. doi:10.1017/s095679689900341x.

[19] J. Breitner, B. Huffman, N. Mitchell, C. Sternagel, Certified HLints with Isabelle/HOLCF-Prelude, CoRR abs/1306.1340 (2013).

[20] Y. Bertot, V. Komendantsky, Fixed point semantics and partial recursion in Coq, in: Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain, 2008, pp. 89–96.

[21] A. Chlipala, Certified Programming with Dependent Types, MIT Press, 2013.

[22] V. Capretta, General recursion via coinductive types, Log. Methods Comput. Sci. 1 (2) (2005) 1–18. doi:10.2168/LMCS-1(2:1)2005.

[23] Y. Bertot, Filters on coinductive streams, an application to Eratosthenes' sieve, in: Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings, Vol. 3461 of Lecture Notes in Computer Science, 2005, pp. 102–115.

[24] Y. Bertot, E. Komendantskaya, Inductive and coinductive components of corecursive functions in Coq, in: Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008, Vol. 203 of Electronic Notes in Theoretical Computer Science, 2008, pp. 25–47.

[25] J. Chan, W. J. Bowman, Practical sized typing for Coq, CoRR abs/1912.05601 (2019).

[26] J. Avigad, M. Carneiro, S. Hudon, Data types as quotients of polynomial functors, in: 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA, Vol. 141 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 6:1–6:19.

[27] A. C. Keizer, Implementing a definitional (co)datatype package in Lean 4, based on quotients of polynomial functors, Master's thesis, Universiteit van Amsterdam (2023).

[28] V. Rusu, D. Nowak, Defining corecursive functions in Coq using approximations, in: ECOOP, Berlin, Germany, 2022, pp. 12:1–12:24.
URL https://hal.inria.fr/hal-03671876

[29] J. J. M. M. Rutten, Universal coalgebra: a theory of systems, Theor. Comput. Sci. 249 (1) (2000) 3–80. doi:10.1016/S0304-3975(00)00056-6.

[30] J. Adámek, Final coalgebras are ideal completions of initial algebras, J. Log. Comput. 12 (2) (2002) 217–242. doi:10.1093/logcom/12.2.217.

[31] B. Ahrens, P. Capriotti, R. Spadotti, Non-wellfounded trees in homotopy type theory, in: T. Altenkirch (Ed.), 13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland, Vol. 38 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 17–30. doi:10.4230/LIPICS.TLCA.2015.17.

[32] F. Rech, Strictly positive types in homotopy type theory, Master's thesis, Saarland University (2017).

[33] The mathlib Community, The lean mathematical library, in: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 367–381. doi:10.1145/3372885.3373824.

[34] S. Ullrich, L. de Moura, Beyond notations: Hygienic macro expansion for theorem proving languages, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II, Vol. 12167 of Lecture

Notes in Computer Science, Springer, 2020, pp. 167–182. doi:10.1007/978-3-030-51054-1_10.

[35] N. Jomaa, P. Torrini, D. Nowak, G. Grimaud, S. Hym, Proof-Oriented Design of a Separation Kernel with Minimal Trusted Computing Base, in: 18th International Workshop on Automated Verification of Critical Systems (AVOCS 2018), Oxford, United Kingdom, 2018. doi:10.14279/tuj.eceasst.76.1080. URL https://hal.science/hal-01816830

[36] F. Vanhems, V. Rusu, D. Nowak, G. Grimaud, A formal correctness proof for an EDF scheduler implementation, in: 28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, Milano, Italy, May 4-6, 2022, IEEE, 2022, pp. 281–292. doi:10.1109/RTAS54340.2022.00030.

Appendix: Proofs from Preliminaries (Section 2)

Completion

Proposition 2. *Any poset has a natural completion, which is unique up to isomorphism.*

Proof. The proof is an instance of the general fact that a bijection with a structured domain endows its codomain with a structure isomorphic to that of the domain. From any given poset $(C^\circ, \preceq^\circ)$ one obtains the ideal completion $(\mathcal{I}_{C^\circ}, \subseteq)$, which is an algebraic DCPO having the poset of compacts $(\mathcal{P}_{C^\circ}, \subseteq)$. Let $C := C^\circ \cup (\mathcal{I}_{C^\circ} \setminus \mathcal{P}_{C^\circ})$. Define $\eta : \mathcal{I}_{C^\circ} \to C$ by $\eta\ I = x$ if $I = {\downarrow}\{x\}$ and $\eta\ I = I$ if $I \notin \mathcal{P}_{C^\circ}$. The function η is a bijection; let $\eta^{-1} : C \to \mathcal{I}_{C^\circ}$ be its inverse. Define $\preceq\ \subseteq\ C \times C$ by $c \preceq c'$ iff $\eta^{-1}\ c\ \subseteq\ \eta^{-1}\ c'$. We prove that $(C, \preceq)$ is an algebraic DCPO whose poset of compacts is $(C^\circ, \preceq^\circ)$, i.e., $(C, \preceq)$ is a natural completion of $(C^\circ, \preceq^\circ)$, and that $(C, \preceq)$ is isomorphic with $(\mathcal{I}_{C^\circ}, \subseteq)$.

Regarding uniqueness: consider any natural completion $(D, \precsim)$ of $(C^\circ, \preceq^\circ)$ - i.e., $(D, \precsim)$ is an algebraic DCPO and $(C^\circ, \preceq^\circ)$ is the poset of compacts of $(D, \precsim)$. By Prop. 1, $(D, \precsim)$ is isomorphic with $(\mathcal{I}_{C^\circ}, \subseteq)$. But, from above, $(C, \preceq)$ is also isomorphic with $(\mathcal{I}_{C^\circ}, \subseteq)$. Hence natural completion is unique up to isomorphism. □

Product

We prove Proposition 5 regarding the product of algebraic CPOs. A series of intermediary lemmas are required. The first one also appear in the paper's body.

Lemma 1. *For any directed set $S \subseteq C = \Pi_{j \in J} C_j$ and $j \in J$, $S \Downarrow j$ is directed, and $lub\ S = \lambda\, j \to (lub\ (S \Downarrow j))$.*

Proof. We start by proving that $S \Downarrow j$ is directed. First, $S \Downarrow j$ is nonempty as a projection of S, which, being directed, is nonempty. Second, let $s_j, s'_j \in S \Downarrow j$. By definition of $S \Downarrow j$, $s_j, s'_j \in C_j$, and for all $i \in J \setminus \{j\}$, there exist $s_i, s'_i \in C_i$ such that $(\lambda\, i \to s_i), (\lambda\, i \to s'_i) \in S$. Since S is directed, there exists $\lambda\, i \to s''_i \in S$ such that $(\lambda\, i \to s_i), (\lambda\, i \to s'_i) \preceq \lambda\, i \to s''_i \in S$. In particular, on the jth components, $s_j, s'_j \preceq_j s''_j$, and $s''_j \in S \Downarrow j$, by definition of the set $S \Downarrow j$; which concludes the proof of directedness of $S \Downarrow j$.

We next show that $l := \lambda\, j \to (lub\ (S \Downarrow j))$ is the least upper bound of S.

- upper bound (for all $c \in S$, $c \preceq l$): c is of the form $\lambda\, j \to c_j$, and, for all $j \in J$, by definition of $S \Downarrow j$, $c_j \in S \Downarrow j$, hence, for all $j \in J$, $c_j \preceq_j lub\ (S \Downarrow j)$, which implies that $c = \lambda\, j \to c_j \preceq \lambda\, j \to (lub\ (S \Downarrow j)) = l$.

- minimality (for all $c' \in C$, if c' is an upper bound for S then $l \preceq c$'): choose an arbitrary upper bound $c' \in C$ for S. Choose an arbitrary $j \in J$ and consider an arbitrary $c_j \in S \Downarrow j$; hence, there are c_i, for all $i \in J \setminus \{i\}$ such that $c := \lambda j \to c_j \in S$, hence, by the choice of c' as upper bound for S, $c \preceq c'$, which implies in particular on the jth component, $c_j \preceq_j c'_j$, hence, c'_j is an upper bound for $S \Downarrow j$. It follows that $lub(S \Downarrow j) \preceq_j c'_j$, and since $j \in J$ was chosen arbitrarily, $l = \lambda\, j \to (lub\ (S \Downarrow j)) \preceq \lambda j \to c'_j = c'$; which proves the minimality of the upper bound l and the lemma. □

The next lemma is a companion to the previous one; it deals with the *lub* of a product.

Lemma 17. *Assume, for all $j \in J$, a directed set $S_j \subseteq C_j$. Then, the set $S = \Pi_{j\in J} S_j$ is directed, and lub $S = \lambda j \to (lub\ S_j)$.*

Proof. We successively prove the following statements:

- (i) S is directed: it is nonempty as the Cartesian product of directed, hence nonempty sets. For arbitrary $s, s' \in S$, it holds that for all $j \in J$, $s_j, s'_j \in S_j$, and since all the S_j are directed, there exists $s''_j \in S_j$ such that $s_j, s'_j \preceq_j s''_j$, and then $s, s' \preceq \lambda\, j \to s''_j \in \Pi_{j\in J} S_j = S$; the directedness is proved.

- (ii) for all $j \in J$, $S \Downarrow j = S_j$: since $S = \Pi_{i\in J} S_i$ (ii) is a consequence of general results about relations between Cartesian products and their projections;

- (iii) $lub\ S = \lambda\, j \to (lub\ S_j)$: we know from Lemma 1 that $lub\ S = \lambda\, j \to (lub\ (S \Downarrow j))$. By item (ii), for all $j \in J$, $S \Downarrow j = S_j$. (iii) follows, and the lemma is proved. □

Lemma 18. *For any indexed set $\{(C_j, \preceq_j, \perp_j) \mid j \in J\}$ of algebraic CPOs, with respective sets of compacts C°_j, the product $\Pi_{j\in J}(C_j, \preceq_j, \perp_j)$ is a CPO having the set of compacts $\{c \in \Pi_{j\in J} C^\circ_j \mid \|c\| \text{ is finite}\}$.*

Proof. The fact that the product $\Pi_{j\in J}(C_j, \preceq_j, \perp_j)$ is a CPO follows from the observtion that the product is already a PPO and from Lemma 1 which ensures the existences of least upper bounds for its directed sets. There remains to show that the product has exactly the set of compacts $\{c \in \Pi_{j\in J} C^\circ_j \mid \|c\| \text{ is finite}\}$.

We first show $(*)$: every $c \in \Pi_{j\in J} C^\circ_j$ with $\|c\|$ finite is compact. Assume $c \preceq lub\ S$ for some directed set S. We have $c = \lambda\, j \to c_j$ and by Lemma 1, $lub\ S = \lambda\, j \to (lub\ (S \Downarrow j))$. Hence $c \preceq lub\ S$ translates to $(**)$ for all $j \in J$, $c_j \preceq_j lub\ (S \Downarrow j)$. Now, by definition of $\|c\|$, $(**)$ amounts to: to for all $j \in \|c\|$, $c_j \preceq_j lub\ (S \Downarrow j)$. (Indeed, for $j \in J \setminus \|c\|$, $c_j \preceq_j lub\ (S \Downarrow j)$ holds trivially because $c_j = \perp_j$).

If $\|c\| = \emptyset$ then $c = \lambda\, j \to \perp_j$, which is trivially compact: if $c \preceq lub\ S$ for some directed (hence nonempty) S, then any $c' \in S$ satisfies $c \preceq c'$, and $(*)$ is proved for c.

If $\|c\| \neq \emptyset$, fix an arbitrary $j \in \|c\|$. Since c_j is compact, there exists $d^j_j \in S \Downarrow j$ such that $c_j \preceq_j d^j_j$. Now, $d^j_j \in S \Downarrow j$ implies that for all $i \in J \setminus \{j\}$, there exist $d^i_j \in C_i$ that, together with $d^j_j \in C_j$, make an element of S: $\lambda\, i \to d^i_j \in S$. But S is directed, hence, there is an upper bound $u := \lambda\, i \to u_i \in S$ for the (finite) subset $\{\lambda\, i \to d^i_j \mid j \in \overline{c}\}$ of S.

Hence for an arbitrary (fixed) $j \in \overline{c}$, $\lambda\, i \to d^i_j \preceq \lambda\, i \to u_i$, and in particular $d^i_j \preceq u_j$. From $c_j \preceq_j d^j_j$ above we obtain $c_j \preceq u_j$. Since $j \in \overline{c}$ was arbitrarily chosen, and for all $k \in J \setminus \lVert c \rVert$, $c_k = \bot_k$, we obtain $c = \lambda\,(j : J) \to c_j \preceq \lambda\,(j : J) \to u_j = u \in S$. Hence c is compact, which proves $(*)$ for c and $(*)$ as a whole.

In the other direction we first prove that $(***)$ any compact of $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ is in $\Pi_{j\in J}C^\circ_j$. Consider then an arbitrary compact $c = \lambda\, j \to c_j \in \Pi_{j\in J}C_j$ of the product. We have to show that for all $j \in J$, c_j is compact, i.e., $c_j \in C^\circ_j$. For this, consider also the set $S = \Pi_{j\in J}S_j \subseteq \Pi_{j\in J}C^\circ_j$ of elements in the product, such that for all $j \in J$, $S_j = \{c^\circ_j \in C^\circ_j \mid c^\circ_j \preceq_j c_j\}$. Since all the elements in the product $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ are algebraic CPOs, for all $j \in J$, S_j is directed and $c_j = lub_j\, S_j$. Using Lemma 17, S is directed and $c = lub\, S$, in particular $c \preceq lub\, S$. From the latter and the fact that c is compact, we obtain $c' \in S$ such that $c \preceq c'$. Hence, for all $j \in J$, $c_j \preceq_j c'_j$ with $c'_j \in S_j$, which from the definition of S_j means $c'_j \in C^\circ_j$ and $c'_j \preceq_j c_j$. Hence, for all $j \in J$, $c_j = c'_j$ and since $c'_j \in C^\circ_j$, we obtain that for all $j \in J$, $c_j \in C^\circ_j$; which is what we had to prove for $(***)$.

There only remains to prove that any compact of $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ has a finite carrier. Consider such a compact element $c^\circ = \lambda\, j \to c^\circ_j$. (By $(***)$ above we know that $c^\circ_j \in C^\circ_j$, for all $j \in J$.) Let S be the set of restrictions of c° to finite subsets of J; i.e., functions that coincide with c° on their finite carrier, and are equal to $\bot$ outside the carrier. Mathematically, $S = \{c'^\circ = \lambda\, j \to c'^\circ_j \mid \lVert c'^\circ \rVert \textit{ is finite} \wedge \forall k \in \lVert c'^\circ \rVert .\, c'^\circ\, k = c^\circ\, k\}$.

We now show that the set S is directed: indeed, it is nonempty as it contains at least $(\lambda\, j \to \bot_j)$, and if $c^\circ_1, c^\circ_2 \in S$, then consider c° such that for all $j \in J$:

- $c^\circ\, j = \bot$ if $c^\circ_1\, j = \bot$ and $c^\circ_2\, j = \bot$;
- $c^\circ\, j = c^\circ_1\, j$ if $c^\circ_1\, j \neq \bot$ and $c^\circ_2\, j = \bot$;
- $c^\circ\, j = c^\circ_2\, j$ if $c^\circ_1\, j = \bot$ and $c^\circ_2\, j \neq \bot$;
- $c^\circ\, j = c^\circ_2\, j$ if $c^\circ_1\, j \neq \bot$ and $c^\circ_2\, j \neq \bot$.

It is not hard to check that $c^\circ_1, c^\circ_2 \preceq c^\circ$ and that $c'^\circ \in S$. Hence S is directed.

Next, we prove that $c^\circ \preceq lub\, S$. For this, let $S' = \{c'^\circ \in S \mid |\lVert c'^\circ \rVert| \leq 1\}$ be the subset of S of functions that differ from $\bot$ in at most one point (and in that point, if any, they coincide with c°, by definition of S). We have $c^\circ = lub\, S'$: indeed, by definition of S', $c'^\circ \preceq c^\circ$ for all $c'^\circ \in S'$, meaning that c° is an upper bound for S'; and, assuming k an upper bound for S': by definition of S', ($c'^\circ \preceq k$ for all $c'^\circ \in S'$) is equivalent to ($c^\circ\, j \preceq k\, j$ for all $j \in J$), i.e., $c^\circ \preceq k$, which proves that $c^\circ = lub\, S'$; and since lub is monotonic, $c^\circ = lub\, S' \preceq lub\, S$, completing the proof of $c^\circ \preceq lub\, S$.

Finally, from $c^\circ \preceq lub\, S$ we obtain $c^\circ \preceq c'^\circ$, for some $c'^\circ \in S$, which implies $\lVert c^\circ \rVert \subseteq \lVert c'^\circ \rVert$, and since by definition of S, $c'^\circ \in S$ implies $\lVert c'^\circ \rVert$ is finite, we obtain that $\lVert c^\circ \rVert$ is finite too; which is what remained to prove in order to complete our proof. □

Lemma 19. *For any nonempty indexed set $\{(C_j, \preceq_j, \bot_j) \mid j \in J\}$ of algebraic CPOs, the product $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ is algebraic.*

Proof. From Proposition 5 we know that $\Pi_{j\in J}(C_j, \preceq_j, \bot_j)$ is a CPO $(C, \preceq, \bot)$ having the set of compacts $C^\circ = \{c \in \Pi_{j\in J}C^\circ_j \mid \lVert c \rVert \textit{ is finite}\}$. For all $c \in C$, let $S^\circ_c := \{c^\circ \in C^\circ \mid c^\circ \preceq c\}$. Fix an arbitracy $c \in C$. We have to prove that S°_c is directed and $c = lub\, S^\circ_c$.

1. S_c° is directed: first, $S_c^\circ \neq \emptyset$ as $(\lambda\, j \to \bot_j) \in S_c^\circ$. Second, consider $c'^\circ, c''^\circ \in S_c^\circ$. Hence, $\|c'^\circ\|$ and $\|c''^\circ\|$ are finite, and $c'^\circ = \lambda\, j \to c_j'^\circ$ and $c''^\circ = \lambda\, j \to c_j''^\circ$, such that for all $j \in J$, $c_j'^\circ, c_j''^\circ \in C_j^\circ$ and $c_j'^\circ, c_j''^\circ \preceq c_j$. We build $c_j^\circ \in C_j$ as follows:
   - $c_j^\circ = \bot$, if $c_j'^\circ = \bot$ and $c_j''^\circ = \bot$;
   - $c_j^\circ = c_j'^\circ$, if $c_j'^\circ \neq \bot$ and $c_j''^\circ = \bot$;
   - $c_j^\circ = c_j''^\circ$, if $c_j'^\circ = \bot$ and $c_j''^\circ \neq \bot$;
   - $c_j^\circ \in \{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j c\ j\}$ is an upper bound for $c_j'^\circ, c_j''^\circ$, if $c_j'^\circ \neq \bot$ and $c_j''^\circ \neq \bot$. (Such a $c_j^\circ \in \{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j c\ j\}$ exists because the set $\{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j c\ j\}$ is directed in the algebraic CPO $(C_j, \preceq_j, \bot_j)$ and $c_j'^\circ, c_j''^\circ \in \{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j c\ j\}$.)

   By the above construction, for all $j \in J$, $c_j^\circ \in C_j^\circ$ and $c_j^\circ \preceq_j c\ j$ and $c_j'^\circ, c_j''^\circ \preceq c_j^\circ$, i.e. $(\lambda\, j \to c_j^\circ) \preceq c$ and $c'^\circ, c''^\circ \preceq (\lambda\, j \to c_j^\circ)$. Let $c^\circ = \lambda\, j \to c_j^\circ$. Hence, $c^\circ \preceq c$ and $c'^\circ, c''^\circ \preceq c^\circ$, $c^\circ \in \Pi_{j\in J} C_j^\circ$ and moreover $\|c^\circ\| = \|c'^\circ\| \cup \|c''^\circ\|$, which, since $\|c'^\circ\|$ and $\|c''^\circ\|$ are finite, implies that $\|c^\circ\|$ is finite, which, in turn, implies $c^\circ \in C^\circ$. And since $c^\circ \preceq c$ it follows that $c^\circ \in S_c^\circ$.Overall, $c'^\circ, c''^\circ \preceq c^\circ$ and $c^\circ \in S_c^\circ$; proving that S_c° is directed.

2. $c = \mathit{lub}\ S_c^\circ$: by definition of $S_c^\circ = \{c^\circ \in C^\circ \mid c^\circ \preceq c\}$, c is an upper bound for S_c°. Assume $k \in C$ is an upper bound for S_c°. Fix an arbitrary $j \in J$. It follows that $k\ j$ is an upper bound for $\{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j (c\ j)\}$. But the set $\{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j (c\ j)\}$ is directed in the algebraic CPO $(C_j, \preceq_j, \bot_j)$, and $\mathit{lub}\,\{c_j^\circ \in C_j^\circ \mid c_j^\circ \preceq_j (c\ j)\} = c\ j$, which implies $c\ j \preceq_j k\ j$, for the arbitarily chosen $j \in J$; hence, $c \preceq k$, which proves the minimality of c, establishes $c = \mathit{lub}\ S_c^\circ$ and completes the proof. □

By combining Lemmas 18 and 19 we obtain a proof of Proposition 5.