Algorithmic aspects of least and greatest fixed points

Ulrich Berger (Swansea University)

j.w.w. Hideki Tsuiki (Kyoto University)

and Olga Petrovska (Swansea University)

Algebra and Algorithms
Djerba, Tunisia, 4-6 February 2020

Overview

- Fixed points in classical and constructive mathematics
- The computational content of fixed points
- Examples
- Program extraction with fixed points in PRAWF.

Varieties of mathematics

Classical mathematics Concerned with concrete and abstract structures. Proofs establish true statements about them.

Constructive mathematics Same as above but requiring constructive evidence, hence rejecting $A \vee \neg A$ as an axiom scheme.

Constructive Type Theory Concerned with types and their inhabitants which are formally derived following constructive principles. Proofs are inhabitants of types, truth is a derived notion (being inhabited), computing with proofs is essential.

Classical vs. constructive mathematics

Both have a big overlap regarding ontology and logic. In (Bishop-style) constructive mathematics it is a widely accepted principle not to contradict classical logic.

Via negative translation, classical logic can be even viewed as a subsystem of constructive logic.

The situation is less clear for mathematical principles such as:

- choice principles
- fixed point principles

While the former can be largely avoided in constructive mathematics the latter are an essential part of both classical and constructive mathematics.

Do fixed points exist?

For a strictly positive operator $\Phi = \lambda X\, \lambda\vec{y}\, A(X, \vec{y})$, where $A(X, \vec{y})$ is a formula in first-order logic with a free predicate variable X in strictly positive position and first-order variables $\vec{y}$, the least and greatest fixed points, $\mu(\Phi)$ and $\nu(\Phi)$, exist and can be defined (impredicatively in second-order logic) as

$$\mu(\Phi) = \bigcap_{\Phi(Y)\subseteq Y} Y$$
$$\nu(\Phi) = \bigcup_{Y\subseteq \Phi(Y)} Y$$

Can the existence of these fixed points justified, constructively?

For many instances, the answer is 'yes'. But in general?

Program extraction in IFP

IFP (Intuitionistic Fixed Point logic) is an intuitionistic first-order theory that permits the construction of least and greatest fixed points of strictly positive operators Φ.

From every IFP proof one can extract a program *realizing* the proven formula.

This provides a computational (if not constructive) justification of the existence of these fixed points.

Program extraction in IFP ctd.

The extracted programs are *garbage free* in the sense that no program is extracted from sub-proofs of *Harrop formulas*, that is, formulas without occurrences of disjunctions at a strictly positive position (existential quantification is unrestricted)

IFP permits interpretations by structures that are not necessarily constructively given, i.p. *decidability of equality is not assumed*.

IFP admits *arbitrary disjunction-free (true) axioms*.

Program extraction for induction (minimality of $\mu(\Phi)$)

$$\frac{s : \Phi(\mathcal{P}) \subseteq \mathcal{P}}{\mathbf{rec}\,\lambda f\,(s \circ (m_\Phi\, f)) : \mu(\Phi) \subseteq \mathcal{P}}\ \mathit{ind}$$

where m_Φ realizes $X \subseteq Y \to \Phi(X) \subseteq \Phi(Y)$.

That is, the extracted program is a function f recursively defined by

$$f\ a = s\,(m_\Phi\ f\ a)$$

Program extraction for coinduction (maximality of $\nu(\Phi)$)

$$\frac{s : \mathcal{P} \subseteq \Phi(\mathcal{P})}{\mathbf{rec}\,\lambda f\,((m_\Phi\, f) \circ s) : \mathcal{P} \subseteq \nu(\Phi)}\ \mathit{coind}$$

That is, the extracted program is a function f recursively defined by

$$f\ a = m_\Phi\ f\ (s\ a)$$

Program extraction for the fixed point property

$$\frac{}{\lambda a\, a : \Phi(\Box(\Phi)) \subseteq \Box(\Phi)}\ \textit{closure}$$

$$\frac{}{\lambda a\, a : \Box(\Phi) \subseteq \Phi(\Box(\Phi))}\ \textit{coclosure}$$

$$(\Box \in \{\mu, \nu\})$$

Program extraction for the quantifier rules

$$\frac{p : A(x)}{p : \forall x\, A(x)}\,\forall^+ \ (*) \qquad \frac{p : \forall x\, A(x)}{p : A(t)}\,\forall^-$$

$$\frac{p : A(t)}{p : \exists x\, A(x)}\,\exists^+ \qquad \frac{p : \exists x\, A(x) \qquad q : \forall x\,(A(x) \to B)}{p\,q : B}\,\exists^- \ (**)$$

This means that first-order variables are treated uniformly ensuring that program extraction is valid for arbitrary structures.

Soundness Theorem

From a proof of a formula A in IFP one can extract a program p realizing A.

The proof that p realizes A can be carried out in an extension of IFP, called RIFP, that axiomatizes the behaviour of programs.

RIFP proofs are constructively and classically valid.

Related systems supporting program extraction

MINLOG
(`http://www.mathematik.uni-muenchen.de/∼minlog`)
supports a similar style of program extraction. In fact, IFP can be viewed as a branch of MINLOG exploring the operator view of least and greatest fixed points and aiming for full compatibility with classical mathematics, and where inductive and coinductive definitions and their extracted programs are dual to each other.

Nuprl, *Coq* and *Agda* (`http://www.nuprl.org/`,
`http://coq.inria.fr/`,
`http://wiki.portal.chalmers.se/agda/`) are type theoretic systems supporting inductive and coinductive definitions, however in a more restricted form and with a different ontology where coinductive definitions correspond to non-wellfounded proofs and their extracted programs to guarded recursion with a special operational semantics.

Natural numbers

In any structure with a constant 0 and a successor function S one can define

$$\mathbb{N} = \mu(\Phi_{\mathbb{N}})$$

where

$$\Phi_{\mathbb{N}} = \lambda X\, \lambda x\, (x = 0 \vee \exists y\, (x = \mathrm{S}(y) \wedge X(y))).$$

If 0 is not a successor and S is injective (this can be stated by disjunction-free axioms), $\mathbb{N}$ will be isomorphic to the natural numbers.

If in addition there is a binary function $+$ satisfying $x + 0 = x$ and $x + \mathrm{S}(y) = \mathrm{S}(x + y)$, then one can prove that $\mathbb{N}$ is closed under $+$,

$$\mathbb{N}(x) \wedge \mathbb{N}(y) \to \mathbb{N}(x + y),$$

and extract from a proof a program performing addition of natural numbers in unary notation.

Accessible induction

The *accessible part* of a binary relation $\prec$ is defined inductively by

$$\mathbf{Acc}_{\prec} = \mu(\Phi_{\prec}) \text{ where } \Phi_{\prec} = \lambda X\, \lambda x\, \forall y \prec x\, X(y).$$

Since $\Phi(P) \subseteq P$ unfolds to the $\prec$-progressiveness of P,

$$\forall x\, (\forall y \prec xP(y) \to P(x))$$

s.p. induction for $\mathbf{Acc}_{\prec}$ is the familiar induction on the accessible part of $\prec$

$$\frac{P \ \prec\text{-progressive}}{\mathbf{Acc}_{\prec} \subseteq P}\ \text{AccInd}$$

Taking for $\prec$ the element relation $\in$ in axiomatic set theory, AccInd is the foundation axiom in the form of $\in$-induction.

Program extraction for accessible induction

$$\frac{s : (P \prec\text{-progressive})}{\mathbf{rec}\, s : \mathbf{Acc}_{\prec} \subseteq P}\ \text{AccInd}$$

Since $\mathbf{Acc}_{\prec}$ is Harrop, this means:

If s realizes the $\prec$-progressiveness of P, that is,

$$\forall x\, \forall a\, ((\forall y \prec x\, (a\, \mathbf{r}\, P(y))) \to (s\, a)\, \mathbf{r}\, P(x))$$

then p, recursively defined by $p = s\, p$, realizes $\mathbf{Acc}_{\prec} \subseteq P$, that is,

$$\forall x\, (\mathbf{Acc}_{\prec}(x) \to p\, \mathbf{r}\, P(x))$$

Brouwer's Thesis and Wellfounded induction

Elements beginning an infinite $\prec$-descending sequence can be characterized coinductively by

$$\mathbf{Path}_{\prec} = \nu(\lambda X\, \lambda x\, \exists y \prec x\, X(y))$$

$\neg\mathbf{Path}_{\prec}(x)$ and $\mathbf{Acc}_{\prec}(x)$ are classically equivalent
Therefore, we can postulate the Harrop axiom

$$\mathbf{BT}_{\prec} \qquad \forall x\, (\neg\mathbf{Path}_{\prec}(x) \to \mathbf{Acc}_{\prec}(x))$$

which can be viewed as an abstract version of *Brouwer's Thesis* (stating that barred sequences of natural numbers are inductively barred). By $\mathbf{BT}_{\prec}$, the conclusion of accessible induction can be replaced by $\neg\mathbf{Path}_{\prec} \subseteq P$ and is then called *wellfounded induction*.

Bar-induction can be obtained as a special case of wellfounded induction.

Archimedean Induction

In an axiomatic approach to real numbers, setting $y \prec x \stackrel{\mathrm{Def}}{=} y \geq 0 \wedge y + 1 = x$, $\mathbf{Path}_{\prec}(x)$ means that x is 'infinite'.

Therefore, the Archimedean property of real numbers can be stated as $\forall x \, \neg \mathbf{Path}_{\prec}(x)$ and wellfounded induction is equivalent to the rule

$$\frac{\forall x \, ((x \geq 0 \to P(x)) \to P(x+1))}{\forall x \, P(x)} \ \mathrm{AI}(P)$$

which we call *Archimedean Induction*.

It is like induction on natural numbers but for abstract real numbers and without an explicit base case.

Archimedean Induction is useful to avoid countable choice. We we will also use it later in this talk.

Signed digit representation

Let $\mathrm{SD} = \{-1, 0, 1\}$ be the set of signed digits, $\mathbb{I} = [-1, 1]$, and set $\mathrm{av}_d(x) = (x + d)/2$.

Defining

$$\mathbf{S} = \nu(\lambda X\, \lambda x\, \exists d \in \mathrm{SD}\, \exists y \in \mathbb{I}\, (x = \mathrm{av}_d(y) \wedge X(y)))$$

one can characterize the signed digit representation of real numbers in $\mathbb{I}$ in the sense that

$$a\, \mathbf{r}\, \mathbf{S}(x) \text{ iff } a \text{ is a signed digit representation of } x$$

This characterization can be generalized to uniformly continuous functions on $\mathbb{I}$ by a nested inductive/coinductive definition: $\mathbf{S}_1 =$

$$\nu(\lambda X\, \mu(\lambda Y\, \lambda x\, (\exists e \in \mathrm{SD}\, \exists y \in \mathbb{I}^{\mathbb{I}}\, x = \mathrm{av}_e \circ y \wedge X(y)) \vee (\forall d \in \mathrm{SD}\; Y(x \circ \mathrm{av}_d))))$$

Infinite Gray code

Infinite Gray code was introduced independently by Pietro Di Gianantonio and Hideki Tsuiki. It can be characterized coinductively by

$$\mathbf{G} = \nu(\lambda X\, \lambda x\, (x \in \mathbb{I} \land (x \neq 0 \to x \leq 0 \lor x \geq 0) \land \mathbf{G}(1 - 2|x|)))$$

Realizers of $\mathbf{G}(x)$ are streams of binary digits (L,R) that may be undefined at one point.

The signed digit representation and infinite Gray code are both admissible representations of the reals but infinite Gray code is in addition *unique*.

Using Archimedean induction one can show $\mathbf{S} \subseteq \mathbf{G}$ and extract a conversion program.

(Manually) extracted program ($\mathbf{S} \subseteq \mathbf{G}$)

```
stog :: SDrep -> InfGrayCode
stog p = case head p of {
  -1 -> L : stog (tail p) ;
   1 -> R : nh (nall (tail p)) ;
   0 -> let { q = stog (tail p) }
        in head q : R : nh (tail q)
 }

nall (L : q) = R : neg q
nall (R : q) = L : neg q

nh (L : q) = R : q
nh (R : q) = L : q
```

Prawf

Prawf (`https://prawftree.wordpress.com/`) is a prototype implementation in Haskell, which allows users to write IFP proofs and extract executable programs from them.

We illustrate Prawf by showing how to implement some of the previous examples.

Declaring a language

```
<sorts>
R
<end sorts>

<constants>
0,1:R
<end constants>

<functions>
+ : (R,R) -> R;
- : (R,R) -> R;
<end functions>

<predicates>
= : (R,R);
<end predicates>
```

Defining natural numbers and declaring axioms

```
Phi:(R) = lambda Y:(R) lambda (z:R) (z=0 v Y(z-1))
N:(R) = Mu(Phi)

ax1 . All x:R x+0 = x
ax2 . All x:R (All y:R (x+y)-1 = x+(y-1))
```

A proof in PRAWF

```
Enter goal formula> all x:R (N(x) -> all y:R (N(y) -> N(x+y)))

Assumptions:
 v1 : N(x)
Context of the goal:
 variables: x: R
Current goal:
 |-  ?2 : All (y:R) (N(y) -> N(x+y))

Enter command> ind

Assumptions:
 v1 : N(x)
Context of the goal:
 variables: x: R
Current goal:
 |-  ?3 : All (y:R) (Phi(lambda (y:R) N(x+y))(y) -> N(x+y))
```

A proof in PRAWF(ctd.)

```
Enter command> unfold Phi

Assumptions:
 v1 : N(x)
Context of the goal:
 variables: x: R
Current goal:
 |-  ?3 : All (y:R) ((y=0 v N(x+(y-1))) -> N(x+y))
...
No Goal
Proof complete.
```

Program extraction

```
Enter quit, ...> extract
Waiting for program to extract ...
================================
Extracted program (simplified):
================================
Abstpr v1
  Recpr
    Abstpr f_mu
      Abstpr a_comp
        Casepr ProgVar a_comp
           *{Lt ["a_ore"]
               ProgVar v1}
           *{Rt ["b_ore"]
               Conpr Rt
                 Apppr
                   ProgVar f_mu
                   ProgVar b_ore}
Extraction completed.
```

Conclusion

- IFP formalizes constructive mathematics and supports program extraction from proofs.
- Fully compatible with classical logic.
- Extendible by arbitrary (true) disjunction free axioms.
- Prototype implementation PRAWF in Haskell.

Further work:

- Extracting typed programs.
- Adding a modal operator for nondeterminism to extract nondeterministic concurrent programs.

Some references

B, K Miyamoto, H Schwichtenberg, M Seisenberger, Minlog - A Tool for Program Extraction for Supporting Algebra and Coalgebra, LNCS 6859, 2011.

B, From coinductive proofs to exact real arithmetic: theory and applications, Logical Methods in Comput. Sci. 7, 2011,
H Schwichtenberg, S S Wainer, Proofs and Computations, Cambridge University Press, 2012.

H Tsuiki. Real Number Computation through Gray Code Embedding. Theor. Comput. Sci. 284, 2002.

B, A Lawrence, F Nordvall, M Seisenberger. Extracting verified decision procedures: DPLL and Resolution. Logical Methods in Computer Science 11, 2015.

B, O Petrovska. Optimized program extraction for induction and coinduction CiE 2018, LNCS 10936, 2018.

B, H Tsuiki. Intuitionistic fixed point logic. Draft
`arxiv.org/abs/2002.00188`