The Optimal Fixed Point Combinator

Arthur Charguéraud

INRIA
arthur.chargueraud@inria.fr

Abstract. In this paper, we develop a general theory of fixed point combinators, in higher-order logic equipped with Hilbert's epsilon operator. This combinator allows for a direct and effective formalization of corecursive values, recursive and corecursive functions, as well as functions mixing recursion and corecursion. It supports higher-order recursion, nested recursion, and offers a proper treatment of partial functions in the sense that domains need not be hardwired in the definition of functionals. Our work, which has been entirely implemented in Coq, unifies and generalizes existing results on contraction conditions and complete ordered families of equivalences, and relies on the theory of optimal fixed points for the treatment of partial functions. It provides a practical way to formalize circular definitions in higher-order logic.

1 Introduction

1.1 Motivation: partial corecursive functions

To the best of our knowledge, there exists, until now, no general approach to formalizing partial corecursive functions in a simple and satisfying manner. Consider for example the filter function on infinite streams. Given a predicate P of type $A \to \mathsf{bool}$ (or $A \to \mathsf{Prop}$), the filter function f takes a stream s and returns a stream made of the elements of s that satisfy P. The filter function is partial because it produces a well-defined stream only when its argument s contains infinitely many items satisfying the predicate P.

One way to constructively formalize the definition of filter in a logic of total functions is to have f take as extra argument a proof that its argument contains infinitely many items satisfying the predicate P. In this approach, studied by Bertot [4], the new filter function does not have the type $\mathsf{stream}\, A \to \mathsf{stream}\, A$, but instead admits a dependent type. Unfortunately, working with dependent types is often associated with numerous technical difficulties, so we would rather find a solution that does not make such a heavy use of dependent types.

A different, non-constructive approach to formalizing the filter function was proposed by Matthews [18]. To apply his technique, the filter function first needs to be turned into a total function, by testing explicitly whether the argument belongs to the domain. Let "$\mathsf{never}\, P\, s$" be a predicate that holds when the stream s does not contain any item satisfying P. The body of the filter function can be described through a functional F, as follows. Throughout the paper, the operator ::

denotes the consing operation on infinite streams.

$$F\,f\,s \quad \triangleq \quad \mathsf{let}\ x :: s' = s\ \mathsf{in}\ \mathsf{if}\ (\mathsf{never}\,P\,s)\ \mathsf{then}\ \mathsf{arbitrary}\ \mathsf{else}\ \mathsf{if}\ (P\,s)\ \mathsf{then}\ x :: (f\,s')\ \mathsf{else}\ f\,s'$$

The filter function f can then be defined as "$\mathsf{Fix}_1\,F$", where Fix_1 is a combinator that picks, using Hilbert's epsilon operator, the unique fixed point of its argument when it exists, and otherwise returns an arbitrary value. Here, the functional F can be proved to admit a unique fixed point using a fixed point theorem based on *contraction conditions*, devised by Matthews [18]. It follows that f satisfies the fixed point equation $f\,s = F\,f\,s$ for any stream s.

The main downside of the approach described above is that the domain of the function needs to be hardwired in its definition. As argued in detail by Krauss [13] for the case of recursive functions, this requirement is unsatisfactory. First, it requires to modify the code of the functional, which is inelegant and may cause difficulties when extracting executable code. Second, it overspecifies the output of the function outside its domain. Third, it requires to know the domain of the function at the time of its definition, which is not always practical (see [13]).

The central matter of this paper is to construct a fixed point combinator Fix that truly supports partial functions. For example, Fix can be directly applied to the functional that describes the original filter function, shown below.

$$F\,f\,s \quad \triangleq \quad \mathsf{let}\ x :: s' = s\ \mathsf{in}\ \mathsf{if}\ (P\,s)\ \mathsf{then}\ x :: (f\,s')\ \mathsf{else}\ f\,s'$$

1.2 Fixed point equations with non-unique solutions

Most forms of circular definitions can be captured by (or encoded as) an equation of the form $a = F\,a$. Yet, such a fixed point equation does not necessarily admit a unique solution.

One typical case is that of partial functions. In a logic of total functions, a partial function can be represented as a pair of a total function f of type $A \to B$ and a domain D of type $A \to \mathsf{Prop}$. The partial function (f, D) is said to be a partial fixed point of a functional F of type $(A \to B) \to (A \to B)$ if the equation $f\,x = F\,f\,x$ holds for any x satisfying D. (We postpone to §3.3 the discussion of how circular definitions for partial functions can be expressed as equations of the form $a = F\,a$.) A functional F typically admits several partial functions as fixed point. Can one of them be considered the "best" fixed point for F?

The starting point of this paper is the observation that the exact answer to this question is given by the theory of the *optimal fixed point* developed in 1975 by Manna and Shamir [17], which we have formalized in Coq. A fundamental idea in this theory is that the only genuine solutions of a fixed point equation are the partial functions that are consistent with any other fixed point (two functions are consistent if they agree on the intersection of their domain). Such fixed points are said to be *generally-consistent*. The optimal fixed point is defined as the generally-consistent fixed point with the largest domain. In a sense, the optimal fixed pointis the most well-defined solution that can be extracted from the fixed

point equation without making arbitrary choices. Manna and Shamir [17] proved that any functional of type $(A \to B) \to (A \to B)$ admits an optimal fixed point.

Another typical case where fixed point equations do not not admit unique solutions occurs when working modulo equivalence relations. A value a is a fixed point of F modulo $(\equiv)$ if the equation $x \equiv F\,x$ holds for any value x such that $x \equiv a$. Moreover, a fixed point a is said to be the unique fixed point modulo $(\equiv)$ of a functional F if any other fixed point x of F is equivalent to a modulo $(\equiv)$. In this case, even though the functional F does not admit a unique solution, it admits a unique equivalence class of solutions. Thus, any element from this class of equivalence can be considered as representing the meaning of the the circular definition associated with F. The interest of the definition of "fixed point modulo" is that it allows defining recursive functions on values compared modulo an equivalence relation without involving an explicit quotient structure.

1.3 A generic fixed point combinator

In order to unify the various forms of circular definitions, we introduce a generic fixed point combinator. The basic idea is to pick the "best" fixed point, for a customizable notion of "best" that depends on the kind of circular definition being targeted. Our combinator, called Fix, takes as argument an equivalence relation $\equiv$, an order relation $\lhd$ and a functional F. It then uses Hilbert's epsilon operator to pick, among the set of all fixed points modulo $\equiv$ of the functional F, the greatest fixed point with respect to $\lhd$ (not be confused with the greatest fixed point in the sense of domain theory).

$$\mathsf{Fix}\,(\equiv)\,(\lhd)\,F \quad \triangleq \quad \epsilon\,x.\,[\,\mathsf{greatest}\;(\lhd)\;(\mathsf{fixed\_point\_modulo}\;(\equiv)\;F)\;x\,]$$

Appropriate instantiations of the binary relations $\equiv$ and $\lhd$ produce a combinator for unique fixed point and a combinator for optimal fixed point (possibly modulo an equivalence relation).

Now, in order to exploit properties about the value returned by $\mathsf{Fix}\,(\equiv)\,(\lhd)\,F$, we need to prove that the functional F indeed admits a greatest fixed point. For a very large scope of circular definitions, the existence of greatest fixed points can be derived from one very general theorem, which is developed in this paper. This theorem combines and generalizes several existing ideas: *contraction conditions* [12], *inductive invariants* [15] and *complete ordered families of equivalence* [18, 10]. Moreover, the corollaries used in the particular case of partial functions rely on the theory of *optimal fixed points* [17] and involves a generalized version of a theorem developed in the context of *maximal inductive fixed points* [15, 14].

The paper is organized as follows. First, we present all the ingredients that our paper builds upon. Second, we describe our generic fixed point combinator and its specialized versions. We then present our fixed point theorem and its corollaries. Finally, we investigate, without formal justification, the possibility for code extraction from circular definitions based on the combinator Fix. Due to space limitations, several results can only be summarized. The details can be found in the long version of this paper [8].

2 Ingredients

2.1 Contraction conditions for recursive functions

Harrison [12] used *contraction conditions* in order to show the existence of a unique fixed point for functionals describing recursive functions.

Definition 1 (Contraction condition for recursive functions). *Let F be a functional of type $(A \to B) \to (A \to B)$, and $<$ be a well-founded relation on values of type A. The* contraction condition *for F with respect to $<$ states:*

$$\forall x\, f_1\, f_2.\ (\forall y < x.\ f_1\, y = f_2\, y) \Rightarrow F\, f_1\, x = F\, f_2\, x$$

This contraction condition ensures the existence of a unique fixed point for F as soon as the codomain of the recursive function, the type B, is inhabited.

To understand why the contraction condition holds for a (simple) terminating recursive function, consider the following functional F, which describes a function that computes the binary logarithm of its argument: $F\, f\, x \triangleq \mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + f\, \lfloor \frac{x}{2} \rfloor$. Let us prove that this functional is contractive. Given arbitrary x, f_1 and f_2 and the assumption "$\forall y < x.\ f_1\, y = f_2\, y$", the proof obligation is:

$$\mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + f_1\, \lfloor \frac{x}{2} \rfloor \quad = \quad \mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + f_2\, \lfloor \frac{x}{2} \rfloor$$

If x is less or equal to 1, then the goal is trivial. Otherwise, we need to show that $f_1\, \lfloor \frac{x}{2} \rfloor$ and $f_2\, \lfloor \frac{x}{2} \rfloor$ are equal. The only way to prove this equality is to use the assumption "$\forall y < x.\ f_1\, y = f_2\, y$". So, we have to justify the fact that $\lfloor \frac{x}{2} \rfloor$ is less than x, which is true because x is greater than one. The inequation $\lfloor \frac{x}{2} \rfloor < x$ indeed captures the fact that the recursive call is made on a value smaller than the current argument x.

Contraction conditions support reasoning on higher-order recursion. They can also be adapted to n-ary recursive functions and mutually-recursive functions, which can be encoded into simple functions using products and sums, respectively. The details of the encoding can be hidden through appropriate reformulations of the contraction condition and of the fixed point theorem.

Moreover, contraction conditions can be easily extended so as to support partial functions by restricting arguments to be in a given domain D. For a functional F contractive on a domain D, the fixed point theorem guarantees the existence of a partial fixed point f on that domain, satifying$\forall x.\, D\, x \Rightarrow f\, x = F\, f\, x$. Notice that the use of this theorem requires one to provide the domain D *before* constructing the fixed point f of F.

2.2 Inductive invariants

As Krstić and Matthews [15] point out, the contraction condition for recursive function fails to handle the case of nested recursion. Consider the nested zero function, described by the functional $F\, f\, x \triangleq \mathsf{if}\ x = 0\ \mathsf{then}\ 0\ \mathsf{else}\ f(f(x-1))$.

Trying to prove F contractive leads to the proof obligation $f_1(f_1(x-1)) = f_2(f_2(x-1))$. The hypothesis of the contraction condition can be used to prove $f_1(x-1)$ equal to $f_2(x-1)$, because $x-1$ is smaller than x. However, we have no assumption at all on the value of $f_1(x-1)$, so we cannot prove the equality $f_1(f_1(x-1)) = f_2(f_1(x-1))$.

To address this limitation, Krstić and Matthews [15] introduced the notion of *inductive invariants* and used it to weaken the contraction condition, thereby obtaining a stronger fixed point theorem able to handle nested recursion.

Definition 2 (Inductive invariants). *A binary relation S of type $A \to B \to$ Prop is an* inductive invariant *for a functional F of type $(A \to B) \to (A \to B)$ if there exists a well-founded relation $<$ such that*

$$\forall x\, f.\ (\forall y < x.\ S\, y\, (f\, y)) \Rightarrow S\, x\, (F\, f\, x)$$

The first observation to be made is that if S is an inductive invariant for F, then any fixed point f of F admits S as post-condition, in the sense that $S\, x\, (f\, x)$ holds for any x. Formally, the *restricted contraction condition* for a functional F, with respect to an inductive invariant S, is similar to the contraction condition except that it includes an extra hypothesis about the function f_1. This condition guarantees the existence and uniqueness of a fixed point.

Definition 3 (Restricted contraction condition for recursive functions).

$$\forall x\, f_1\, f_2.\ (\forall y < x.\ f_1\, y = f_2\, y) \wedge (\forall y.\ S\, y\, (f_1\, y)) \Rightarrow F\, f_1\, x = F\, f_2\, x$$

By instantiating S as the predicate "$\lambda x\, r.\ (r = 0)$", one can prove that the nested zero function admits a unique fixed point and always returns zero.

2.3 Complete ordered families of equivalences

The contraction conditions described so far can only deal with recursion, for the basic reason that recursive calls must be applied to smaller values with respect to a well-founded relation. In order to deal with corecursive functions, Matthews [18] introduced a different form of contraction conditions stated in terms of *families of converging equivalence relations*. Di Gianantonio and Miculan [10] slightly simplified this structure, calling it *complete ordered families of equivalence*, abbreviated as "c.o.f.e.". We follow their presentation.

The contraction condition for a functional F of type $(A \to A) \to A$ is stated in terms of a family of equivalence relations over values of type A, written $\overset{i}{\approx}$, indexed with values of an ordered type I. This family needs to be *complete* in the sense that all *coherent sequences* converge to some limit. Note: the definitions of coherence and of completeness can be skipped upon first reading.

Definition 4 (Ordered families of equivalence). *The structure $(A, I, \prec, \overset{i}{\approx})$ is an* ordered family of equivalences *when $\prec$ is a well-founded transitive relation over the type I and $\overset{i}{\approx}$ is an equivalence relation over the type A for any i of type I. Thereafter, the intersection of all the relations $\overset{i}{\approx}$ is written $\approx$.*

Definition 5 (Coherent sequences). *A sequence u_i of values of type A indexed by elements of type I is said to be* coherent *if for any indices i and j such that $i \prec j$ the values u_i and u_j are equivalent at level i, that is, $u_i \overset{i}{\approx} u_j$. More generally, the sequence u_i is said to be* coherent on the domain K*, for a predicate K of type $I \to \mathsf{Prop}$, when the property $u_i \overset{i}{\approx} u_j$ holds for any i and j satisfying K and such that $i \prec j$ holds.*

Definition 6 (Completeness for an ordered family of equivalences). *An ordered family of equivalences $(A, I, \prec, \overset{i}{\approx})$ is said to be* complete *if, for any downward-closed domain K (i.e., such that $i \prec j$ and $K\,j$ imply $K\,i$) and for any sequence u_i coherent on the domain K, the sequence u_i admits a limit l on the domain K, in the sense that $u_i \overset{i}{\approx} l$ holds for any i satisfying K.*

A basic example of c.o.f.e. is the one associated with streams. In this case, I is the set of natural numbers ordered with $<$. The relation $\overset{i}{\approx}$ relates any two streams that agree up to their i-th element. The intersection $\approx$ of the family of relations $(\overset{i}{\approx})_{i \in \mathbb{N}}$ corresponds to stream bisimilarity. This construction of a c.o.f.e. for sterams can be easily generalized to coinductive trees.

Complete ordered families of equivalences are used to state the following sufficient condition for the existence of a unique fixed point for F modulo $\approx$.

Definition 7 (Contraction condition for c.o.f.e.'s). *The functional F is* contractive *w.r.t. a complete ordered family of equivalences $(A, I, \prec, \overset{i}{\approx})$ when*

$$\forall x\, y\, i.\ (\forall j \prec i.\ x \overset{j}{\approx} y) \Rightarrow F\,x \overset{i}{\approx} F\,y$$

In the particular case of streams, the contraction condition expresses the fact that if x and y are two streams that agree up to the index $i-1$, then $F\,x$ and $F\,y$ agree up to the index i. More generally, the contraction condition asserts that, given any two values x and y, the functional F is such that $F\,x$ and $F\,y$ are always closer to one another than x and y are, for an appropriate distance.

Di Gianantonio and Miculan [11] have described a general theory, expressed in categories of sheaves, in which complete ordered families of equivalences are simply particular cases of sheaves on well-founded topologies. Their theory also covers the case of well-founded recursion, described by functionals of type $\forall x : A.\, (\{y \mid y < x\} \to B) \to B$. However, di Gianantonio and Miculan do not cover the important case of nested calls, nor do they explain how the contraction condition for recursive functions described by functionals of type $(A \to B) \to (A \to B)$ fits their model.

2.4 Optimal fixed point

The combinator Fix_1 for unique fixed points [18] described in the introduction does not work for partial functions because the associated fixed point equation typically admits several partial fixed points. One idea, put forward by Krstić

and Matthews [15] and investigated in more details by Krstić in [14], is that there is always a "best" domain for any functional describing a terminating recursive function, and that, on this domain, there exists a unique fixed point. The formalization of this idea relies on the notion of *inductive fixed point*.

Definition 8 (Inductive fixed point). *f is an* inductive fixed point *of a functional F on a domain D if there exists a well-founded relation $<$ such that:*

$$\forall g\, x.\ D\, x \Rightarrow (\forall y < x.\ D\, y \Rightarrow f\, y = g\, y) \Rightarrow f\, x = F\, g\, x$$

Interestingly, an inductive fixed point on a given domain is always the unique fixed point on that domain. Moreover, any functional admits a *maximal* inductive fixed point, which is the inductive fixed point with the largest domain. This theorem, which does not appear to have ever been mechanized, may suggest the definition of a *maximal inductive fixed point combinator*. Such a combinator would be useful for terminating functions. However, it would not accommodate corecursive functions.

In this paper, we invoke an older and much more general theorem in order to formalize the notion of "best" fixed point. The theorem, due to Manna and Shamir [17], asserts the existence of an *optimal fixed point* for any functional describing a partial function. While it was initially designed for recursive programs, the theorem turns out to apply to a much larger class of circular definitions.

Several definitions need to be introduced before we can state this theorem. A *partial function* $\bar{f}$, written with an overline, is represented as a pair (f, D) of a total function f of type $A \to B$ and of a domain D of type $A \to \mathsf{Prop}$. We write $A \hookrightarrow B$ the type of partial functions from A to B. Moreover, we write $\mathrm{dom}(\bar{f})$ the right projection of $\bar{f}$ and write f (without an overline) the left projection of $\bar{f}$. Two partial functions $\bar{f}$ and $\bar{f}'$ are said to be *equivalent*, written $\bar{f} \overset{\hookrightarrow}{=} \bar{f}'$, if they have the same domain and are extensionally equal on that domain. Moreover, two partial functions $\bar{f}$ and $\bar{f}'$ are said to be *consistent* if they agree on the intersection of their domains. Finally, a partial function $\bar{f}'$ *extends* a partial function $\bar{f}$, written $\bar{f} \sqsubseteq \bar{f}'$, if the domain of $\bar{f}$ is included in the domain of $\bar{f}'$ and if f and f' are extensionally equal on the domain of $\bar{f}$. Note that the relation $\sqsubseteq$ defines a partial order (modulo $\overset{\hookrightarrow}{=}$) on the set of partial functions. The next two definitions formalize the notion of optimal fixed point.

Definition 9 (Generally-consistent fixed points). *Let $\bar{f}$ be a fixed point modulo $\overset{\hookrightarrow}{=}$ (the equivalence between partial functions) of a functional F of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$. The fixed point $\bar{f}$ is said to be a* generally consistent*, written* $\mathsf{generally\_consistent}\, F\, \bar{f}$*, if any other fixed point $\bar{f}'$ of F modulo $\overset{\hookrightarrow}{=}$ is consistent with $\bar{f}$.*

In other words, a generally-consistent fixed point $\bar{f}$ of a functional F is such that, for any other fixed point $\bar{f}'$ of F, the equation $f'(x) = f(x)$ holds for any x that belongs both to the domain of $\bar{f}$ and that of $\bar{f}'$. The contrapositive of this statement asserts that the domain of a generally-consistent fixed point cannot include any point x whose image is not uniquely determined by the fixed point

equation for F. Thus, as argued by Manna and Shamir [17], generally-consistent fixed points are the only genuine solutions of any circular function definition.

Definition 10 (Optimal fixed point). *A partial function $\bar{f}$ of type $A \hookrightarrow B$ is the* optimal fixed point *of a functional F of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$ if it is the greatest generally-consistent fixed point of F, with respect to the partial order $\sqsubseteq$ on the set of partial functions.*

In short, the optimal fixed point $\bar{f}$ of a functional F is the generally-consistent fixed point of F with the largest domain. This means that every other generally-consistent fixed point of F is a restriction of $\bar{f}$ to a smaller domain.

Theorem 1 (Optimal fixed point theorem). *For any functional F of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$, where B is inhabited, F admits an optimal fixed point.*

The optimal fixed point theorem appears to have had relatively little impact as a theory of circular *program* definitions, probably because optimal fixed points are not computable in general. Yet, as a foundation for a theory of circular *logical* definitions, we find the optimal fixed point theorem to be the tool of choice.

2.5 Contributions of this paper

1. By spotting the interest of optimal fixed points for logical circular definitions and by conducting the first formal development of the optimal fixed point theorem, we obtain a proper treatment of partiality for recursive and corecursive functions in higher-order type theory.
2. Using invariants to generalize existing results on complete ordered families of equivalences, we provide the first general method for justifying the well-definiteness of nested corecursive functions. The use of invariants also supports reasoning on certain forms of corecursive values that could not be formalized with previously-existing contraction conditions.
3. By showing that contraction conditions for recursive functions can be obtained as a particular instance of contraction conditions for complete ordered families of equivalences, even when nested calls are involved, we are able to offer a unified presentation of a number of fixed point theorems based on contraction conditions.

3 The greatest fixed point combinator

3.1 Definition of the greatest fixed point combinator

The combinator Fix takes as argument an equivalence relation $\equiv$ and a partial order $\lhd$, both defined on values of an inhabited type A. It then takes a functional F of type $A \to A$ and returns the greatest fixed point of F modulo $\equiv$ with respect to $\lhd$, if it exists. Its definition relies on the predicate "greatest $\prec$ $P\,x$", which asserts that x satisfies P and that x is greater than any other value satisfying P, with respect to $\prec$.

Definition 11 (The greatest fixed point combinator).

$$\mathit{Fix}\,(\equiv)\,(\lhd)\,F \quad \triangleq \quad \epsilon\,x.\,[\,\mathit{greatest}\,(\lhd)\,(\mathit{fixed\_point\_modulo}\,(\equiv)\,F)\,x\,]$$

The application of the epsilon operator requires a proof that the type A is inhabited. We encapsulate this proof using an inductive data type Inhabited, of sort Type → Prop. (Note that proofs of type Inhabited A need not be manipulated explicitly, thanks to the use of Coq's typeclass facility.) Thus, Fix has type:

$$\forall A.\,(\mathsf{Inhabited}\,A) \to (A \to A \to \mathsf{Prop}) \to (A \to A \to \mathsf{Prop}) \to (A \to A) \to A$$

3.2 Instantiation as a unique fixed point combinator

The unique fixed point combinator Fix_1, useful for circular definitions that do not involve partial functions, can be defined in terms of Fix. To that end, it suffices to instantiate both $\equiv$ and $\lhd$ as the equality between values of type A.

Definition 12 (Another unique fixed point combinator).

$$\mathit{FixVal}\,F \quad \triangleq \quad \mathit{Fix}\,(=)\,(=)\,F$$

FixVal is provably equivalent to the definition $\epsilon\,x.(\forall y.\ y = F\,y \iff y = x)$.

More generally, we can construct a combinator for unique fixed point modulo an equivalence relation $\sim$, simply by instantiating both $\equiv$ and $\lhd$ as $\sim$.

Definition 13 (Combinator for unique fixed point modulo).

$$\mathit{FixValMod}\,(\sim)\,F \quad \triangleq \quad \mathit{Fix}\,(\sim)\,(\sim)\,F$$

3.3 Instantiation as an optimal fixed point combinator

We now construct a combinator that returns the optimal fixed point of a functional F of type $(A \to B) \to (A \to B)$. First, we need to transform F as a functional between partial functions, of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$, so as to be able to invoke the theory of optimal fixed points. Second, we need to find a suitable instantiation of the relation $\lhd$ to ensure that the greatest fixed point with respect to $\lhd$ is exactly the optimal fixed point. We start with the first task.

Definition 14 ("Partialization" of a functional). *A functional F of type $(A \to B) \to (A \to B)$ can be viewed as a functional of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$, i.e. as a functional on partial functions, by applying the following "partialization" operator: $\mathsf{partialize}\,F \triangleq \lambda(f, D).\,(F\,f, D)$.*

Definition 15 (Partial fixed points). *Given a functional F of type $(A \to B) \to (A \to B)$, we say that $\bar{f}$ is a* partial fixed point *of F if and only if it is a fixed point of the functional "$\mathsf{partialize}\,F$" modulo $\overset{\hookrightarrow}{=}$.*

Our next step is to define a relation $\ll_F$ over the set of fixed points of "$\mathsf{partialize}\,F$" so that the greatest element of $\ll_F$ is exactly the optimal fixed point of F. On the one hand, the optimal fixed point is a generally-consistent fixed point of "$\mathsf{partialize}\,F$", moreover it is the greatest with respect to $\sqsubseteq$. On the other hand, the combinator Fix produces a fixed point $\bar{f}$ of "$\mathsf{partialize}\,F$" which is the greatest with respect to the relation $\ll_F$, meaning that any other fixed point $\bar{f}'$ satisfies $\bar{f}' \ll_F \bar{f}$. To ensure that $\bar{f}$ is the optimal fixed point, we need to ensure (1) that $\bar{f}$ is generally consistent, meaning that it is consistent with any other fixed point and (2) that $\bar{f}$ extends any other generally-consistent fixed point. These two requirements give birth to the following definition of $\ll_F$.

Definition 16 (Partial order selecting the optimal fixed point).

$$\bar{f}' \ll_F \bar{f} \quad \triangleq \quad \mathit{consistent}\,\bar{f}\,\bar{f}' \wedge (\mathit{generally\_consistent}\,F\,\bar{f}' \Rightarrow \bar{f}' \sqsubseteq \bar{f})$$

Given a functional F of type $(A \to B) \to (A \to B)$, the value returned by "$\mathsf{Fix}\,(\overset{\hookrightarrow}{=})\,(\ll_F)\,(\mathsf{partialize}\,F)$" is a function of type $A \hookrightarrow B$. Since we are not interested in the domain of the resulting function but only in its support, of type $A \to B$, we retain only the first projection.

Definition 17 (The optimal fixed point combinator).

$$\mathit{FixFun}\,F \quad \triangleq \quad \pi_1(\,\mathit{Fix}\,(\overset{\hookrightarrow}{=})\,(\ll_F)\,(\mathit{partialize}\,F)\,)$$

The following theorem relates the definition of FixFun with that of the optimal fixed point, thereby justifying that FixFun indeed picks an optimal fixed point.

Theorem 2 (Correctness of the optimal fixed point combinator). *Given a functional F of type $(A \to B) \to (A \to B)$ and a partial function $\bar{f}$ of type $A \hookrightarrow B$, the following two propositions are equivalent:*

1. $\mathit{greatest}\,(\sqsubseteq)\,(\mathit{generally\_consistent}\,F)\,\bar{f}$
2. $\mathit{greatest}\,(\ll_F)\,(\mathit{fixed\_point\_modulo}\,(\overset{\hookrightarrow}{=})\,(\mathit{partialize}\,F))\,\bar{f}$

This ends our construction of the optimal fixed point combinator. The construction can be easily generalized to the case where values from the codomain B are compared with respect to an arbitrary equivalence relation $\equiv$ rather than with respect to Leibniz' equality. This construction results in a strictly more general combinator, called $\mathsf{FixFunMod}$, which is parameterized by the relation $\equiv$.

4 The general fixed point theorem and its corollaries

4.1 A general contraction theorem for c.o.f.e.'s

Our fixed point theorem for c.o.f.e.'s strengthens the result obtained in [18] and later refined in [10], adding, in particular, support for nested calls. Our contraction condition generalizes the contraction condition for c.o.f.e.'s with an invariant, in a somewhat similar way as in the restricted contraction condition.

Definition 18 (Contraction condition). *Given a c.o.f.e.* $(A, I, \prec, \overset{i}{\approx})$*, a functional* F *of type* $A \to A$ *is said to be contractive with respect to an invariant* Q *of type* $I \to A \to \mathsf{Prop}$ *when*

$$\forall x\, y\, i.\ (\forall j \prec i.\ x \overset{j}{\approx} y \wedge Q\, j\, x \wedge Q\, j\, y) \Rightarrow F\, x \overset{i}{\approx} F\, y \wedge Q\, i\, (F\, x)$$

Our fixed point theorem asserts that a given functional admits a unique fixed point as soon as it is contractive with respect to a *continuous* invariant. The notion of continuity that we introduce for this purpose is defined as follows.

Definition 19 (Continuity of an invariant). *Given a c.o.f.e.* $(A, I, \prec, \overset{i}{\approx})$*, an invariant* Q *is said to be* continuous *if the following implication holds for any downward-closed domain* K*, for any sequence* $(u_i)_{i:I}$ *and for any limit* l*.*

$$(\forall i.\, K\, i \Rightarrow u_i \overset{i}{\approx} l) \ \wedge\ (\forall i.\, K\, i \Rightarrow Q\, i\, (u_i)) \ \Rightarrow\ (\forall i.\, K\, i \Rightarrow Q\, i\, l)$$

Theorem 3 (Fixed point theorem for c.o.f.e.'s). *If* $(A, I, \prec, \overset{i}{\approx})$ *is a c.o.f.e. and if* F *is a functional of type* $A \to A$ *contractive with respect to a continuous invariant* Q *in this c.o.f.e., then* F *admits a unique fixed point* x *modulo* $\approx$*. Moreover, this fixed point* x *is such that the invariant* $Q\, i\, x$ *holds for any* i*.*

The proof of this theorem is fairly involved. The fixed point is constructed as a limit of a sequence, defined by well-founded induction on $\prec$. Each element of this sequence is itself defined in terms of a limit of the previous elements in the sequence. Moreover, the convergence of all those limits depend on the fact that the i-th value of the sequence satisfies the invariant at level i, that is, the predicate $Q\, i$. The details of the proof are described in the long version [8].

4.2 Fixed point theorem for corecursive values

When F is a contractive functional modulo $\approx$, it admits a unique fixed point modulo $\approx$ (by Theorem 3), thus "$\mathsf{FixValMod}\,(\approx)\,F$" satisfies the fixed point equation for F.

Theorem 4 (Fixed point theorem for FixValMod).

$$\begin{cases} x = \mathit{FixValMod}\,(\equiv)\,F \\ (A, I, \prec, \overset{i}{\approx})\ \mathit{is\ a\ c.o.f.e.} \\ \equiv\ \mathit{is\ equal\ to}\ \bigcap_{i:I} \overset{i}{\approx} \\ F\ \mathit{is\ contractive\ w.r.t.}\ Q \\ Q\ \mathit{is\ continuous} \end{cases} \Rightarrow \quad \begin{cases} x \equiv F\, x \\ \forall i.\, Q\, i\, x \end{cases}$$

Compared with previous work, the use of an invariant in the contraction condition makes it strictly more expressive. For the sake of presentation, we consider a simple example. The circular definition associated with the functional $F\, s \triangleq 1 :: (\mathsf{filter}\,(\geq a)\, s)$ is correct only if $a \leq 1$. When this is the case, we can prove F contractive. It suffices to define the invariant Q in such a way that "$Q\, i\, s$" implies that the i first elements of s are greater than or equal to a.

4.3 Fixed point theorem for recursive functions

The goal of this section is to build a c.o.f.e. that can be used to prove that a functional F of type $(A \to B) \to (A \to B)$ describing a terminating recursive function on a domain D admits a unique fixed point of type $A \to B$. This relatively simple construction, which allows to unify the various forms of contraction conditions, does not seem to have appeared previously in the literature.

Theorem 5 (c.o.f.e. for recursive functions). *Let $\equiv$ be an equivalence relation of type $A \to A \to \mathsf{Prop}$, let $<$ be a well-founded relation of type $A \to A \to \mathsf{Prop}$, and let D be a domain of type $A \to \mathsf{Prop}$. Then, the structure $(A \to B, A, <^+, \overset{x}{\approx})$ is a complete ordered family of equivalences, where $(\overset{x}{\approx})_{x:A}$ is a family of equivalence relations on values of type $A \to B$ defined as follows:*

$$f_1 \overset{x}{\approx} f_2 \quad \triangleq \quad \forall y <^* x.\, D\, y \;\Rightarrow\; f_1\, y \equiv f_2\, y$$

Above, $<^+$ is the transitive closure of $<$ and $<^*$ its reflexive-transitive closure.

In this particular c.o.f.e., the contraction condition can be reformulated in a way which, in practice, is equivalent to the conjunction of the propositions "S is an inductive invariant for F" and "F satisfies the restricted contraction condition with respect to S" (Definition 2 and Definition 3).

Theorem 6 (Contraction condition for recursive functions). *Let D be a domain of type $A \to \mathsf{Prop}$ and let S be a post-condition of type $A \to B \to \mathsf{Prop}$ compatible with $\equiv$, in the sense that if "$S\, x\, y_1$" holds and if "$y_1 \equiv y_2$" then "$S\, x\, y_2$" also holds. Then, in the c.o.f.e. for recursive functions, a functional F is contractive w.r.t. the invariant "$\lambda x\, f.\, D\, x \Rightarrow S\, x\, (f\, x)$" as soon as F satisfies*

$$\begin{aligned}\forall x\, f_1\, f_2.\, D\, x \;\wedge\; (\forall y < x.\, D\, y \;\Rightarrow\; f_1\, y \equiv f_2\, y \;\wedge\; S\, y\, (f_1\, y))\\ \Rightarrow\; F\, f_1\, x \equiv F\, f_2\, x \;\wedge\; S\, x\, (F\, f_1\, x)\end{aligned}$$

A corollary, not shown here, to the general fixed point theorem (Theorem 3) can be stated for this reformulated contraction condition. The conclusion of this corrolary asserts the existence of a partial fixed point f modulo $\equiv$ on the domain D. Moreover, this fixed point f satisfies the post-condition $\forall x.\, D\, x \Rightarrow S\, x\, (f\, x)$.

The next key theorem in our development establishes that the partial fixed point (f, D) is a *generally-consistent* fixed point of the functional "$\mathsf{partialize}\, F$". The proof of this theorem is quite technical. It reuses and generalizes several ideas coming from the proof that inductive fixed points are generally-consistent [14].

Combining the existence of a generally-consistent fixed point f for F on the domain D with the existence of an optimal fixed point for F, we deduce that the domain of the optimal fixed point of F contains D. It follows that the optimal fixed point for F satisfies the fixed point equation on the domain D.

Theorem 7 (Specification of FixFunMod for recursive functions).

$$\begin{cases} f = \mathsf{FixFunMod}\,(\equiv)\, F \\ \equiv \text{ is an equivalence} \\ < \text{ is well-founded} \\ S \text{ is compatible with } \equiv \\ F \text{ is contractive on } D \text{ w.r.t. } < \text{ and } S \text{ modulo } \equiv \end{cases} \Rightarrow \begin{cases} \forall x.\, D\, x \;\Rightarrow\; f\, x \equiv F\, f\, x \\ \forall x.\, D\, x \;\Rightarrow\; S\, x\, (f\, x) \end{cases}$$

4.4 Fixed point theorem for mixed recursive-corecursive functions

Due to space limitations, we skip the description of the c.o.f.e. for simple corecursive functions and directly focus on the strictly more general c.o.f.e. for functions mixing recursion and corecursion. Compared with the construction proposed by Matthews [18], we have added support for partial functions and for nested calls.

Let $A \to B$ be the type of the function to be constructed and let D be the domain on which we want to prove the function well-defined. The values of the input type A are compared with respect to some well-founded relation, written $<$. The values of the coinductive output type B are compared using an existing c.o.f.e. $(B, I, \prec, \overset{i}{\approx})$. The following result explains how to combine $<$ and $\prec$ in order to construct a c.o.f.e. for the function space $A \to B$.

Theorem 8 (c.o.f.e. for mixed recursive-corecursive functions). *The structure* $(A \to B, I \times A, <', \overset{(i,x)}{\approx'})$ *is a c.o.f.e., where* $<'$ *is the lexicographical order associated with the pair of relations* $(\prec, <^+)$ *and where* $(\overset{(i,x)}{\approx'})_{(i,x):I\times A}$ *is a family of equivalence relations on values of type* $A \to B$ *such that*

$$f_1 \overset{(i,x)}{\approx'} f_2 \quad \triangleq \quad \forall (j,y) \leq' (i,x).\ D\,y \Rightarrow f_1\,y \overset{j}{\approx} f_2\,y$$

The associated contraction condition and the fixed point theorem follow.

Theorem 9 (Contraction condition for corecursive functions). *Let* D *be a domain of type* $A \to \mathsf{Prop}$. *Let* S *be an indexed post-condition of type* $I \to A \to B \to \mathsf{Prop}$, *compatible with* $\overset{i}{\approx}$ *in the sense that if "*$S\,i\,x\,y_1$*" holds and if "*$\forall j \prec i.\, y_1 \overset{j}{\approx} y_2$*" holds then "*$S\,i\,x\,y_2$*" holds. Then, in the c.o.f.e. for mixed recursive-corecursive functions, a functional* F *is contractive w.r.t. the invariant "*$\lambda(i,x)\,f.\ D\,x \Rightarrow S\,i\,x\,(f\,x)$*" as soon as* F *satisfies the condition:*

$$\begin{aligned}\forall i\,x\,f_1\,f_2.\ D\,x \wedge (\forall (j,y) <' (i,x).\ D\,y \Rightarrow f_1\,y \overset{j}{\approx} f_2\,y \wedge S\,j\,y\,(f_1\,y) \wedge S\,j\,y\,(f_2\,y))\\ \Rightarrow F\,f_1\,x \overset{i}{\approx} F\,f_2\,x \wedge S\,i\,x\,(F\,f_1\,x)\end{aligned}$$

Theorem 10 (Specification of FixFunMod for mixed functions).

$$\begin{cases} f = \mathsf{FixFunMod}\,(\equiv)\,F \\ < \text{ is a well-founded relation} \\ (B, I, \prec, \overset{i}{\approx}) \text{ is a c.o.f.e.} \\ \equiv \text{ is equal to } \bigcap_{i:I} \overset{i}{\approx} \\ F \text{ is contractive on } D \text{ w.r.t. } S \\ S \text{ is compatible with } \overset{i}{\approx} \end{cases} \Rightarrow \begin{cases} \forall x.\ D\,x \Rightarrow f\,x \equiv F\,f\,x \\ \forall i\,x.\ D\,x \Rightarrow S\,i\,x\,(f\,x) \end{cases}$$

Let us apply this theorem to the filter function. Let f be the function $\mathsf{FixFunMod}\,(\equiv)\,F$, where F is the functional defined in §1.1 and $\equiv$ stands for

stream bisimilarity. The domain D characterizes streams that contain infinitely many elements satisfying the predicate P. Two streams from the domain are compared as follows: $s < s'$ holds if the index of the first element satisfying P in s is less than the index of the first element satisfying P in s'. No invariant is needed here, so we define S such that $S\,i\,s\,s'$ always holds. Let us prove F contractive, as in [18]. Assume the argument s decomposes as $x :: s'$. There are two cases. If x satisfies P, then the goal is $x :: (f_1\,s') \overset{i}{\approx} x :: (f_2\,s')$. This fact is a consequence of the assumption $f_1\,s' \overset{i-1}{\approx} f_2\,s'$, which we can invoke because $(i-1, s')$ is lexicographically smaller than (i, s). If x does not satisfy P, the goal is $f_1\,s' \overset{i}{\approx} f_2\,s'$. This fact also follows from the hypothesis of the contraction condition, because (i, s') is lexicographically smaller than the pair (i, s). Note that this relation holds only because the argument s belongs to the domain D. In conclusion, the equation $f\,s \equiv F\,f\,s$ holds for any stream s in the domain D.

5 Code Extraction

Given a formal development carried out in higher-order logic, one can extract a purely functional program by retaining only the computational content and erasing all the proof-specific elements. The extracted code enjoys a partial correctness property with respect to the original development. Note that termination is usually not guaranteed: even a Caml program extracted from a Coq development can diverge [3, 8]. Our definition of Fix relies on Hilbert's epsilon operator, a non-constructive axiom that does not admit an executable counterpart. Nevertheless, it is still possible to translate the constant Fix into a native "let-rec" construct from the target programming language.

Our experiments suggest that this extraction leads to efficient and correct programs, with respect to partial correctness. However, a formal justification of our approach is not attempted in this paper. The theory of code extraction is already far from trivial (see, e.g. [16]) and there exists, as far as we know, no theory able to account for the correctness of code extraction in the presence of user-defined extraction for particular constants. Thus, we leave the proof of correctness as a challenge to code extraction experts, and simply explain how to set up the extraction process in practice.

In Haskell, where evaluation is lazy by default, the extraction of the constant Fix is very simple: it suffices to translate "Fix" into "$\lambda F.\ \mathsf{let}\, x = F\,x\,\mathsf{in}\,x$". This value has type "$\forall A.\,(A \to A) \to A$", which is appropriate given the type of Fix. The extraction towards OCaml code is slightly trickier: due to the explicit boxing of lazy values, we need to extract the combinator for corecursive values towards a different constant than that used to extract functions. See [8] for details.

6 Other related work

The most closely related work has already been covered throughout §2. In this section, we briefly mention other approaches to circular definitions. (A detailed list of papers dealing with recursive function definitions can be found in [13].)

The package TFL developed by Slind [20] supports the definition of total recursive functions for which a well-founded termination relation can be exhibited. Building on Slind's ideas, Krauss [13] developed the *function* package, which supports a very large class of partial recursive functions. It relies on the generation of an inductive definition that captures exactly the domain of the recursive function. Contrary to our work, this approach does not support code generation for partial functions (except tail-recursive ones) and does not support corecursion.

The technique of recursion on an ad-hoc predicate, which consists in defining a function by structural induction on an inductive predicate that describes its domain, was suggested by Dubois and Donzeau-Gouge [9] and developed by Bove and Capretta [7]. Later, Barthe et al. [2] used it in the implementation of a tool for Coq. Besides the fact that it relies heavily on programming with dependent types, one major limitation of this approach is that the treatment of nested recursion requires the logic to support inductive-recursive definitions, an advanced feature absent from many theorem provers.

Another possibility for defining terminating recursive functions is to work directly with a general recursion combinator [19], using dependently-typed functionals. Balaa and Bertot [1] proved a fixed point theorem in terms of a contraction condition for functions of type "$\forall x : A.\ (\forall y : A.\ R\,y\,x \Rightarrow B\,y) \Rightarrow B\,x$", where R is some well-founded relation. More recently, Sozeau [21] implemented facilities for manipulating subset types in Coq, including a fixed point combinator for functionals of type $\big(\forall x : A.\ \big(\forall y : \{y : A \mid R\,y\,x\}.\ B\,(\pi_1\,y)\big) \Rightarrow B\,x\big) \Rightarrow \forall x : A.\,(B\,x)$. This approach supports higher-order and nested recursion, but only if the inductive invariant of the function appears explicitly in its type.

As mentioned in the introduction, Bertot [4] has investigated the formalization of the filter function in constructive type theory. This work was later generalized to support more general forms of mixed recursive-corecursive functions [5]. More recently, Bertot and Komendantskaya [6] experimented reasoning about non-guarded corecursive definitions by exploiting the correspondence between streams and functions over natural numbers.

7 Future work

In the future, we would like to implement a generator for automatically deriving corollaries to the general fixed point theorem, covering each possible function arity and providing versions with and without domains and invariants. Proving such corollaries by hand on a per-need basis is generally manageable, but having a generator would certainly be much more convenient.

References

1. Antonia Balaa and Yves Bertot. Fix-point equations for well-founded recursion in type theory. In Mark Aagaard and John Harrison, editors, *TPHOLs*, volume 1869 of *LNCS*, pages 1–16, 2000.
2. Gilles Barthe, Julien Forest, David Pichardie, and Vlad Rusu. Defining and reasoning about recursive functions: A practical tool for the Coq proof assistant. In Masami Hagiya and Philip Wadler, editors, *FLOPS*, volume 3945 of *LNCS*, pages 114–129. Springer, 2006.

3. Gilles Barthe, Maria João Frade, E. Giménez, Luis Pinto, and Tarmo Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004.
4. Yves Bertot. Filters on coinductive streams, an application to eratosthenes' sieve. In Pawel Urzyczyn, editor, *TLCA*, volume 3461 of *LNCS*, pages 102–115. Springer, 2005.
5. Yves Bertot and Ekaterina Komendantskaya. Inductive and Coinductive Components of Corecursive Functions in Coq. In *Proceedings of CMCS'08*, volume 203 of *ENTCS*, pages 25 – 47, April 2008.
6. Yves Bertot and Ekaterina Komendantskaya. Inductive and coinductive components of corecursive functions in coq. *ENTCS*, 203(5):25–47, 2008.
7. Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In Richard J. Boulton and Paul B. Jackson, editors, *TPHOLs*, volume 2152 of *LNCS*, pages 121–135. Springer, 2001.
8. Arthur Charguéraud. Long version of the current paper, 2010. `http://arthur.chargueraud.org/research/2010/fix/`.
9. C. Dubois and V. Donzeau-Gouge. A step towards the mechanization of partial functions: domains as inductive predicates. In *CADE-15 Workshop on mechanization of partial functions*, 1998.
10. Pietro Di Gianantonio and Marino Miculan. A unifying approach to recursive and co-recursive definitions. In Herman Geuvers and Freek Wiedijk, editors, *Selected Papers from 2nd Int. Wksh. on Types for Proofs and Programs, Berg en Dal, The Netherlands, 24–28 Apr. 2002*, volume 2646 of *KBCS*, pages 148–161. Springer-Verlag, Berlin, 2003.
11. Pietro Di Gianantonio and Marino Miculan. Unifying recursive and co-recursive definitions in sheaf categories. In Igor Walukiewicz, editor, *FOSSACS*, volume 2987 of *LNCS*, pages 136–150. Springer, 2004.
12. John Harrison. Inductive definitions: Automation and application. In E. Thomas Schubert, Phillip J. Windley, and Jim Alves-Foss, editors, *TPHOLs*, volume 971 of *LNCS*, pages 200–213. Springer, 1995.
13. Alexander Krauss. Partial and nested recursive function definitions in higher-order logic. *Journal of Automated Reasoning*, December 2009. To appear.
14. Sava Krstić. Inductive fixpoints in higher order logic. February 2004.
15. Sava Krstić and John Matthews. Inductive invariants for nested recursion. In David A. Basin and Burkhart Wolff, editors, *TPHOLs*, volume 2758 of *LNCS*, pages 253–269. Springer, 2003.
16. Pierre Letouzey. Programmation fonctionnelle certifiée : L'extraction de programmes dans l'assistant Coq, June 01 2007.
17. Zohar Manna and Adi Shamir. The theoretical aspects of the optimal FixedPoint. *SIAM Journal on Computing*, 5(3):414–426, September 1976.
18. John Matthews. Recursive function definition over coinductive types. In Yves Bertot, Gilles Dowek, André Hirschowitz, C. Paulin, and Laurent Théry, editors, *TPHOLs*, volume 1690 of *LNCS*, pages 73–90. Springer, 1999.
19. Bengt Nordström. Terminating general recursion. *BIT*, 28(3):605–619, 1988.
20. Konrad Slind. *Reasoning about Terminating Functional Programs*. PhD thesis, Institut für Informatik, Technische Universität München, 1999.
21. Matthieu Sozeau. Subset coercions in Coq. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *LNCS*, pages 237–252. Springer, 2006.