Noname manuscript No.
(will be inserted by the editor)

The Optimal Fixed Point Combinator — Long Version

Arthur Charguéraud

Received: date / Accepted: date

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

Arthur Charguéraud
INRIA Rocquencourt
Domaine de Voluceau
Rocquencourt - B.P. 105
78153 Le Chesnay, France
E-mail: arthur.chargueraud@inria.fr

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 [21]. 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.

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

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 [21]. 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 [15] 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 [15]).

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 $a = F\, a$. If this fixed point equation admits a unique solution, then the functional F characterizes a unique circular value. If the equation does not admit any solution, F is of little interest. But what if the equation admits several solutions?

One typical case of fixed point equations that do not not admit a unique solution occurs with the circular definitions 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 §4.3 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 [19], 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 point, which is always unique, is the most well-defined solution that can be extracted from the fixed point equation without making arbitrary choices. Since the optimal fixed point captures the maximal amount of non-ambiguous information contained in the functional F, it can be naturally regarded as *the* meaning of the circular definition associated with F. Manna and Shamir [19] 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). If the unique fixed point does not exist, the result of Fix is simply unspecified.

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* [17] and *complete ordered families of equivalence* [21,10]. Moreover, the corollaries used in the particular case of partial functions rely on the theory of *optimal fixed points* [19] and involves a generalized version of a theorem developed in the context of *maximal inductive fixed points* [17,16].

The paper is organized as follows. First, we consider a series of examples in order to illustrate the difficulties involved when reasoning on fixed point equations. Those examples are used throughout the paper to illustrate various fixed point theorems. Second, we explain the notions of contraction conditions, inductive invariants, complete ordered families of equivalence, and optimal fixed points, which are the four key ingredients that this paper builds upon. Third, 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.

2 Examples of fixed point equations

We consider a series of examples illustrating particular difficulties that may arise when trying to prove that a given functional admits a unique fixed point. We start with recursive function definitions, and then consider corecursive definitions of values and functions. All the examples manipulate natural numbers unless stated otherwise. Note that curried n-ary functions and mutually-recursive definitions are not discussed here, as they can be easily encoded into equations of the form $a \equiv F\,a$ using tuples and sums.

▶ *The fixed point might not exists or might not be unique* Consider the two following functionals.

$$F_1\,f\,x \triangleq 1 + f\,x \qquad\qquad F_2\,f\,x \triangleq f\,x$$

The functional F_1 does not admit any total function as fixed point. Indeed, no function f can satisfy the fixed point equation $f\,x = 1 + f\,x$. On the contrary, the functional F_2 admits any total function as fixed point. Thus, it is not be possible to consider "the" fixed point of any of those two functions.

▶ *A partial function is usually not a unique fixed point* The following functional defines the division function between natural numbers.

$$F_3\,f\,(p,q) \triangleq \textsf{if } p \le q \textsf{ then } 0 \textsf{ else } 1 + f(p-q,q)$$

The functional F_3 admits as fixed point the function $(p,q) \mapsto \lfloor \frac{p}{q} \rfloor$ defined over the domain $\mathbb{N} \times \mathbb{N}^*$. This function is the unique fixed point on that domain, however there exist other fixed points on other domains. For instance, the function $(p,q) \mapsto p$ is a fixed point on the domain $\mathbb{N} \times \{1\}$. Moreover, any function is a fixed point on the empty domain.

One way to ensure the existence of a unique fixed point is to modify the original functional so that it admits a *total* function as fixed point, by testing whether the argument belong to the domain:

$$F_3'\,f\,(p,q) \triangleq \textsf{if } q = 0 \textsf{ then arbitrary else if } p \le q \textsf{ then } 0 \textsf{ else } 1 + f(p-q,q)$$

As explained in the introduction, this approach is not entirely satisfying.

Intuitively, there exists a "best" fixed point to the functional F_3, which is the function defined on the domain $\mathbb{N} \times \mathbb{N}^*$. Ideally, we would like to be able to characterize the "best" fixed point without providing its domain explicitly at the time of construction of the fixed point. The theory of optimal fixed points gives a formal meaning to the notion of "best" fixed point in the general case.

▶ *Termination is sufficient, but not necessary* Any functional of type $(A \to B) \to (A \to B)$ describing a recursive function that terminates on all inputs admits a unique fixed point. For example, the following functional defines the binary logarithm function:

$$F_4\, f\, x \triangleq \mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + f\,\lfloor\frac{x}{2}\rfloor$$

Interestingly, termination is *not* necessary to the existence of a unique fixed point. In other words, there exist recursive definitions whose execution does not terminate but which still admit a unique fixed point, as illustrated by the following two examples.

$$F_5\, f\, x \triangleq f\, x - f\, x \qquad\qquad F_6\, f\, x \triangleq f\,(x-1) \vee (x = 0)$$

The former admits as unique fixed point the constant function zero, while the latter, taken from [14], admits as unique fixed point the constant predicate True. Depending on one's motivation, it may or may not be interesting to accept these kind of well-defined yet non-termination circular definitions. Our approach does support them.

▶ *The functional may involve nested recursion* Nested recursion occurs when a recursive call is made on a result built from another recursive call. Nested recursion occurs naturally is the definition of normalization functions [**?**] and in the computation of most-general unifiers (see, e.g., [15]). For the sake of presentation, we consider the simpler example of the *nested zero* function:

$$F_7\, f\, x \triangleq \mathsf{if}\ x = 0\ \mathsf{then}\ 0\ \mathsf{else}\ f(f(x-1))$$

To reason on the functional F_7, one needs to prove by induction on x the property "f terminates and returns zero". Without such a precise property, there is no way to show that the outer recursive call to f is made on a value smaller than x. More generally, in order to handle nested recursion, one must be able to reason about particular properties of the function in the same time as establishing its termination.

▶ *The functional may involve higher-order recursion* Higher-order recursion occurs when the recursive function is given as argument to a higher-order function. Consider the example of a function that increments the values stored in the leaves of a finitely-branching tree:

$$F_8\, f\, x \triangleq \mathsf{match}\ x\ \mathsf{with}\ \mathsf{Leaf}\, n \mapsto \mathsf{Leaf}\,(n+1) \mid \mathsf{Node}\, l \mapsto \mathsf{Node}\,(\mathsf{List.map}\, f\, l)$$

With higher-order recursion, showing that recursive calls are made to smaller arguments can be arbitrarily complex, because it involves reasoning on the behaviour of the higher-order function being applied (in this case List.map).

▶ *Not all corecursive definitions are correct* The functional

$$F_9\, s \triangleq 0 :: (\mathsf{map}\,\mathsf{succ}\, s)$$

is a correct definition of a corecursive value in the sense that it admits a unique fixed point, which is the stream of natural numbers "0 :: 1 :: 2 :: 3 :: ..". More challenging is the justification of the well-definiteness of Hamming's sequence:

$$F_{10}\, s \triangleq 1 :: \mathsf{merge}\,(\mathsf{map}\,(\mathsf{mult}\,2)\, s)\,(\mathsf{map}\,(\mathsf{mult}\,3)\, s)$$

where merge is a function that takes two streams of natural numbers and produces a new stream by always picking the smallest element from the heads of the two given streams.

Not all corecursive definitions admit a unique fixed point, as illustrated by the next two definitions. The first one admits as fixed point any stream that starts with a 0. The second one does not admit any fixed point at all. Thus, neither of them is a correct circular definition.

$$F_{11}\, s \triangleq 0 :: (\mathsf{tail}\, s) \qquad\qquad F_{12}\, s \triangleq 0 :: (\mathsf{map}\,\mathsf{succ}\,(\mathsf{tail}\, s))$$

► *Corecursive functions are not always syntactically productive* If a functional is such that all the recursive calls of a corecursive function are guarded by constructors and only by constructors, then this functional admits a unique fixed point. For instance, the functional F_{13} satisfies this criteria. Its fixed point produces the stream of natural numbers greater than or equal to n.

$$F_{13}\, f\, n \triangleq n :: f(n+1)$$

However, not all corecursive functions are syntactically-productive in this way. For example, the following corecursive function produces the stream of prime numbers greater than a given value n, in increasing order. The difficulty with such a definition is that the existence of a unique fixed point relies on the fact that there exist infinitely many prime numbers.

$$F_{14}\, f\, n \triangleq \mathsf{if}\ (\mathsf{is\_prime}\, n)\ \mathsf{then}\ n :: f\,(n+1)\ \mathsf{else}\ f\,(n+1)$$

The generalization of this example is the filter function on infinite streams. Given a predicate P of type "$A \to \mathsf{bool}$" (or "$A \to \mathsf{Prop}$"), we define:

$$F_{15}\, f\, s \triangleq \mathsf{let}\ x :: s' = s\ \mathsf{in}\ \mathsf{if}\ (P\, s)\ \mathsf{then}\ x :: (f\, s')\ \mathsf{else}\ f\, s'$$

The intended fixed point of F_{15} is a partial function that accepts a stream containing infinitely many items satisfying the property P and returns a stream containing all such items. Circular definitions such as the two above are often said to be *mixed recursive-corecursive*, because they require several recursive calls before producing the head element of their output.

► *Corecursive functions may involve nested calls* Nested calls can also occur in functions producing corecursive values. For example, the following functional takes a stream s of the form "$s_0 :: s_1 :: ... :: s_n :: ...$" and returns the stream "$(2^{2^0} \cdot s_0) :: (2^{2^1} \cdot s_1) :: ... :: (2^{2^n} \cdot s_n) :: ...$".

$$F_{16}\, f\, s \triangleq \mathsf{let}\ x :: s' = s\ \mathsf{in}\ (2 \cdot x) :: f\,(f\, s')$$

While reasoning about a *recursive* function with nested calls requires the ability to specify results of the function, reasoning on a *corecursive* function with nested calls requires the ability to specify arbitrarily-long prefixes of its outputs. To the extent of our knowledge, there exists no published technique supporting formal reasoning about nested corecursion.

A similar need for specifying properties of a value in the same time as proving well-definiteness can occur in the definition of corecursive values, as illustrated by the following functional, which admits a unique fixed point only when the constant a is less than or equal to 1.

$$F_{17}\, s \triangleq 1 :: (\mathsf{filter}\,(\geq a)\, s)$$

▶ *Fixed points might not be computable* We end our series of example with two examples suggesting how exotic functionals can be. The following example is adapted from [16].

$$F_{18}\, f\, x \triangleq \mathsf{if}\ (\forall x.\, f\, x = 1)\ \mathsf{then}\ a\ \mathsf{else}\ x$$

If the constant a is equal to 1, then the functional admits exactly two fixed points: the constant function that always returns 1, and the identity function. Otherwise, the functional admits only the identity function as fixed point. The second example, adapted from a program exhibited in [19], is even tricker:

$$F_{19}\, f\, x \triangleq \mathsf{if}\ x \leq 1\ \mathsf{then}\ f\, 0 + f\, 1\ \mathsf{else}\ x$$

Suppose there exists a fixed point f of F_{19}. The fixed point equation applied to the argument 0 produces the equation $f\,0 = f\,0 + f\,1$, which implies that $f\,1$ must be equal to 0. Similarly, for the argument 1, we get $f\,1 = f\,0 + f\,1$, hence $f\,0$ must be equal to 0 as well. For any x greater than 1, we have $f\,x = x$. Thus, we deduce that the unique fixed point of F_{19} is the function "$x \mapsto \mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ x$".

The kind of equations needed to solve the fixed point equation suggests the potential complexity of such a resolution in general. In fact, the fixed point of a computable functional can be a non-computable function [19].

3 Ingredients

Our proposal builds upon four existing ingredients: *contraction conditions*, *complete ordered families of equivalences*, *inductive invariants*, and *optimal fixed points*. The matter of this section is to present those notions, as well as their associated fixed point theorems.

3.1 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_4, which describes a function that computes the binary logarithm of its argument: $F_4\, f\, x \triangleq \mathsf{if}\ x \leq 1\ \mathsf{then}\ 0\ \mathsf{else}\ 1 + f\, \lfloor \frac{x}{2} \rfloor$. 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:

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

These are two copies of the body of the functions, with the only difference that recursive calls are made with a function f_1 in the former and with a function f_2 in the latter. Those two functions are not specified, apart from the fact that the two are extensionally equal on arguments smaller than the current argument x. This hypothesis on f_1 and f_2 suffices to prove the contraction condition, as follows. 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.

The contraction condition for a given functional F ensures the existence of a unique fixed point for F, when the codomain of the recursive function, the type B, is inhabited. This fixed point can be defined by well-founded recursion on the relation $<$. The details of the implementation depend on the definition of well-foundedness, but, intuitively, the fixed point f of F is constructed by taking the fixed point of the following functional:

$$\lambda f\, x.\, F\, (\lambda y.\, \mathsf{if}\, y < x\, \mathsf{then}\, f\, y\, \mathsf{else\, arbitrary})\, x$$

This functional can indeed be applied to a well-founded recursion combinator because a recursive call to f on a value y is only made when y is actually smaller than x. We can easily verify that f is indeed a fixed point, i.e. that $f\, x$ is equal to $F\, f\, x$ for any x. By definition of f, it suffices to show that "$F\, (\lambda y.\, \mathsf{if}\, y < x\, \mathsf{then}\, f\, y\, \mathsf{else\, arbitrary})\, x$" is equal to $F\, f\, x$. Since F is contractive, it suffices to show that for any y smaller than x, the expression "$\mathsf{if}\, y < x\, \mathsf{then}\, f\, y\, \mathsf{else\, arbitrary}$" is equal to $f\, y$, which is true because $y < x$.

The contraction condition also guarantees the uniqueness of a fixed point for F. Indeed, let f_1 and f_2 be two fixed points. Then we can prove by well-founded induction on x that $f_1\, x = f_2\, x$. Since f_1 and f_2 satisfy the fixed point equation, the goal is equivalent to $F\, f_1\, x = F\, f_2\, x$. By the contraction condition, it suffices to check that, for any y smaller than x, $f_1\, y = f_2\, y$, which is exactly the induction hypothesis.

If the type B is not inhabited, then the contraction condition does not imply the existence of a unique fixed point. As counter-example, let F be an identity function of type $(\mathsf{nat} \to \mathsf{False}) \to (\mathsf{nat} \to \mathsf{False})$. Asserting the existence of a fixed point of type $\mathsf{nat} \to \mathsf{False}$ would clearly be unsound. Yet, F satisfies the contraction condition. Indeed, the function f_1 quantified at the head of this condition have type $\mathsf{nat} \to \mathsf{False}$, so applying f_1 to any natural number produces a proof of False, which can be used to prove the conclusion $F\, f_1\, x = F\, f_2\, x$. Thus, the hypothesis stating that the output type B must be inhabited is necessary.

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.

Definition 2 (Guarded contraction condition for recursive functions) The contraction condition for F guarded by a predicate D states:

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

Note that the hypothesis "$D\,y$" requires the user to justify that recursive calls are made to values that belong to the domain D. Here, the fixed point theorem states that a functional F contractive on a domain D admits a unique partial function f as fixed point on the domain D. This fixed point satisfies the guarded fixed point equation $\forall x.\, D\,x \Rightarrow f\,x = F\,f\,x$.

3.2 Inductive invariants

As Krstić and Matthews [17] 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

To address this limitation, Krstić and Matthews [17] 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 3 (Inductive invariants) A binary relation S of type $A \to B \to \mathsf{Prop}$ 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. Indeed, if f is fixed point, then the property $S\,x\,(f\,x)$ can be proved by well-founded induction on x. As f is a fixed point, it suffices to show $S\,x\,(F\,f\,x)$, which follows directly from the fact that S is an inductive invariant and from the induction hypothesis.

Krstić and Matthews [17] further observed that if S is an inductive invariant for F, then one can safely assume f_1 to admit S as post-condition while proving the contractivity of F. 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 4 (Restricted contraction condition for recursive functions) The contraction condition for F restricted to the inductive invariant S is:

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

3.3 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 [21] 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. The proof of existence of a unique fixed point for F requires the ordered families of equivalence to satisfy a notion of completeness. Intuitively, completeness asserts that all *coherent sequences* of values of type A admit a limit. The following definitions formalize the notion of complete ordered families of equivalences. Note: the definitions of coherence and of completeness can be skipped upon first reading.

Definition 5 (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 equivalence relation obtained as the intersection of all the relations $\overset{i}{\approx}$ is written $\approx$.

Definition 6 (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 7 (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. Completeness for this particular ordered family of equivalences asserts that given a sequence of streams $(s_i)_{i \in \mathbb{N}}$ such that for all j and for all i less than j the stream s_i agrees with the stream s_j up to index i, there exists a limit stream l such that for any i the stream s_i agrees with l up to index i. Intuitively, $(s_i)_{i \in \mathbb{N}}$ describes a sequence of streams where each stream refines the previous one by fixing one additional item, and l is the limit stream obtained by diagonalization, i.e. the i-th element of l is the i-th element of the i-th stream. 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 8 (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. Intuitively, this property captures the idea that the application of F is productive. 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.

3.4 Optimal fixed point

As explained in the introduction, Matthews [21] relied on the combinator Fix_1 to pick the unique fixed point of a functional, whenever such a fixed point exists.

Definition 9 (The unique fixed point combinator)

$$\mathsf{Fix}_1\,F \quad \triangleq \quad \epsilon\,x.\,(\forall y.\; y = F\,y \iff y = x)$$

Given a function F, one can always construct the term $\mathsf{Fix}_1\,F$. If the functional F is later shown to be contractive, then it can be deduced that $\mathsf{Fix}_1\,F$ is actually a fixed point for F, i.e. that the fixed point equation "$\mathsf{Fix}_1\,F = F\,(\mathsf{Fix}_1\,F)$" holds. Unfortunately, the combinator Fix_1 cannot be used for partial functions, because fixed point equation for partial functions generally admit several solutions.

One idea, put forward by Krstić and Matthews [17] and investigated in more details by Krstić in [16], 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 10 (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 [19], 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.

Before we can state the theorem formally, we need to introduce several definitions in order to formalize the definition and the properties of partial functions in a theory of total functions. Throughout the paper, we let $P \subseteq P'$ be an abbreviation for $\forall x.\, P\,x \Rightarrow P'\,x$ and let $f =_P f'$ be an abbreviation for $\forall x.\, P\,x \Rightarrow f\,x = f'\,x$.

Definition 11 (Representation of partial functions) A partial function $\bar{f}$ of type $A \hookrightarrow B$ is a pair (f, D) made of a total function f of type $A \to B$ and of a domain D represented as a predicate of type $A \to \mathsf{Prop}$. Throughout the rest of the paper, we use overlined symbols to denote partial functions and we write $\text{dom}(\bar{f})$ the right projection of $\bar{f}$ and write f (without an overline) the left projection of $\bar{f}$.

Definition 12 (Equivalence between partial functions) Two partial functions $\bar{f}$ and $\bar{f}'$ are *equivalent*, written $\bar{f} \overset{\hookrightarrow}{=} \bar{f}'$, if they have the same domain, i.e. $\text{dom}(\bar{f}) = \text{dom}(\bar{f}')$ and if they are extensionally equal on that domain, that is, $f =_{\text{dom}(\bar{f})} f'$.

Definition 13 (Consistency between partial functions) Two partial functions $\bar{f}$ and $\bar{f}'$ are *consistent*, written $\bar{f} \triangledown \bar{f}'$, if they agree on the intersection of their domain, that is, if $f =_P f'$ for $P = \text{dom}(\bar{f}) \cap \text{dom}(\bar{f}')$.

Definition 14 (Partial order on partial functions) 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 $\bar{f}$. Formally:

$$\bar{f} \sqsubseteq \bar{f}' \quad \triangleq \quad \text{dom}(\bar{f}) \subseteq \text{dom}(\bar{f}') \,\wedge\, f =_{\text{dom}(\bar{f})} f'$$

Remark: the antisymmetry property associated with the order on partial functions states that if both $\bar{f}$ extends $\bar{f}'$ and $\bar{f}'$ extends $\bar{f}$ then the two functions $\bar{f}$ and $\bar{f}'$ are equivalent, i.e. $\bar{f} \overset{\hookrightarrow}{=} \bar{f}'$. The next two definitions formalize the notion of optimal fixed point.

Definition 15 (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 [19], generally-consistent fixed points are the only genuine solutions of any circular function definition.

Definition 16 (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.*

Note that the optimal fixed point is always unique modulo the equivalence between partial functions. 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.

3.5 Contributions of this paper

Our contribution can be summarized as follows.

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.

The result of our work is a general approach to the definition of recursive, corecursive and mixed recursive-corecursive values. It supports nested recursion, higher-order recursion, and offers a proper treatment of partial functions in the sense that domains need not be hardwired in the definition of the functionals. Curried n-ary functions and mutually-recursive definitions are also supported. Our results have all been implemented in Coq and can be conveniently used in practice to formalize a large scope of circular definitions.

4 The greatest fixed point combinator

We now introduce a greatest fixed point combinator and show how to instantiate it as a unique fixed point combinator and as an optimal fixed point combinator.

4.1 Definition of the greatest fixed point combinator

The idea behind the greatest fixed point combinator Fix is very simple. Our goal is to construct a fixed point modulo $\equiv$ of a given functional F. Since several fixed point might exist, we pick the "best" fixed point, for a suitable notion of "best" described through an order relation written $\lhd$. 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 "$\mathsf{greatest} \prec\ P\,x$", which asserts that x satisfies P and that x is greater than any other value satisfying P, with respect to $\prec$.

Definition 17 (The greatest fixed point combinator)

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

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

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

4.2 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 18 (Another unique fixed point combinator)

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

We thereby obtain a combinator that picks the unique fixed point when it exists. 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 19 (Combinator for unique fixed point modulo)

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

The particular combinator of FixVal is in fact defined as "$\mathsf{FixValMod}\,(=)$".

4.3 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 20 ("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:

$$\mathsf{partialize}\,F \quad \triangleq \quad \lambda(f, D).\,(F\,f, D)$$

Definition 21 (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 $\overset{\hookrightarrow}{=}$.

There exists another, equivalent definition of partial fixed point (e.g., [16]). Recall that $f =_D f'$ is an abbreviation for $\forall x.\, D\, x \Rightarrow f\, x = f'\, x$.

Definition 22 (Alternative definition of partial fixed points) A partial function (f, D) is a *partial fixed point* of F if and only if the equation $F\, f' =_D f'$ holds for every function f' such that $f' =_D f$.

Remark: one needs to quantify over f' to ensure that the computation of "$F\, f$" does not depend on the values produced by f outside the domain D.

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.

Definition 23 (Partial order selecting the optimal fixed point)

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

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 24 (The optimal fixed point combinator)

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

The following theorem justifies the fact that the above definition indeed constructs the optimal fixed point. It relates the definition of the optimal fixed point, which is the greatest generally-consistent fixed point of F with respect to $\sqsubseteq$, with the definition of FixFun, which picks the greatest fixed points of "$\mathsf{partialize}\, F$" modulo $\overset{\hookrightarrow}{=}$ with respect to $\ll_F$.

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

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. In this case, equivalence between partial functions, written $\overset{\hookrightarrow}{\equiv}$, is defined in terms of the predicate $f \equiv_P f'$ which compares result modulo $\equiv$, and which is defined formally as $\forall x.\, P\, x \Rightarrow f\, x \equiv f'\, x$. The construction results in a more general combinator, called FixFunMod, which takes an equivalence relation $\equiv$ and a functional F as argument and returns a fixed point which is optimal modulo $\overset{\hookrightarrow}{\equiv}$.

5 The general fixed point theorem and its corollaries

5.1 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 [21] 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 25 (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

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

Note that contrary to the combined contraction condition from §3.2, the invariant Q is assumed both for x and y and not just for x. Assuming Q only for x would lead to a too weak condition.

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. The notions of limits and of downward-closedness used below are those introduced in §3.3. The definition of continuity can be skipped in a first reading.

Definition 26 (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.

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

We can now state the general fixed point theorem for c.o.f.e.'s.

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].

5.2 Fixed point theorem for corecursive values

By the above theorem, when F is a contractive functional, it admits a unique fixed point. In this case, it can thus be proved that the unique fixed point combinator FixVal picks a fixed point of F. More generally, when F is a contractive functional modulo $\equiv$, it admits a unique fixed point modulo $\equiv$. The combinator FixValMod picks a value which is a fixed point of F. Those results are captured by the following theorem.

Theorem 4 (Fixed point theorem for FixValMod)

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

Compared with previous work, the use of an invariant in the contraction condition makes it strictly more expressive. Intuitively, while reasoning about a *recursive* function with nested calls requires the ability to specify results of the function, reasoning on a *corecursive* function with nested calls requires the ability to specify arbitrarily-long prefixes of its outputs. The invariant Q describes the specification of those prefixes. Example 3, which appears further on, illustrates this fact.

Let us first start with simpler examples. We define functions on streams, using the c.o.f.e. for streams introduced in §3.3, where $\overset{i}{\approx}$ captures equality between streams up to length i and where $\equiv$ corresponds to stream bisimilarity.

Example 1 (A simple well-defined stream) The stream of natural numbers $s_\mathbb{N}$ can be defined as "$\mathsf{FixValMod}\,(\equiv)\,F_9$", where "$F_9\,s \triangleq 0 :: (\mathsf{map}\,\mathsf{succ}\,s)$". The fixed point theorem can be invoked to establish the fixed point equation $s_\mathbb{N} \equiv F_9\,s_\mathbb{N}$. By definition of F_9, it is equivalent to the equation $s_\mathbb{N} \equiv 0 :: (\mathsf{map}\,\mathsf{succ}\,s_\mathbb{N})$, which can be used to "unfold" the definition of $s_\mathbb{N}$ at any time. For example, three such unfolding operations produce the equation $s_\mathbb{N} \equiv 0 :: 1 :: 2 :: 3 :: (\mathsf{map}\,\mathsf{succ}\,s_\mathbb{N})$.

In order to invoke the fixed point theorem, we first need to specify the invariant Q. In this simple example, we define it simply as the predicate that always returns True. This degenerated invariant is clearly continuous. The second step is to prove that the functional "F_9" is contractive.

$$\forall s_1\, s_2\, i.\ (\forall j \prec i.\ s_1 \overset{j}{\approx} s_2) \Rightarrow F_9\,s_1 \overset{i}{\approx} F_9\,s_2$$

To prove this goal, it suffices to show $(0 :: \mathsf{map}\,\mathsf{succ}\,s_1) \overset{i}{\approx} (0 :: \mathsf{map}\,\mathsf{succ}\,s_2)$. If i is equal to zero, the goal is trivial. Otherwise, we need to prove $(\mathsf{map}\,\mathsf{succ}\,s_1) \overset{i-1}{\approx} (\mathsf{map}\,\mathsf{succ}\,s_2)$. This fact follows directly from $s_1 \overset{i-1}{\approx} s_2$, which comes from the hypothesis on s_1 and s_2.

Example 2 (A stream that is not well-defined) To see a counter-example where the contraction condition is not satisfied, recall the invalid circular definition $F_{11}\,s \triangleq 0 :: (\mathsf{tail}\,s)$. To prove the contraction condition associated with the functional F_{11}, we would need to prove that $(\mathsf{tail}\,s_1) \overset{i-1}{\approx} (\mathsf{tail}\,s_2)$, which would in turn require a proof of $s_1 \overset{i}{\approx} s_2$. The assumption that $s_1 \overset{j}{\approx} s_2$ holds for any $j < i$ is clearly too weak to derive this conclusion. Intuitively, the fact that the proof fails because s_1 and s_2 are not know to be similar beyond the rank "$i-1$" translates the idea that the functional is not "productive" enough.

Example 3 (A stream that requires an invariant) As a third and last example, we consider the circular definition associated with the functional $F_{17}\, s \triangleq 1 :: (\mathsf{filter}\,(\geq a)\, s)$, where a is a value less or equal to 1. Justifying the existence of a unique fixed point for s requires the use of an invariant specifying some property of the resulting stream. For example, let us specify that all the values in the stream described by F_{17} are greater than or equal to 1. More precisely, the define Q such that "$Q\, i\, s$" holds if the i first elements of s are greater than or equal to 1. The contraction condition is:

$$\forall s_1\, s_2\, i.\ (\forall j \prec i.\ s_1 \overset{j}{\approx} s_2 \ \wedge\ Q\, j\, s_1 \ \wedge\ Q\, j\, s_2) \ \Rightarrow\ F_{17}\, s_1 \overset{i}{\approx} F_{17}\, s_2 \ \wedge\ Q\, i\,(F\, s_1)$$

For the first part of the conclusion, we need to show "$\mathsf{filter}\,(\geq a)\, s_1$" and "$\mathsf{filter}\,(\geq a)\, s_2$" similar up to index "$i-1$". By assumption, both s_1 and s_2 contain only element greater than or equal to 1 up to index "$i-1$", and s_1 and s_2 are equal up to index "$i-1$", so the result holds by transitivity, as follows:

$$\mathsf{filter}\,(\geq a)\, s_1 \overset{i-1}{\approx} s_1 \overset{i-1}{\approx} s_2 \overset{i-1}{\approx} \mathsf{filter}\,(\geq a)\, s_2$$

For the second part of the conclusion, we need to show that "$1 :: (\mathsf{filter}\,(\geq a)\, s_1)$" contains only element greater than or equal to 1 up to index i, knowing that "s_1" itself contains only such items up to index "$i-1$". To that end, it suffices to check that, under this assumption, "$\mathsf{filter}\,(\geq a)\, s_1$" is in fact equal to s_1 up to length "$i-1$".

The fixed point theorem also tells us that the fixed point of F_{17} satisfies the invariant "$Q\, i$" for all i. Here, it means that the fixed point contains only elements greater than or equal to 1 on its i first elements, for any value of i. We could in fact have used a more precise invariant throughout the proof, defining Q such that "$Q\, i\, s$" holds if the i first elements of s are exactly equal to 1. In this case, the fixed point theorem would have let us conclude that the fixed point of F_{17} contains only values equal to 1.

5.3 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.

The main difficulty in this proof is to establish completeness. Given a sequence u of functions of type $A \to B$, indexed with values of type A, we need to show that if this sequence is coherent then it admits a limit. This limit function can be defined by diagonalization, as "$\lambda x.\, u\, x\, x$". The proof that it is indeed a limit is relatively straightforward.

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 3 and Definition 4).

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 the following combined contraction condition on the domain D with respect to $<$ and S modulo $\equiv$:*

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

Remark: we need not include an assumption $S\, y\, (f_2\, y)$ because it would be redundant with $S\, y\, (f_1\, y)$. Indeed, we already have $f_1\, y \equiv f_2\, y$ and S is compatible with $\equiv$. A corollary, not shown here, to the general fixed point theorem (Theorem 22) 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 $\forall x.\, D\, x \Rightarrow S\, x\, (f\, x)$. More precisely, the general fixed point theorem (Theorem 22) can be specialized to the particular case of recursive functions as follows. Recall that "$f_1 \equiv_D f_2$" stands for the proposition $\forall x.\, D\, x \Rightarrow f_1\, x \equiv f_2\, x$.

Theorem 7 (Fixed point theorem for recursive functions) *Let $\equiv$ be an equivalence relation, $<$ be a well-founded relation, D be a domain, S be a post-condition compatible with $\equiv$. If the functional F satisfies the combined contraction condition on the domain D with respect to $<$ and S modulo $\equiv$, then F admits a unique partial fixed point f modulo $\equiv_D$. Moreover, this fixed point f satisfies the post-condition S on the domain D, that is, $\forall x.\, D\, x \Rightarrow S\, x\, (f\, x)$.*

The next key theorem in our development establishes that the partial fixed point (f, D) is a *generally-consistent* fixed point of the functional "$\mathsf{partialize}\, F$".

Theorem 8 (General consistency for recursive fixed points) *Under the hypotheses of the previous theorem, the partial function (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 [16].

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. This result, which is central to the paper, is the matter of our final theorem for recursive functions, which is the only one that the end-user needs to invoke directly.

Theorem 9 (Specification of FixFunMod for recursive functions)

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

A corollary can be immediately deduced for the combinator FixFun, by instantiating $\equiv$ as $=$. In this case, the second hypothesis ($\equiv$ is an equivalence) and the fourth hypothesis (S is compatible with $\equiv$) are not needed as they are always verified. Another useful corollary is the one for functions that do not make nested recursive calls. In this case, there is no need to involve an invariant, i.e. S can be instantiated as the predicate that always returns True.

5.4 Fixed point theorem for corecursive functions

While the previous section focused on the construction of a c.o.f.e. for recursive functions, this section details the construction of a c.o.f.e. for corecursive functions. Compared with the construction proposed in [21], we have added support for partial functions and for nested calls.

Our goal is now to build a c.o.f.e. to prove that a functional of type $(A \to B) \to (A \to B)$ describing a corecursive function on a domain D admit a unique fixed point. Here, the values of the result type B are compared using an existing c.o.f.e. $(B, I, \prec, \overset{i}{\approx})$. This c.o.f.e. induces a c.o.f.e. on the function space $A \to B$.

Theorem 10 (c.o.f.e. for corecursive functions) *Let $(B, I, \prec, \overset{i}{\approx})$ be a c.o.f.e., and let D be a domain of type $A \to$ Prop. Then, the structure $(A \to B, A, <, \overset{i}{\approx}_D)$ is a complete ordered family of equivalences, where $(\overset{i}{\approx}_D)_{i:I}$ is a family of equivalence relations such that $f_1 \overset{i}{\approx}_D f_2$ holds if $\forall x.\, D\,x \Rightarrow f_1\,x \overset{i}{\approx} f_2\,x$.*

In this c.o.f.e., the notions of continuity of invariants and of contraction condition can be given a direct formulation, as detailed below.

Theorem 11 (Contraction condition for corecursive functions) *Let D be a domain of type $A \to$ Prop and let S be a indexed post-condition of type $I \to A \to B \to$ 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 corecursive functions, a functional F is contractive with respect to the invariant "$\lambda i\,f.\ \forall x.\, D\,x \Rightarrow S\,i\,x\,(f\,x)$" as soon as F satisfies the condition:*

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

Again, a specialized fixed point theorem can be devised, as well as a result about the general-consistency of the fixed point. It follows a fixed point theorem, similar to Theorem 9, specialized for reasoning on applications of FixFunMod to a functional describing a partial corecursive function.

Theorem 12 (Fixed point theorem for corecursive functions) *Let* $(B, I, \prec, \overset{i}{\approx})$ *be a c.o.f.e., D be a domain, and S be a continuous indexed post-condition. If the functional F satisfies the contraction condition for corecursive function stated above, then F admits a unique partial fixed point f modulo $\approx_D$. Moreover, this fixed point f is generally consistent, and satisfies the property $\forall i\, x.\, D\, x \Rightarrow S\, i\, x\, (f\, x)$.*

The final theorem for corecursive function explains how to derive properties of corecursive functions built using the combinator FixFunMod. In the following statement, the notions of contraction and of continuity are those of Theorem 11.

Theorem 13 (Specification of FixFunMod for corecursive functions)

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

Example 4 (A corecursive function with nested calls) Recall the functional F_{16}, defined as $F_{16}\, f\, s \triangleq$ let $x :: t = s$ in $(2 \cdot x) :: f\,(f\, t)$. Let us prove that the fixed point of this functional, defined as "FixFunMod $(\equiv)\, F$", is a function that f multiplies the i-th element of a stream with the value 2^{2^i}. Here and throughout the example, $\equiv$ stands for stream bisimilarity, and we write "$s[i]$" the i-th element of the stream s. We invoke the theorem above, using the c.o.f.e. for streams, taking the entire type A as domain and defining the invariant S such that $S\, i\, s\, s'$ captures the proposition "$\forall j < i.\ s'[j] = s[j] \cdot 2^{2^j}$". This invariant relates elements of the output stream s' with elements of the input stream s, up to an arbitrary index i. This invariant is continuous: if $S\, i\, s\, s'_1$ holds and if s'_1 is equivalent to s'_2 up to any index less than i, then $S\, i\, s\, s'_2$ also holds.

It remains to show that the functional is contractive in order to derive the fixed point equation $f\, s \equiv F_{16}\, f\, s$, and to derive the expected specification for f. The details of the proof of the contraction condition are left as an exercise to the reader. In short, the first step of this proof is relatively easy: the hypothesis is used to show that "$f_1\, t$" and "$f_2\, t$" are equal up to index $i - 1$. The second step is trickier: one need to prove that "$f_1\, s_1$" and "$f_2\, s_2$" are equal up to index $i - 1$, under the assumption that s_1 and s_2 are equal up to index i. This second step requires one to exploit the fact that both f_1 and f_2 satisfy the invariant S. The last step in the proof relies on the following argument: if the inner application of f multiplies the i-th element of t by 2^{2^i}, and if the outer application of f multiplies again the i-th element of t by 2^{2^i}, then the i-th element of stream t has been multiplied by $2^{2^{i+1}}$. Therefore, the $i+1$-th element of the stream $(2 \cdot x) :: f(f(t))$ is equal to the $i+1$-th element of the stream $x :: t$ multiplied by 2^{2^i+1}, as expected.

5.5 Fixed point theorem for mixed recursive-corecursive functions

The two previous sections have focused on the reasoning on recursive functions and on corecursive functions. This section addresses a strictly-more general class of functions:

those mixing recursion and corecursion. A function from this class produces a value of coinductive type. Each recursive call must either be guarded by some constructor (in which case the function produces some structure from its output), or the recursive call can be made to an argument strictly-smaller than the current argument (smaller with respect to a given well-founded relation). Here again, our construction of a c.o.f.e. for mixed recursive-corecursive functions extends the one given by Matthews [21] by adding support for partial functions and 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 14 (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 15 (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:*

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

The major difference with the contraction condition for simple recursive functions is in the lexicographical comparison $(j,y) <' (i,x)$. It means that a recursive call is allowed to be either performed at a level j strictly smaller than the current level i with respect to $\prec$, which typically corresponds to the case where the recursive call is guarded by some constructor, or the recursive call can be performed at the original level i, but in this case it must be on an argument y strictly smaller than x with respect to $<$.

The fixed point theorem (not shown here) based on this extended contraction condition leads to the following specification of the combinator $\mathsf{FixFunMod}$.

Theorem 16 (Specification of FixFunMod for mixed functions)

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

Let us apply this theorem to the filter function. Let f be the function $\mathsf{FixFunMod}\,(\equiv)\,F_{15}$, where F_{15} 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 [21]. 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 $f\,s \equiv F_{15}\,f\,s$ holds for any stream s in the domain D.

6 Code Extraction

In this section, we investigate the possibility for extracting executable code from circular definitions constructed in terms of our fixed point combinators.

6.1 Towards code extraction for fixed point combinators

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. Extracted code enjoys a partial correctness property, which we now describe informally. Suppose that a function f defined in the logic admits a pre-condition P and a post-condition Q, meaning that for any argument x satisfying P, the result y of "$f\,x$" satisfies $Q\,x\,y$. Then, the executable function extracted from f also admits P and Q as pre- and post-conditions, meaning that for any argument x satisfying P, if the execution of "$f\,x$" terminates, then its result y satisfies $Q\,x\,y$.

It might be surprizing that extracted code does not necessarily terminate, although the logic admit only total functions. For example, we can construct a function loops whose extracted code diverge in call-by-value evaluation. It is defined in such a way that "$\mathsf{loops}\,x$" recursively calls "$\mathsf{loops}\,x$", but with the value of the recursive call being ignored so that the recursive call becomes irrelevant from a logical point of view. The example can be written in Coq as follows.

```
Definition ignore (n:nat) : nat := 0.
Fixpoint loops (x:nat) : nat := ignore (loops x).
Extraction loops. (* let rec loops x = ignore (loops x) *)
```

The definition of loops is accepted in Coq because the check that recursion is made on structurally-smaller values is allowed to perform certain forms of unfolding (see [3] for further explanations). A similar definition can be defined in Isabelle/HOL. In this case, the definition is accepted because one can prove a congruence rule asserting that "$\mathsf{ignore}\,n$" is equal to "$\mathsf{ignore}\,m$" for any values of n and m. This property suffices to show that a fixed point equation can be provided for the function loops without breaking consistency of the logic. For a similar reason, any tail-recursive function can be safely defined in the logic, because it always admits a fixed point. This observation was exploited in ACL2 [20], and later implemented in Isabelle/HOL [13] and HOL4 [22].

Given that code extraction mechanisms provide no more than a partial correctness result, we would be relatively happy if we were able to extract our circular definitions towards code that produces correct results whenever it terminates. Yet, we face one major difficulty. The definition of Fix relies on Hilbert's epsilon operator, which is a non-constructive axiom that does not admit an executable counterpart. Thus, it does not seem immediate to extract Fix towards executable code. Nevertheless, it turns out to be possible to devise a piece of functional code that can be used to extract the constant Fix in a correct way, with respect to partial correctness. This piece of code simply relies on the native "let-rec" construct from the target programming language.

Our experiments suggest that such an extraction of the fixed point combinators lead to correct and efficient programs. This approach thus appears to be very interesting to generate certified programs making use of advanced forms of recursive and corecursive definitions. 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. [18]). Worse, 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, which is precisely what we do here. Thus, we leave the proof of correctness of our approach as a challenge to code extraction experts, and simply explain how to set up and test the extraction process in practice.

6.2 Extraction of the fixed point combinators in Haskell

In Haskell, where evaluation is lazy by default, the extraction of the constant Fix is very simple. Indeed, it suffices to extract "$\mathsf{Fix}\, F$" towards "$\mathsf{let}\, x = F\, x \,\mathsf{in}\, x$". This definition works both for recursive and corecursive values. The Coq command used to force extraction of the constant Fix appears next. Note that the variable F is renamed to bigf in order to meet syntactic requirements.

```
Extract Constant Fix => "(\bigf -> let x = bigf x in x)".
```

It is crucial to check that the type of the function "$\lambda F.\, \mathsf{let}\, x = F\, x \,\mathsf{in}\, x$" has an appropriate type with respect to the type of the constant Fix in the logic. Otherwise, the extraction of the definitions that depend on Fix would all be ill-typed. In the logic, the type of Fix is as follows:

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

The type of Fix is polymorphic in the type A, which is the type of the value to be defined circularly. The combinator Fix first expects a proof that the type A is inhabited. This proof is encapsulated within the inductive data type Inhabited, which admits the type "$\mathsf{Type} \to \mathsf{Prop}$". Because this argument lies in the world of propositions, it is erased through extraction. The next two arguments are the binary relations $\equiv$ and $\lhd$ of type "$A \to A \to \mathsf{Prop}$". They are also erased through the extraction process. The last argument is the functional of type $A \to A$, which is preserved by extraction. It follows that the combinator Fix must be extracted towards a value of type "$\forall A.\, (A \to A) \to A$". This is indeed the type of "$\lambda F.\, \mathsf{let}\, x = F\, x \,\mathsf{in}\, x$".

To illustrate the way extraction works, let us test the behaviour of the filter function on streams (desribed by functional F_{15}), which is a partial corecursive function. Firstly, suppose we compute the stream obtained by filtering even numbers from the stream of natural numbers. If we extract the corresponding Haskell code and write a command to print, say, the 5 first elements of the sequence, the program displays "0 :: 2 :: 4 :: 6 :: 8", which indeed corresponds to beginning of the stream of even numbers. Secondly, suppose we want to compute the stream obtained by filtering natural numbers less than 3 from the stream of natural numbers. If we extract the corresponding Haskell code and write a command to print the 3 first elements of the sequence, the program terminates and displays "0 :: 1 :: 2". However, if we ask for more elements, the program loops forever, as it fails to find any other element less than 3 in the remaining of the stream of natural numbers.

6.3 Extraction of the fixed point combinators in OCaml

In OCaml, extraction is a little more complex due to the fact that evaluation is strict by default, which means that corecursive values need to be explicitly tagged as lazy. As a consequence, we need to introduce two distinct pieces of code: one to extract the combinator for functions and another to extract the combinator for corecursive values.

The combinator for functions FixFun (or its generalization FixFunMod) takes as argument a functional F of type $(A \to B) \to (A \to B)$ and returns its optimal fixed point, of type $A \to B$. We extract FixFun as a function that maps F to its computational fixed point, defined as "$\mathsf{let\,rec}\, f\, x = F\, f\, x\, \mathsf{in}\, f$". The corresponding Coq command appears next.

```
Extract Constant FixFun => "(fun bigf -> let rec f x = bigf f x in f)".
```

The key intuition involved here is that whenever the computation fixed point terminates, it necessarily returns the same value as the optimal fixed point.

The combinator for values FixVal (or its generalization FixValMod) can be extracted only when it is used to produce a value that admits a coinductive type in the logic (which is generally the case). Indeed, only inductive types are extracted using OCaml's lazy type. In this case, the combinator FixVal takes as argument a functional F of type "α Lazy.t $\to \alpha$ Lazy.t", and returns a value of type "α Lazy.t". Intuitively, we would like to define the fixed point of F as "$\mathsf{let}\, x = F\, x\, \mathsf{in}\, x$", like in Haskell, but this definition is not accepted by OCaml's compiler, which enforces a strict constraint on the form of recursive value definitions. To work around this limitation, it suffices to insert an explicit lazy keyword, immediately followed by a call to Lazy.force, at the head of the definition of the fixed point x.

```
Extract Constant FixVal =>
  "(fun bigf -> let rec x = lazy (Lazy.force (bigf x)) in x)".
```

In conclusion, the definition of combinators in OCaml is slightly less direct than in Haskell, but is nevertheless possible. We end this section by pointing out that direct implementations can be devised in order to extract fixed point combinators for curried n-ary functions and mutually-recursive functions. Direct implementations benefit from improved efficiency, because they avoid the encodings based on binary pairs and binary sums that come with the default implementation. For example, the combinator FixFunMod2 for taking the fixed point of a function of arity two can be extracted as follows.

```
Extract Constant FixFunMod2 =>
   "(fun bigf -> let rec f x y = bigf f x y in f)".
```

7 Other related work

The most closely related work has already been covered throughout §3. 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 [15].)

The package TFL developed by Slind [24] supports the definition of total recursive functions for which a well-founded termination relation can be exhibited. Building on Slind's ideas, Krauss [15] 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. In particular, the package support nested recursion through a provisional induction principle and supports higher-order recursion through congruence rules. 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 [23], 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 [25] implemented facilities for manipulating subset types in Coq, including a fixed point combinator for functionals of type

$$\big(\forall x : A.\ \big(\forall y : \{y : A \,|\, R\,y\,x\}.\ B\,(\pi_1\,y)\big) \Rightarrow B\,x\big) \quad \Rightarrow \quad \forall x : A.\,(B\,x)$$

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]. Intuitively, the key idea is to define an auxiliary coinductive type that captures both the steps of corecursion and the steps of recursion. More recently, Bertot and Komendantskaya [6] experimented reasoning about non-guarded corecursive definitions by exploiting the correspondence between streams and functions over natural numbers. While this approach is entirely constructive, their authors acknowledge that it is somewhat limited.

8 Conclusion

The theory developed in this paper can be used to formalize a fairly large class of circular definitions. Our development provides several combinators and fixed point theorems. They all derive from a single greatest fixed point combinator and from a single general fixed point theorem for contraction conditions. We have formalized all this material using the Coq proof assistant. A similar development could presumably be reproduced in any other general-purpose theorem prover based on higher-order logic.

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. Defining recursive functions in isabelle/hol, 2006.
14. Alexander Krauss. Partial recursive functions in higher-order logic. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *LNCS*, pages 589–603. Springer, 2006.
15. Alexander Krauss. Partial and nested recursive function definitions in higher-order logic. *Journal of Automated Reasoning*, December 2009. To appear.
16. Sava Krstić. Inductive fixpoints in higher order logic. February 2004.
17. 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.

18. Pierre Letouzey. Programmation fonctionnelle certifiée : L'extraction de programmes dans l'assistant Coq, June 01 2007.
19. Zohar Manna and Adi Shamir. The theoretical aspects of the optimal FixedPoint. *SIAM Journal on Computing*, 5(3):414–426, September 1976.
20. Panagiotis Manolios and J Strother Moore. Partial functions in ACL2. 2003.
21. 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.
22. Magnus O. Myreen and Michael J. C. Gordon. Verified LISP implementations on ARM, x86 and powerPC. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *LNCS*, pages 359–374. Springer, 2009.
23. Bengt Nordström. Terminating general recursion. *BIT*, 28(3):605–619, 1988.
24. Konrad Slind. *Reasoning about Terminating Functional Programs.* PhD thesis, Institut für Informatik, Technische Universität München, 1999.
25. Matthieu Sozeau. Subset coercions in Coq. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *LNCS*, pages 237–252. Springer, 2006.

Appendix A: Equivalence of the definitions of partial fixed points

Let us prove the two definitions of partial fixed point equivalent. Let F be a functional of type $(A \to B) \to (A \to B)$. In the sense of Definition 21, $\bar{f}$ is a partial fixed point of F if

$$\forall \bar{f}'.\quad \bar{f}' \overset{\hookrightarrow}{\equiv} \bar{f} \quad\Rightarrow\quad \bar{f}' \overset{\hookrightarrow}{\equiv} \mathsf{partialize}\, F\, \bar{f}'$$

Let us write D the domain of $\bar{f}$ and D' the domain of $\bar{f}'$, and expand the definition of $\mathsf{partialize}$. We get:

$$\forall f'\, D'.\quad (f', D') \overset{\hookrightarrow}{\equiv} (f, D) \quad\Rightarrow\quad (f', D') \equiv (F\, f', D')$$

Unfolding now the definition of $\equiv$ gives:

$$\forall f'\, D'.\quad (D' = D \,\wedge\, f' =_D f) \quad\Rightarrow\quad (D' = D' \,\wedge\, f' =_{D'} f)$$

The above is equivalent to

$$\forall f'.\quad f' =_D f \quad\Rightarrow\quad f' =_D F\, f'$$

which asserts that (f, D) is a partial fixed point in the sense of Definition 22.

Appendix B: Proof ot the optimal fixed point theorem

Let us prove Theorem 1, which asserts the existence of an optimal fixed point for any functional F of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$. Our proof is directly adapted from the original proof proposed by Manna and Shamir [19]. For the sake of presentation, we assume values of the type B are using Leibnitz' equality $(=)$, but the Coq implementation supports comparison with respect to an arbitrary equivalence relation $\equiv$.

Throughout the development, we identify sets of values of type T with predicates of type $T \to \mathsf{Prop}$. In particular, we write $x \in P$ to mean that "$P\, x$" holds.

Definition 27 (Consistent sets) A S of partial functions is a *consistent set* if its elements are pairwise consistent, that is, $\forall \bar{f}\, \bar{f}' \in S.\ \bar{f} \triangledown \bar{f}'$.

Theorem 17 (Existence of a lub for consistent sets) *Any consistent set S of partial functions admits a least upper bound with respect to $\sqsubseteq$.*

Proof We say that a function $\bar{f}$ *covers* a point x if $\bar{f}$ belongs to S and x belongs to the domain of $\bar{f}$. Let D be the set of all covered points, that is, the points x such that there exists a function $\exists \bar{f}.$ that covers x. Let g be the function defined on D that, given an argument x, returns the application of any function from S that covers x to the value x. Formally:

$$\begin{array}{rcl}\mathsf{covers}\,\bar{f}\,x & \triangleq & \bar{f} \in S \,\wedge\, x \in \mathrm{dom}(\bar{f}) \\ D & \triangleq & \lambda x.\, \exists \bar{f}.\, \mathsf{covers}\,\bar{f}\,x \\ g & \triangleq & \lambda x.\, \mathsf{if}\ x \in D\ \mathsf{then}\ (\epsilon \bar{f}.\, \mathsf{covers}\,\bar{f}\,x)\,x\ \mathsf{else\ arbitrary}\end{array}$$

Let $\bar{g}$ be the partial function (g, D). We want to prove that $\bar{g}$ is the least upper bound of the set S.

First, let us show that it is an upper bound, that is, $\forall \bar{f} \in S.\ \bar{f} \sqsubseteq \bar{g}$. Let $\bar{f}$ be a function in S. For the domains, we have $\mathrm{dom}(\bar{f}) \subseteq \mathrm{dom}(\bar{g})$, because D contains $\{x \mid \mathsf{covers}\,\bar{f}\,x\}$. For the return values, let x be in $\mathrm{dom}(\bar{f})$ and let us prove that $f(x) = g(x)$. By definition of g, the value $g(x)$ is equal to $f'(x)$ for some $\bar{f}' \in S$. By consistency of S, which contains both $\bar{f}$ and $\bar{f}'$, the value $f'(x)$ is equal to $f(x)$. Hence, by transitivity, $f(x) = g(x)$.

Second, let us show that $\bar{g}$ is the smaller upper bound. Let $\bar{g}'$ be another upper bound of S, i.e. $\forall \bar{f} \in S.\ \bar{f} \sqsubseteq \bar{g}'$. Our goal is to prove $\bar{g} \sqsubseteq \bar{g}'$. Let x be in the domain of $\bar{g}$. By definition of $\bar{g}$, there exists a function $\bar{f}$ in S such that $x \in \mathrm{dom}(\bar{f})$ and $g(x) = f(x)$. By assumption, $\bar{f} \sqsubseteq \bar{g}'$. Therefore x belongs to the domain of $\bar{g}'$ and $g'(x) = f(x)$. By transitivity, it follows that $g(x) = g'(x)$. Hence, $\bar{g}$ extends $\bar{g}'$. □

Theorem 18 (Lub of consistent sets of fixed points) *Let F be a functional of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$. The least upper bound of a consistent set of partial fixed points of F is always a partial fixed point of F.*

Proof The previous theorem asserts the existence of the least upper bound $\bar{g}$ of a consistent set S, and describes it as the pair (g, D). Let us show that $\bar{g}$ is a partial fixed point of F, under the assumption that all the elements of the set S are partial fixed point of F. The goal is: $\forall g'.\, g' =_D g \Rightarrow g' =_D F\,g'$. Let g' be such that $g' =_D g$, let x be a value in D and let us prove that $g'\,x = F\,g'\,x$. Since $x \in D$, there exists a function $\bar{f}$ that covers x, i.e. such that $\bar{f} \in S$ and $x \in \mathrm{dom}(\bar{f})$. Because $\bar{f}$ is in S, it is, by assumption, a partial fixed point of F, meaning that the equation $f'\,x = F\,f'\,x$ holds for any f' such that $f' =_{\mathrm{dom}(\bar{f})} f$. So, we can prove the goal $g'\,x = F\,g'\,x$ by instantiating f' as g'. It remains to establish the premise $g' =_{\mathrm{dom}(\bar{f})} f$.

To that end, let y be a value in $\mathrm{dom}(\bar{f})$. We need to prove $g'(y) = f(y)$. Since $\bar{f} \in S$ and $y \in \mathrm{dom}(\bar{f})$, the function $\bar{f}$ covers y, therefore we have $y \in D$. From $y \in D$ and $g' =_D g$, we deduce $g'(y) = g(y)$. By transitivity we are left to prove $f(y) = g(y)$. This fact is a consequence of $\bar{f} \sqsubseteq \bar{g}$, which comes from the fact that $\bar{f} \in S$ and that $\bar{g}$ is defined as the lub of S. □

Theorem 19 (Existence of the optimal fixed point) *Any functional of type $(A \hookrightarrow B) \to (A \hookrightarrow B)$ admits a greatest generally-consistent partial fixed point.*

Proof Let F be the functional. Let S be the set of generally-consistent partial fixed point of F. Clearly, S is a consistent set. Indeed, if $\bar{f}$ and $\bar{f}'$ are two generally-consistent partial fixed point, then $\bar{f}$, which is a generally-consistent partial fixed point, is certainly consistent with the partial fixed point $\bar{f}'$.

By Theorem 17, the consistent set S admits a lub $\bar{g}$. Thus, $\bar{g}$ is greater than any generally-consistent partial fixed point of F. We want to show that $\bar{g}$ itself is a generally-consistent partial fixed point of F. By Theorem 18, we know that $\bar{g}$ is a partial fixed point of F. Thus, it remains to show that $\bar{g}$ is generally-consistent.

To that end, let $\bar{f}$ be any partial fixed point of F. Our goal is to prove that $\bar{f}$ is consistent with $\bar{g}$, that is, $\bar{f} \triangledown \bar{g}$. Let S' be the set $S \cup \{\bar{f}\}$, which is a set of partial fixed point of F. Moreover, we can show that S' is a consistent set, as follows. We know that S and $\{\bar{f}\}$ are two consistent sets, and we can check that any element $\bar{f}'$ from S is consistent with $\bar{f}$. Indeed, by definition of S, $\bar{f}'$ is a generally-consistent fixed point. Thus, $\bar{f}'$ is consistent with the partial fixed point $\bar{f}$. It follows that $S \cup \{\bar{f}\}$ is consistent.

By Theorem 17 again, the consistent set S' admits a lub $\bar{h}$. To prove that $\bar{f}$ and $\bar{g}$ are consistent, the plan is to show that $\bar{h}$ extends both $\bar{f}$ and $\bar{g}$. First, $\bar{f} \sqsubseteq \bar{h}$ holds because $\bar{h}$ is the lub of the set S', which contains $\bar{f}$. Second, we can show $\bar{g} \sqsubseteq \bar{h}$. On the one hand, $\bar{g}$ is a generally-consistent partial fixed point, so it belongs to S. On the other hand, $\bar{h}$ is the lub of S' and is thus greater than any element of S, hence greater than $\bar{g}$. Third, let us prove $\bar{f} \triangledown \bar{g}$. Let x be a value that belongs both to the domain of $\bar{f}$ and to the domain of $\bar{g}$. Because $\bar{h}$ extends both $\bar{f}$ and $\bar{g}$, x also belongs to the domain of $\bar{h}$ and we have $f(x) = h(x)$ and $g(x) = h(x)$. Hence, $\bar{f}$ and $\bar{g}$ are equal on the intersection of their domains. □

Appendix C: Definition of coherence, limits and completeness

The definitions of local and global notions of coherence and limits are directly adapted from the work of Matthews [21]. Throughout the section, let $(A, I, \prec, \overset{i}{\approx})$ be a c.o.f.e..

Definition 28 (Local coherence) A sequence u of type $I \to A$ is *locally-coherent at level* i, written $\mathsf{locally\_coherent}\, u\, i$, if it is coherent on the domain $\{j \mid j \prec i\}$. This is equivalent to satisfying the proposition $\forall j\, k.\, k \prec j \prec i \;\Rightarrow\; u\, k \overset{k}{\approx} u\, j$.

Definition 29 (Local limit)

$$\mathsf{lim}_i\, u \quad \triangleq \quad \epsilon\, l.\ (\forall j \prec i.\ u\, j \overset{j}{\approx} l)$$

Lemma 1 (Specification of local limits)

$$l = \mathit{lim}_i\, u \quad \wedge \quad \mathit{locally\_coherent}\, u\, i \quad \Rightarrow \quad \forall j \prec i.\, u\, j \overset{j}{\approx} l$$

Proof The sequence u is coherent on the domain $\{j \mid j \prec i\}$, thus by completeness of the c.o.f.e., it admits a limit l on that domain. □

Definition 30 (Global coherence) A sequence u of type $I \to A$ is *globally-coherent*, written $\mathsf{globally\_coherent}\, u$, if it is coherent on the set of all indices of type I. This is equivalent to satisfying the proposition $\forall i\, j.\, i \prec j \;\Rightarrow\; u\, i \overset{i}{\approx} u\, j$.

Definition 31 (Global limit)

$$\mathsf{lim}\, u \quad \triangleq \quad \epsilon\, l.\ (\forall i.\ u\, i \overset{i}{\approx} l)$$

Lemma 2 (Specification of global limits)

$$l = \mathit{lim}\, u \quad \wedge \quad \mathit{globally\_coherent}\, u \quad \Rightarrow \quad \forall i.\, u\, i \overset{i}{\approx} l$$

Proof The sequence u is coherent on the set of all indices of type I, thus by completeness of the c.o.f.e. it admits a limit l on that domain. □

Appendix D: Alternative characterisations of completeness

Matthews [21] worked in terms of *local completeness* and *global completeness.* The definition of completeness introduced by di Gianantonio and Miculan [10] is more elegant and eases the construction of c.o.f.e.'s on top of other c.o.f.e.'s. They proved that completeness is equivalent to the conjunction of local completeness and global completeness. This alternative caracterization of completeness is needed to build initial c.o.f.e.'s, in particular c.o.f.e.'s indexed by natural numbers. Due to the interest of local and global compeleteness, we reproduce here the proof of equivalence between the two characterizations of completeness.

Definition 32 (Local completeness) An ordered family of equivalence $(A, I, \prec, \overset{i}{\approx})$ is *locally-complete* if any locally-coherent sequence u at index i admits a local limit l in the sense that $\forall j \prec i.\ u\, j \overset{j}{\approx} l$.

Definition 33 (Global completeness) An ordered family of equivalence $(A, I, \prec, \overset{i}{\approx})$ is *globally-complete* if any globally-coherent sequence u admits a global limit l in the sense that $\forall i.\ u\, i \overset{i}{\approx} l$.

Lemma 3 (Completeness implies local and global completeness) *If an ordered family of equivalence $(A, I, \prec, \overset{i}{\approx})$ is complete, then it is both locally-complete and globally-complete.*

Proof For local completeness, apply the definition of completeness to the domain K defined as $\lambda j.\, j \prec i$. Any locally-coherent sequence u at index i is coherent on the domain K, thus admits a limit on the domain K. This limit is a local limit for u at index i.

For global completeness, use the domain K defined as $\lambda i.\,\mathsf{True}$. Any globally-coherent sequence u is coherent of the domain K, thus admits a limit on the domain K. This limit is a global limit for u.

In both cases, it is immediate to check that the domain K used is downward-closed.

Lemma 4 (Local and global completeness imply completeness) *If an ordered family of equivalence $(A, I, \prec, \overset{i}{\approx})$ is both locally-complete and globally-complete, then it is complete.*

Proof Let u be a sequence coherent on a downward-closed domain K. The goal is to show that u admits a limit l on the domain K. The idea of the proof is to construct a sequence v somehow related to u, show that v is both locally-coherent, that it is globally-coherent, and that the global limit of v is also a limit for u on the domain K. The sequence v is defined by well-founded recursion on indices as follows:

$$v\,i \quad \triangleq \quad \mathsf{if}\ K\,i\ \mathsf{then}\ u\,i\ \mathsf{else}\ \mathsf{lim}_i\,v$$

First, let us prove that v is locally-coherent. By well-founded induction on i, we want to establish that $j \prec k \prec i$ implies $v\,j \overset{j}{\approx} v\,k$. There are two cases. On the one hand, assume that $K\,k$ holds. Because K is downward-closed, $K\,j$ also holds. By unfolding the definition of v, the goal $v\,j \overset{j}{\approx} v\,k$ becomes $u\,j \overset{j}{\approx} u\,k$. This property is an immediate consequence from the assumption that u is coherent on the domain K. On the other hand, assume that $K\,k$ does not hold. The goal $v\,j \overset{j}{\approx} v\,k$ becomes $v\,j \overset{j}{\approx} \mathsf{lim}_k\,v$. This property holds from by Lemma 1. The fact that v is locally-coherent at index k comes from the induction hypothesis.

Second, let us prove that v is globally-coherent. The goal is to show that $j \prec i$ implies $v\,j \overset{j}{\approx} v\,i$. Again, there are two cases. On the one hand, assume that $K\,i$ holds. Because K is downward-closed, $K\,j$ also holds. By unfolding the definition of v, the goal $v\,j \overset{j}{\approx} v\,i$ becomes $u\,j \overset{j}{\approx} u\,i$. This property is an immediate consequence from the assumption that u is coherent on the domain K. On the other hand, assume that $K\,i$ does not hold. The goal $v\,j \overset{j}{\approx} v\,i$ becomes $v\,j \overset{j}{\approx} \mathsf{lim}_i\,v$. This property holds from by Lemma 1. The fact that v is locally-coherent at index i has been established earlier on.

Third, we define l as $\mathsf{lim}\,v$. Given the fact that v is globally coherent, this limit exists (by Lemma 2), and we have $\forall i.\, v\,i \overset{i}{\approx} l$. Our goal is to show that l is a limit for u on the domain K, that is, $\forall i.\, K\,i \Rightarrow\ u\,i \overset{i}{\approx} l$. Let i be an index in the domain K. By definition of v, we have $v\,i = u\,i$, since $K\,i$ holds. Thus, $u\,i \overset{i}{\approx} l$.

Theorem 20 (Alternative characterization of completeness) *An ordered family of equivalence* $(A, I, \prec, \overset{i}{\approx})$ *is complete if and only if it is both locally-complete and globally-complete.*

Proof Combine the previous two lemmas.

Appendix E: Completeness of nat-indexed c.o.f.e.'s

The following theorem is directly adapted from the work of Matthews [21].

Theorem 21 (Completeness of nat-indexed c.o.f.e.'s) *Consider a family of equivalence relations* $(A, \mathsf{nat}, <, \overset{i}{\approx})$*, where i range over the set of natural numbers* nat *and* $<$ *is the standard order on natural numbers. To show this family complete, it suffices to prove the following* nat-completeness *property:*

$$\forall u\,i.\ (\forall i.\, u\,i \overset{i}{\approx} u\,(i+1)) \quad \Rightarrow \quad \exists l.\, \forall i.\, u\,i \overset{i}{\approx} l$$

Proof This proof exploits the alternative definition of completeness (Theorem 20). So, let us establish local completeness and global completeness.

To establish local completeness, we consider a sequence u and an index i such that $\forall j\, k.\, k < j < i \;\Rightarrow\; u\, k \overset{k}{\approx} u\, j$ holds and we must prove that u admits a local limit at index i. We define a auxiliary sequence v as follows:

$$v\, k \quad \triangleq \quad \mathsf{if}\ k < i\ \mathsf{then}\ u\, k\ \mathsf{else}\ u\,(i-1)$$

We construct the limit l of v by applying the assumption of nat-completeness to the sequence u and the index i. The premise to be verified is $\forall k.\, v\, k \overset{k}{\approx} v\,(k+1)$. Let k be a natural number. There are three cases. First, assume that $k+1 < i$. In this case, we also have $k < i$. By definition of v, the goal $v\, k \overset{k}{\approx} v\,(k+1)$ becomes $u\, k \overset{k}{\approx} u\,(k+1)$. This fact follows from the coherence of u at index i, because $k < k+1 < i$. Second, assume that $k+1 = i$. In this case, $k = i-1$. By definition of v, the goal $v\, k \overset{k}{\approx} v\,(k+1)$ becomes $u\, k \overset{k}{\approx} u\,(i-1)$, which holds by reflexivity. Third and last, assume that $k+1 > i$. In this case, the goal $v\, k \overset{k}{\approx} v\,(k+1)$ becomes $u\,(i-1) \overset{k}{\approx} u\,(i-1)$, which also holds by reflexivity.

By construction of l as the limit of v, we have $\forall j.\, v\, j \overset{i}{\approx} l$. Let us show that l is a local limit for u at index i, that is, $\forall j < i.\, u\, j \overset{i}{\approx} l$. Let j be an index less than i. By definition of v, we have $v\, j = u\, j$ since $j < i$. Thus, the property $v\, j \overset{j}{\approx} l$ implies $u\, j \overset{j}{\approx} l$.

To establish global completeness, we consider a sequence u such that $\forall i\, j.\, i < j \;\Rightarrow\; u\, i \overset{i}{\approx} u\, j$ holds and we must prove that u admits a global limit. This limit is directly obtained by application of the nat-completeness assumption. The premise $\forall i.\, u\, i \overset{i}{\approx} u\,(i+1)$ is an immediate consequence of $\forall i\, j.\, i < j \;\Rightarrow\; u\, i \overset{i}{\approx} u\, j$.

Appendix F: Fixed point theorem for contractive functionals

Throughout the section, let $(A, I, \prec, \overset{i}{\approx})$ be a c.o.f.e. and F be a functional of type $A \rightarrow A$ contractive with respect to a continuous invariant Q of type $I \rightarrow A \rightarrow \mathsf{Prop}$.

Definition 34 (Construction of the fixed point)

$$l \triangleq \mathsf{lim}\, u \qquad\qquad \mathsf{where} \qquad\qquad u\, i \triangleq F(\mathsf{lim}_u\, i)$$

The definition of u is by well-founded recursion on the indices i. Indeed, the definition of the local-limit at index i of the sequence u depends only on the values of u at indices smaller than i. Due to this well-founded recursion, the definition is somewhat technical to implement.

Lemma 5 (Invariant for contractive functionals)

$$\forall i\, x.\, (\forall j \prec i.\, Q\, j\, x) \;\Rightarrow\; Q\, i\, (F\, x)$$

Proof Apply the contraction condition with $y = x$ and use the fact that $\overset{j}{\approx}$ is reflexive. □

Lemma 6 (Relation between values of u at different indices)

$$\forall i\, j.\ \textsf{\textit{locally\_coherent}}\, u\, i \ \wedge\ \textsf{\textit{locally\_coherent}}\, u\, j \ \wedge\ (\forall k \prec i.\, Q\, k\, (u\, k)) \ \Rightarrow\ u\, j \overset{j}{\approx} u\, i$$

Proof After unfolding the definition of u, the goal becomes $F(\mathsf{lim}_u\, j) \overset{j}{\approx} F(\mathsf{lim}_u\, i)$. By applying twice Lemma 1 (using the two local-coherence hypotheses), the goal becomes $F\, l \overset{j}{\approx} F\, l'$, with the assumptions $\forall k \prec j.\ v\, k \overset{k}{\approx} l$ and $\forall k \prec i.\ v\, k \overset{k}{\approx} l'$. We now apply the contraction condition to prove $F\, l \overset{j}{\approx} F\, l'$, which induces three subgoals, for a given $k \prec j$. First, we justify the subgoal $l \overset{k}{\approx} l'$ by transitivity on $v\, k \overset{k}{\approx} l$ (using $k \prec j$) and $v\, k \overset{k}{\approx} l'$ (using $k \prec j \prec i$). Second, we establish $Q\, k\, l$ by continuity of Q, proving the premise $\forall k' \prec k.\, Q\, k'\, (u\, k')$ using the third hypothesis of the lemma and the fact that $\forall k' \prec k.\, v\, k' \overset{k'}{\approx} l$. Third, we establish $Q\, k\, l'$ by continuity of Q, proving the premise $\forall k' \prec k.\, Q\, k'\, (u\, k')$ using the third hypothesis of the lemma and the fact that $\forall k' \prec k.\, v\, k' \overset{k'}{\approx} l'$. □

Lemma 7 (Induction to establish local coherence and the invariant)

$$\forall i.\ \textsf{\textit{locally\_coherent}}\, u\, i \ \wedge\ Q\, i\, (u\, i)$$

Proof By well-founded induction on i. For local-coherence, we need to prove $u\, k \overset{k}{\approx} u\, i$ under the assumption $k \prec j \prec i$. By Lemma 6, it suffices to check the local-coherence of u at level j and at level k (both true by the first part of the induction hypothesis) and the proposition $\forall k \prec i.\ Q\, k\, (u\, k)$, which holds by the second part of the induction hypothesis.

It remains to prove $Q\, i\, (u\, i)$. By definition of u, we need to show $Q\, i\, (F(\mathsf{lim}_i\, u))$. Since we have just shown u coherent at level i, we can invoke Lemma 1 to replace the goal with $Q\, i\, (F\, l)$, with the assumption that $\forall j \prec i.\ u\ j \overset{j}{\approx} l$. We now apply lemma Lemma 5, which leaves the goal $\forall j \prec i.\, Q\, j\, l$. For a given j, we can show $Q\, j\, l$ by continuity of Q. To that end, we first check $\forall k \prec j.\, Q\, k\, (u\, k)$, using the induction hypothesis, and then check $\forall k \prec j.\, u\, k \overset{k}{\approx} l$, which comes from the definition of l and the transitivity $k \prec j \prec i$. □

Lemma 8 (Global coherence) *$\textsf{\textit{globally\_coherent}}\, u$ holds, that is,*

$$\forall j\, k.\ k \prec j \prec i \quad \Rightarrow \quad u\, k \overset{k}{\approx} u\, j$$

Proof Apply Lemma 6 and use Lemma 7 to justify the three premises, which are "$\textsf{locally\_coherent}\, u\, i$" and "$\textsf{locally\_coherent}\, u\, j$" and "$\forall k \prec i.\, Q\, k\, (u\, k)$". □

Lemma 9 (Elimination of the continuity of Q)

$$(\forall j \prec i.\, u\, j \overset{j}{\approx} l) \Rightarrow Q\, i\, l \qquad \textsf{\textit{and similarly}} \qquad (\forall j.\ u\, j \overset{j}{\approx} l) \Rightarrow Q\, i\, l$$

Proof The statements are immediate consequences of the continuity of Q and of the result shown in Lemma 7, asserting that $Q\, j\, (u\, j)$ holds for any j. □

Lemma 10 (Fixed point equation for the limit)
The limit l defined as $l \triangleq \textsf{\textit{lim}}\, u$ is a fixed point of F modulo $\approx$.

Proof Remark: the limit l exists because u is globally-coherent, as established by Lemma 8. Thus, $\forall i.\ u\,i \overset{i}{\approx} l$. Let l' be such that $l' \approx l$. The goal is to show $l' \approx F\,l'$, which amounts to proving $l' \overset{i}{\approx} F\,l'$ for an arbitrary index i. We know that $l' \overset{i}{\approx} l$ and that $l \overset{i}{\approx} u\,i$. Furthermore, $u\,i$ is equal to $F(\mathsf{lim}_i\,u)$. Thus, it suffices to prove the equation $F(\mathsf{lim}_i\,u) \overset{i}{\approx} F\,l'$. As u is locally-coherent up to index i (Lemma 7), the limit l'' of $\mathsf{lim}_i\,u$ exists, so the value l'' is such that $\forall j \prec i.\, u\,j \overset{j}{\approx} l''$. We apply the contraction condition to the remaining goal $F\,l'' \overset{i}{\approx} F\,l'$, which produces three subgoals.

First subgoal is $\forall j \prec i.\ l'' \overset{j}{\approx} l'$. It is justified by transitivity: $l'' \overset{j}{\approx} u\,j \overset{j}{\approx} l \overset{j}{\approx} l'$. Second subgoal is $Q\,j l''$. We prove it by continuity, using Lemma 9 and checking $\forall k \prec j.\, u\,k \overset{k}{\approx} l''$. Third subgoal is $Q\,j l'$. We also prove it by continuity, using Lemma 9 and establishing $\forall k \prec j.\, u\,k \overset{k}{\approx} l'$ by transitivity: $u\,k \overset{k}{\approx} l \overset{k}{\approx} l'$. □

Lemma 11 (Uniqueness of the limit)
The limit l defined in Lemma 10 is the unique fixed point of F modulo $\approx$.

Proof Let l' be another fixed point, that is, a value such that $\forall l'' \approx l'.\ l'' \approx F\,l''$. Our goal is to show $l' \approx l$. We prove by induction on i that $l' \overset{i}{\approx} l$. We know that l and l' are fixed point, thus $l' \overset{i}{\approx} F\,l$ and $l' \overset{i}{\approx} F\,l'$. By transitivity, the goal can be changed to $F\,l' \overset{i}{\approx} F\,l$. We apply the contraction condition and need to prove $\forall j \prec i.\, l \overset{j}{\approx} l' \wedge Q\,j\,l \wedge Q\,j\,l'$. Given a value j, the fact $l \overset{j}{\approx} l'$ is can be directly deduced from the induction hypothesis. The property $Q\,j\,l$ follows from Lemma 9, justifying $\forall k \prec j.\, v\,k \overset{k}{\approx} l$ by the definition of l. The property $Q\,j\,l'$ also follows from Lemma 9, justifying $\forall k \prec j.\, v\,k \overset{k}{\approx} l'$ by transitivity: $v\,k \overset{k}{\approx} l \overset{k}{\approx} l'$. □

Lemma 12 (Invariant satisfied by the limit)
The limit l defined in Lemma 10 is such that "$Q\,i\,l$" holds for any i.

Proof Exploit the continuity of Q, using Lemma 9, checking that $\forall j.\ u\,j \overset{j}{\approx} l$. □

Theorem 22 (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.*

Proof The limit l defined in Lemma 10 is a fixed point of F modulo $\approx$, it is the unique such fixed point as shown in Lemma 11, moreover it satisfies the predicate $Q\,i$ for any i as proved in Lemma 12. □

Appendix G: Fixed point theorem for recursive functions

Theorem 23 (C.o.f.e. for recursive functions) *Let $\equiv$ be an equivalence relation of type $A \to A \to$ Prop, let $<$ be a well-founded relation of type $A \to A \to$ Prop, and let D be a domain of type $A \to$ 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.*

Proof It is immediate to check that $<^+$ is a transitive well-founded relation and that $\overset{x}{\approx}$ is an equivalence relation on the set of functions of type $A \to B$. It remains to check the completeness of the ordered family of equivalences $(A \to B, A, <^+, \overset{x}{\approx})$. Let K be a downward-closed domain (of type $A \to \mathsf{Prop}$), and let u be a sequence (of type $A \to (A \to B) \to \mathsf{Prop}$) coherent on the domain K. By definition of coherence, this means:

$$(\forall x\, y \in K.\, y <^+ x \Rightarrow u\, y \overset{y}{\approx} u\, x) \;\Rightarrow\; \exists l.\, \forall x \in K.\, u\, x \overset{x}{\approx} l$$

Unfolding the definition of $\overset{x}{\approx}$, and using the fact that K is downward-closed with respect to $<^+$ (thus, we know that $x \in K$ and $z <^* x$ imply $z \in K$), the above proposition becomes:

$$\begin{aligned}&(\forall x\, y\, z \in K.\, z <^* y <^+ x \wedge D\, z \;\Rightarrow\; u\, y\, z \equiv u\, x\, z)\\ &\quad\Rightarrow\; \exists l.\; \forall x\, z \in K.\, z <^* x \wedge D\, z \;\Rightarrow\; u\, x\, z \equiv l\, z\end{aligned}$$

To prove the statement, we instantiate l as $\lambda x.\, u\, x\, x$. Let $z \in K$ be a value such that $z <^* x$ and $D\, z$. Our goal is to show $u\, x\, z \equiv (\lambda x.\, u\, x\, x)\, z$, which is equivalent to $u\, x\, z \equiv u\, z\, z$. There are two cases. If $x = z$, then the goal is trivial. Otherwise, $z <^+ x$. In this case, we apply the assumption with $y = z$, and get $u\, z\, z \equiv u\, x\, z$, which allows us to conclude.

Theorem 24 (Fixed point theorem for recursive functions) *Let* $\equiv$ *be an equivalence relation and* $<$ *be a well-founded relation on a type* A*. Let* D *be a domain of type* $A \to \mathsf{Prop}$*, and* 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. Let* F *be a functional of type* $(A \to B) \to (A \to B)$ *that satisfies the following contraction condition:*

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

Then, F *admits a unique partial fixed point* f *modulo* $\equiv_D$*, where "*$f_1 \equiv_D f_2$*" stands for the proposition* $\forall x.\, D\, x \;\Rightarrow\; f_1\, x \equiv f_2\, x$*. Moreover, this fixed point* f *satisfies the post-condition* S *on the domain* D*, that is,* $\forall x.\, D\, x \Rightarrow S\, x\, (f\, x)$*.*

Proof The idea is to construct the partial fixed point of F using the general fixed point theorem for c.o.f.e.'s (Theorem 22) applied to the c.o.f.e. for recursive functions (Theorem 5).

Before we can apply this theorem is to define an invariant Q of type $A \to (A \to B) \to \mathsf{Prop}$ in terms of D and S, So, we let $Q\, x\, f \triangleq \forall y <^* x.\, D\, y \Rightarrow S\, y\, (f\, y)$. We also need to establish the continuity of Q. To that end, let K be a downward-closed domain, f be a value of type A and u be a sequence of functions indexed with values of type A. Suppose that $\forall x \in K.\, u\, x \overset{x}{\approx} f$ and that $\forall x \in K.\, Q\, x\, (u\, x)$ both hold. Our goal is to prove $\forall x \in K.\, Q\, x\, f$. By definition of Q and downward-closedness of K, we need to prove $\forall x, y \in K.\, y <^* x \Rightarrow D\, y \Rightarrow S\, y\, (f\, y)$. Given x and y in K with $D\, y$, we can prove $S\,y\,(f\,y)$. Indeed, the first assumption $u\,x \overset{x}{\approx} f$ gives us $u\,x\,y \equiv f\,y$, and the second assumption $Q\,x\,(u\,x)$ gives us $S\,y\,(u\,x\,y)$. The conclusion $S\,y\,(f\,y)$ follows from the compatibility of S with respect to $\equiv$.

We can now apply the fixed point theorem for c.o.f.e.'s. This gives us a function f which is a fixed point of F modulo $\approx$. This means that for any f' such that $f' \approx f$, the equality $f' \approx F\,f'$ holds. Moreover, the fixed point f is such that $\forall x.\, Q\,x\,(f\,x)$. Our goal is now to prove that (f, D) is a partial fixed point of F modulo $\equiv$.

To start with, let us establish that $\approx$ and $\equiv_D$ both describe the same binary relation, as follows. On the one hand, $f_1 \approx f_2$ holds if $\forall x. \forall y <^* x.\, D\,y \Rightarrow f_1\,y \equiv f_2\,y$. On the other hand, $f_1 \equiv_D f_2$ holds if $\forall y.\, D\,y \Rightarrow f_1\,y \equiv f_2\,y$. The two propositions are cleary equivalent, hence $\approx$ and $\equiv_D$ are two equivalent relations. It follows that for any f' such that $f' \equiv_D f$, the equality $f' \equiv_D F\,f'$ holds. Thus, (f, D) is a partial fixed point of F modulo $\equiv$.

The second conclusion of the fixed point theorem, which is $\forall x.\, Q\,x\,(f\,x)$, ensures that f admits S as post-condition. Indeed, by definition of Q, we have $\forall x. \forall y <^* x.\, D\,y \Rightarrow S\,y\,(f\,y)$, which implies $\forall y.\, D\,y \Rightarrow S\,y\,(f\,y)$.

Theorem 25 (General consistency for recursive fixed points) *Under the hypotheses of the previous theorem, the partial function (f, D) is a generally-consistent fixed point of the functional "$\mathsf{partialize}\,F$".*

Proof The proof of this result is adapted from the proof that inductive fixed points are generally-consistent, developed by Krstić [16].

Let (f', D') be another partial fixed point of F. Our goal is to show that (f, D) and (f', D') are consistent, that is, $\forall x.\, D\,x \wedge D'\,x \Rightarrow f\,x \equiv f'\,x$. Let f'' be the following function:

$$f''\,x \quad \triangleq \quad \mathsf{if}\ D'\,x\ \mathsf{then}\ f'\,x\ \mathsf{else}\ f\,x$$

First of all, observe that $f'' \equiv_{D'} f'$, meaning that for any x in the domain D', we have $f''\,x \equiv f'\,x$.

Using this observation, it is clear that the proposition $\forall x.\, D\,x \Rightarrow f\,x \equiv f''\,x$ is a sufficient condition to prove the goal $\forall x.\, D\,x \wedge D'\,x \Rightarrow f\,x \equiv f'\,x$. The proof of $\forall x.\, D\,x \Rightarrow f\,x \equiv f''\,x$ goes by well-founded induction on x. Let x be a value such that $D\,x$. Our goal is $f\,x \equiv f''\,x$. There are two cases. If $D'\,x$ does not hold, then, by definition of f'', we have $f''\,x \equiv f\,x$, so we are done. Otherwise, we can assume $D'\,x$ to hold to prove our goal $f\,x \equiv f''\,x$.

Because (f, D) is a partial fixed point of F and x belongs to D, we have $g\,x \equiv F\,g\,x$ for any g such that $g \equiv_D f$. In particular, we have $f\,x = F\,f\,x$, because $f \equiv_D f$ is trivially true. Because (f', D') is a partial fixed point of F and x belongs to D', we have $g\,x \equiv F\,g\,x$ for any g such that $g \equiv_{D'} f'$. In particular, we can derive $f''\,x = F\,f''\,x$, because $f'' \equiv_{D'} f'$ was established earlier on. Thus, by transitivity, the goal $f\,x \equiv f''\,x$ is equivalent to $F\,f\,x \equiv F\,f''\,x$.

We prove this latter equality using the contraction condition for F. The first premise to be verified is $\forall y < x.\, D\,y \Rightarrow f\,y \equiv f''\,y$. It corresponds exactly the induction hypothesis that we have. The second premise to be verified is $\forall y < x.\, D\,y \Rightarrow S\,y\,(f\,y)$. The property $S\,y\,(f\,y)$ was already obtained from the fixed point theorem for recursive functions.

In conclusion, (f, D) is a partial fixed point of F such that (f', D') are consistent for any (f', D') partial fixed point of F, thus (f, D) is a generally-consistent partial fixed point of F.

A similar development can be conducted for the case of mixed recursive-corecursive functions.