The Optimal Fixed Point Combinator

Arthur Charguéraud
INRIA

Interactive Theorem Proving (ITP'10) Edinburgh, 2010/07/14
Example: filter function for streams

Step 1: write a functional, e.g. for filter on streams

```coq
Definition Filter filter s :=
  let (x:::t) := s in
  if (P x) then (x ::: filter t) else (filter t).

// "filter" is a partial function mixing recursion and co-recursion
```

Step 2: construct its fixed point (non-constructively)

```coq
Definition filter := Fix Filter.  // return type inhabited
Definition filter := FixModulo (≈) Filter.  // actual
```

Step 3: prove a fixed point equation

```coq
Lemma filter_fix : forall s, infinitely_many P s ->
  filter s ≈ Filter filter s.
```

Step 4: use that equation to unfold the definition

```coq
filter (x:::t)  // rewrite filter_fix
≈ Filter filter (x:::t)  // unfold Filter
≈ if (P x) then (x:::filter t) else (filter t)
```
Examples of recursive functions

Basic recursive function:

Definition Log log x :=
    if x <= 1 then 0 else 1 + log (x/2).

Definition log := FixModulo (=) Log.
Definition log := Fix Log. // equivalent to the line above

Nested recursion, e.g. the nested zero function:

Definition F f x =
    if x = 0 then 0 else f(f(x-1)).

    // need to justify that f(x-1) is smaller than x

Higher-order recursion, e.g. a function modifying trees:

type tree = Leaf of nat | Node of list tree
Definition Incr incr x := match x with
    | Leaf n => Leaf (n+1)
    | Node xs => Node (List.map incr xs)

    // need to justify that "incr" is applied to smaller trees
Examples of co-recursive values

Definition of co-recursive values:

Definition $F\ s := 0 :::\ map\ succ\ s$.

Definition $s := \text{FixValModulo}\ (\approx)\ F.\ // 0:::1:::2:::3:::....$

Lemma $s\_\text{fix} : s \approx F\ s$.

A trickier definition:

Definition $F\ s := 2 :::\ filter\ (\geq 1)\ s$.

$// F$ defines the stream "2:::2:::2:::....", because $2 \geq 1$.

An invalid definition:

Definition $F\ s := 0 :::\ filter\ (\geq 1)\ s$.

$// This\ functional\ does\ not\ admit\ a\ fixed\ point$

Definition $s := \text{FixValModulo}\ (\approx)\ F.$

$// The\ stream\ s\ is\ unspecified$
Program extraction is possible

The fixed point combinators are not constructive. They rely on Hilbert's epsilon operator, which does not have any computational equivalent.

Extraction towards a "let-rec" is possible:

\[
\text{Extract Constant Fix } \Rightarrow \\
\text{"(\text{bigf} \rightarrow \text{let } x = \text{bigf } x \text{ in } x)".} \quad \text{// Haskell code}
\]

→ Partial correctness of the extracted code is to be expected (although I have not proved it formally)

→ Same trick used, e.g., by Bertot et al (2002)
Main fixed point approaches

- **Well-founded recursion**: for partial functions, the domain needs to appear explicitly.

- **Domain-predicate recursion** (Dubois & Donzéau-Gouge, Bove & Capretta) and **inductive graph predicate** (Krauss): works for recursion but does not seem to extend to co-recursion.

- **Co-recursion with guard conditions**: definitions need to be modified so as to satisfy guard conditions either syntactic or type-based (e.g., work by Bertot and others), but such tricks are not always possible.

- **Contraction conditions**: allow proving the existence of a unique fixed point on a given domain, but does not help in constructing partial fixed point.
Ingredients and contribution

The combinator is built upon two ingredients:

1) Optimal fixed points
   → First formalization of optimal fixed point theory
   → First fixed point library using optimal fixed points

2) Contraction conditions
   → Generalization of contr. conditions for co-recursion
   → Unification of the various contraction conditions
Optimal fixed points

Consider the combinator for total recursive function:

\[
\text{Definition Fix } F := \\
\epsilon f. (\forall x, f \ x = F \ f \ x).
\]

It generalizes to partial functions with something like:

\[
\text{Definition Fix } D \ F := \\
\epsilon f. (\forall x, D \ x \rightarrow f \ x = F \ f \ x).
\]

However, the domain must be provided explicitly.

**Question:** is there a best possible domain \( D \) that can be deduced from the functional \( F \) alone?

**Positive answer** [Manna and Shamir, 1975]:

Any functional admits an *optimal* fixed point.
Domains of fixed points

The union of the domains of all the fixed points might not be the domain of a fixed point:

→ This generally happens with inconsistent fixed points
Domain of the optimal fixed point

The restriction to the set of arguments for which all fixed points return the same results:

\[ f_2 \ x = f_3 \ x \]

\[ f_2 \ x \neq f_3 \ x \]

→ This domain admits exactly one fixed point, which captures the maximal amount of non-ambiguous information contained in the functional.
Optimal fixed point combinator

The optimal fixed point of a functional \( F \) is the largest generally-consistent fixed point of \( F \).

(A fixed point of \( F \) is generally-consistent if it does not disagree with any other fixed point of \( F \)).

\[
\text{Definition Fix } A \ B \ (F : (A \to B) \to (A \to B)) : A \to B := \\
\quad \varepsilon f. \ (\text{optimal_fixed_point_of } F \ f).
\]

// Remark: the type \( B \) is required to be inhabited.

// Partial functions are represented in the logic as pairs of type \((A \to \text{Prop}) \to (A \to B)\). The optimal fixed point returned by the combinator Fix is undefined outside of the optimal domain.

Another construction (Gonthier, 2005)

\[
\text{Definition Fix } A \ B \ F := \text{fun } x => \\
\quad \text{let } f := \varepsilon f. (\exists D. \text{fixed_point_on } D \ F \ f \land x \in D) \text{ in } (f \ x).
\]
Contraction conditions

A contraction condition is a sufficient condition for a functional to admit a unique fixed point, expressing the fact that the functional *brings its arguments closer*.

- Guarantees unique fixed point in Banach spaces.
  \[ \| F(x) - F(y) \| \leq \alpha \cdot \| x - y \| \quad \text{with } \alpha < 1 \]

- Paulson (1992): implement the theory of inductive definitions in Isabelle/HOL.


Fixed point theorems

How to use contraction conditions to reason on results of the optimal fixed point combinator:

1) Given a functional $F$, build $f := \text{Fix } F$.

2) Prove that $F$ satisfies a contraction condition on some domain $D$.

3) Deduce that $f$ satisfies the fixed point equation on $D$.

Theorem Fix_spec : forall $F$ $D$ $f$, $f = \text{Fix } F \rightarrow \text{contractive_on } D F \rightarrow$forall $x$, $D x \rightarrow f x = F f x$. 
What's next

Application of the optimal fixed point combinator using existing contraction conditions:

– Total recursion
– Partial function
– Nested recursion
– Co-recursive values
– Co-recursive functions
– Mixed rec./co-recursive

(Supported but not presented: higher-order recursion)

Generalization and unification of the various contraction conditions:

– Generalization of the contraction condition
– Presentation of the unifying fixed point theorem
Treatment of total functions

Fixed point theorem for total recursive functions:

Lemma Fix_spec : forall f F R, well_founded R ->
  f = Fix F ->
  (forall f1 f2 x,
    (forall y, R y x -> f1 y = f2 y) ->
    F f1 x = F f2 x) ->
  (forall x, f x = F f x).

Illustration with the functional Log:

Hypothesis: forall y, y < x -> f1 y = f2 y

Goal: Log f1 x = Log f2 x

Goal: (if x <= 1 then 0 else 1 + f1(x/2))
  = (if x <= 1 then 0 else 1 + f2(x/2))

Subgoal: x <= 1  |-  0 = 0
Subgoal: x > 1  |-  1 + f1(x/2) = 1 + f2(x/2)

Apply the hypothesis to y = x/2, and check (x/2) < x
Treatment of partial functions

Restriction to arguments from a domain D:

Lemma Fix_spec : forall f F R D, well_founded R ->
  f = Fix F ->
  (forall f1 f2 x, D x ->
   (forall y, D y -> R y x -> f1 y = f2 y) ->
   F f1 x = F f2 x) ->
  (forall x, D x -> f x = F f x).

→ The argument $x$ is assumed to be in the domain $D$.
→ Recursive calls must be made to values $y$ inside $D$.
→ The fixed point equation is available only on $D$. 
Treatment of nested recursion

The basic contraction condition does not suffice. Consider for example the nested zero function:

\[
\text{Definition } F \ f \ x = \\
\quad \text{if } x = 0 \text{ then } 0 \text{ else } f(f(x-1)).
\]

→ For the outer recursive call \( f(f(x-1)) \), we need to know that the argument \( f(x-1) \) is smaller than \( x \).

→ We need to know that the function \( f \) returns zero.

Adding an invariant [Matthews & Krstić, 2003]:

\[
\text{Lemma Fix_spec : forall } f \ F \ R \ Q, \text{ well_founded } R \rightarrow \\
\quad f = \text{Fix } F \rightarrow \\
\quad (\forall f1 \ f2 \ x, \\
\quad \quad (\forall y, y < x \rightarrow f1 \ y = f2 \ y \land Q \ y \ (f1 \ y)) \rightarrow \\
\quad \quad F \ f1 \ x = F \ f2 \ x \land Q \ x \ (F \ f1 \ x)) \rightarrow \\
\quad (\forall x, f \ x = F \ f \ x \land Q \ x \ (f \ x)).
\]
Treatment of co-recursive values

Example:

Definition $F\ s := 0 :::\ map\ succ\ s$. // $0:::1:::2:::3:::...$

Definition $s := \text{FixValModulo } (\approx) \ F$.

Lemma $s\_\text{fix} : s \approx F\ s$.

Fixed point combinator for values:

$\Rightarrow \text{FixValModulo } (\approx) \ F$ picks a fixed point of $F$ modulo $\approx$

The insufficient, naive definition:

Definition $\text{FixValModulo } (\approx) \ F := \\
\epsilon x. (x \approx F\ x)$.

The appropriate, standard definition:

Definition $\text{FixValModulo } (\approx) \ F := \\
\epsilon x. (\forall y, y \approx x \rightarrow y \approx F\ y)$.
Contraction condition for streams

The contraction condition [Matthews, 1999]:

\[
\text{forall } i \ s1 \ s2, \ s1 \approx_i s2 \rightarrow F \ x1 \approx_{i+1} F \ s2
\]

implies the existence of a unique fixed point \(s\) modulo bisimilarity, where \((\approx_i)\) relates two streams that are identical up to their \(i\)-th element.

Illustration with the stream of natural numbers:

Hypothesis: \(s1 \approx_i s2\)

Goal: \(F \ s1 \approx_{i+1} F \ s2\)

Goal: \(0 ::\ map \ succ \ s1 \approx_{i+1} 0 ::\ map \ succ \ s2\)

Goal: \(map \ succ \ s1 \approx_i map \ succ \ s2\)

Exploit the fact that an application of map preserves the degree of similarity between two streams.
General presentation of c.o.f.e.'s


The contraction condition

\[
\text{forall } i \ x_1 \ x_2, \\
(\text{forall } j < i, \ x_1 \approx_j x_2) \rightarrow \\
F \ x_1 \approx_i F \ x_2
\]

ensures the existence of a unique fixed point \( x \) of \( F \) modulo \( \approx \), where:

- \( F \) has type \( A \rightarrow A \)
- \( I \) is a type with a transitive well-founded relation \(<\)
- \( \approx \) is the intersection of the equivalence relations \( \approx_i \)
- \( (\approx_i)_{i:I} \) needs to be a complete family of relations
Treatment of co-recursive functions

The contraction condition for co-recursive functions given by Matthews (1999) leads to the following fixed point theorem for co-recursive functions:

Lemma FixModulo_spec : forall F f (≈_i)_i∈I,
   f = FixModulo (≈) F -> cofe (≈_i)_i∈I ->
   (forall f1 f2 x i,
     (forall j<i, forall y, f1 y ≈_j f2 y) ->
     F f1 x ≈_i F f2 x) ->
   forall x, f x ≈ F f x.
Matthews (1999) also showed how to derive the **fixed point theorem for mixed rec/corec functions**:

**Lemma FixModuloLexico_spec** : \( \forall (\approx_i)_{i \in I} F \ f \ D, \)
\( f = \text{FixModulo} (\approx) F \rightarrow \text{cofe} (\approx_i)_{i \in I} \rightarrow \)
\( (\forall f1 \ f2 \ x \ i, \ D \ x \rightarrow \)
\( (\forall y \ j, \ (j,y) < (i,x) \rightarrow D \ y \rightarrow f1 \ y \approx_j f2 \ y) \rightarrow \)
\( F \ f1 \ x \approx_i F \ f2 \ x) \rightarrow \)
\( \forall x, \ D \ x \rightarrow f \ x \approx F \ f \ x. \)

**Illustration with the filter function:**

\( (j,y) < (i,x) \) is a lexicographical comparison.
\( i \) decreases when the head value satisfies P.
\( x \) decreases when the next element satisfying P gets closer.
Co-recursion with an invariant

The tricky co-recursive definition:

Definition $F \ s := 2 :: \ldots \text{filter} (\geq 1) \ s$.

New generalized form of contraction conditions:

forall $x_1 \ x_2 \ i,$

\[ x_1 \approx_{i} x_2 \land Q \ i \ x_1 \land Q \ i \ x_2 \rightarrow \]

\[ F \ x_1 \approx_{i+1} F \ x_2 \land Q \ (i+1) \ (F \ x_1) \]

Illustration: it suffices to consider an invariant stating that the elements before index $i$ are greater than 1:

Definition $Q \ i \ s := (\forall j<i, \ \text{nth} \ j \ s \geq 1)$.

Side-condition: the invariant $Q$ has to be continuous.

Here, we need to show that if $Q \ i \ s$ holds for any $i$, then $s$ contains only values greater than 1.
Key idea about invariants

Recursive definition → specify results
post-condition $Q \times (f \times)$

Co-recursive definition → specify prefixes
invariant $Q \times i \times s$
The unifying fixed point theorem

If the following hypotheses hold

- \( F \) is a functional of type \( A \to A \) (where \( A \) is inhabited)
- \((A, I, <, \approx_i)\) is a c.o.f.e.
- \( Q \) is a continuous property of type \( I \to A \to \text{Prop} \)
- The following contraction condition holds

\[
\forall i \; x_1 \; x_2, \\
(\forall j<i, \; x_1 \approx_j x_2 \land Q j \; x_1 \land Q j \; x_2) \to \\
F \; x_1 \approx_i F \; x_2 \land Q i (F \; x_1)
\]

Then we can deduce that

- \( F \) admits a unique fixed point \( x \) modulo \( \approx \)
- Moreover \( x \) satisfies the invariant: \( \forall i, Q i \; x \)
Several examples formalized

<table>
<thead>
<tr>
<th>Recursion:</th>
<th>Lines of proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>log function</td>
<td>2</td>
</tr>
<tr>
<td>gcd function</td>
<td>3</td>
</tr>
<tr>
<td>div function</td>
<td>3</td>
</tr>
<tr>
<td>nested zero function</td>
<td>3</td>
</tr>
<tr>
<td>trees with lists of subtrees</td>
<td>4</td>
</tr>
<tr>
<td>Ackermann's function</td>
<td>3</td>
</tr>
<tr>
<td>McCarthy's function</td>
<td>8</td>
</tr>
</tbody>
</table>

**Co-recursion:** (≈ 100 lines to establish a new c.o.f.e.)

<table>
<thead>
<tr>
<th>Co-recursion:</th>
<th>Lines of proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant stream</td>
<td>3</td>
</tr>
<tr>
<td>mutually-defined streams</td>
<td>9</td>
</tr>
<tr>
<td>filter on streams</td>
<td>13</td>
</tr>
<tr>
<td>&quot;product&quot; of infinite trees</td>
<td>24</td>
</tr>
</tbody>
</table>
Conclusion

1) Optimal fixed points:
- for long, a curiosity about circular *program* definitions
- the tool of choice to justify circular *logical* definitions
- allows to separate definitions from their justification

2) Contraction conditions:
- well-foundedness and productivity inside the logic
- support for a very large scope of circular definitions
- all contraction conditions derivable from a single one

\[(1) + (2) = \text{Fix } F\]
Thanks!

Extended version of the paper available from:
http://arthur.chargueraud.org/research/2010/fix