
FP Seminar, Nottingham

Program extraction from proofs: induction and coinduction

Ulrich Berger

Swansea University

Program extraction?

Thesis:

Program extraction from proofs has a good chance to become an accepted methodology for producing verified software.

Defence:

- ▶ Proofs can be carried out in surprisingly simple formal systems, namely mild extensions of first-order predicate logic.
- ▶ A lot of classical mathematics can be reused, without constructivization.
- ▶ Case studies (medium size) indicate that the method is practical and useful, with and without proof assistant.

(We give examples from real analysis and monadic parsing.)

Overview

- ▶ Mathematical and formal framework
- ▶ From coinduction to exact real number computation
- ▶ From induction to monadic parsers
- ▶ Related work
- ▶ Conclusion

Classical mathematics with constructive topping

Axioms:

Any suitable axiom system of classical mathematics (for example ZFC) in a negative formulation, i.e double-negation translated.

On top of that:

Inductive and coinductive definitions as least and greatest fixed points of strictly positive predicate operators.

Intuitionistic logic.

Program extraction

Realisability with uniform interpretation of quantifiers:

$$\forall = \bigcap \quad \exists = \bigcup$$

A suitable formalisation yields Haskell-like extracted programs.

Paper with M. Seisenberger to appear.

Real and natural numbers

\mathbb{R} = the usual (classical) complete ordered field.

\mathbb{N} = the natural numbers as an inductively defined subset of \mathbb{R} , i.e. the least subset of \mathbb{R} such that

$$\begin{aligned} \mathbb{N} &= \{0\} \cup \{x + 1 \mid x \in \mathbb{N}\} \\ & (= \{x \in \mathbb{R} \mid x = 0 \vee \exists y (y \in \mathbb{N} \wedge x = y + 1)\}) \end{aligned}$$

Here, \vee is *constructive* disjunction.

A first example of program extraction:

From a constructive proof of

$$\forall x, y (x \in \mathbb{N} \wedge y \in \mathbb{N} \rightarrow x + y \in \mathbb{N})$$

one extracts a program computing addition for natural numbers in unary notation.

Approaching real numbers coinductively

$$\mathbb{I} := [-1, 1] \subseteq \mathbb{R}.$$

Define C_0 coinductively as the largest subset of \mathbb{I} such that

$$C_0 = \left\{ \frac{x+d}{2} \mid x \in C_0, d \in \text{SD} \right\}$$

where $\text{SD} = \{0, 1, -1\}$ is the set of signed (binary) digits.

Theorem 1 $x \in C_0$ iff $\forall n \in \mathbb{N} \exists q \in \mathbb{Q} \cap \mathbb{I} |x - q| \leq 2^{-n}$.

This theorem and all results in the following are constructive.

A realiser of $x \in C_0$ is an infinite stream of signed digits $a = a_0 : a_1 : \dots$ representing x , i.e. $x = \sum_i a_i 2^{-(i+1)}$.

From the proof of the lemma one extracts programs translating between the signed-digit- and the Cauchy-representation.

Extracting exact real arithmetic

Theorem 2 If $x, y \in C_0$ then $\frac{x+y}{2} \in C_0$.

Theorem 3 If $x, y \in C_0$ then $xy \in C_0$.

From these theorems one extracts implementations of addition and multiplication w.r.t. the signed digit representation.

Similar implementations were studied by Edalat, Potts, Heckmann, Escardo, Ciaffaglione, Gianantonio, e.t.c.

The difference is that we extract the programs –together with their correctness proofs.

Approaching real functions (co)inductively

$x \in C_0$ roughly means that there is a signed digit stream $a = a_0 : a_1 : \dots$ such that $x = \text{av}_{a_0} \circ \text{av}_{a_1} \circ \dots$ where $\text{av}_d(y) = \frac{d+y}{2}$. The stream a can be viewed as a process that emits the digits a_i .

A (uniformly) continuous function $f \in \mathbb{I}^{\mathbb{I}}$ can be viewed as real number that depends on an input. Therefore it cannot always *emit* digits: occasionally it must *absorb* digits from the input.

This idea is captured by the set C_1 which is defined coinductively as the largest subset of $\mathbb{I}^{\mathbb{I}}$ such that

$$C_1 = \mu F. \{ \text{av}_e \circ g \mid e \in \text{SD}, g \in C_1 \} \cup \{ f \mid \forall d \in \text{SD} f \circ \text{av}_d \in F \}$$

where $\mu F. \Phi(F)$ denotes the least fixed point of Φ , i.e. an inductive definition.

Memo trees (tries?) for continuous functions

Theorem 4 $f \in \mathbb{I}^{\mathbb{I}}$ is continuous iff $f \in C_1$.

From the proof of this theorem one extracts programs translating between realisers of “ f is continuous” (where continuity has to be defined in a constructively meaningful way) and realisers of “ $f \in C_1$ ”.

What is a realiser of “ $f \in C_1$ ”?

It is a finitely branching non-wellfounded tree describing when f emits and absorbs digits. I.p. it is a *data structure*, not a function.

Similar trees have been studied by P. Hancock, D. Pattinson, N. Ghani.

Extracting memoized exact real arithmetic

The definition of $C_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $C_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 5 The average function lies in C_2 .

Theorem 6 Multiplication lies in C_2 .

From Theorems 5,6 one extracts implementations of addition and multiplication as memo-tries (relation to work by Hinze and Altenkirch?)

Experiments show considerable speed-up when sampling “hard” functions (e.g. high iterations of the logistic map) on a very fine grid.

Theorem 7 If $f \in C_1$, then $\int f \in C_0$.

The extracted program has some similarity with A. Simpson's, but is more efficient because the functions to be integrated are represented differently.

Generalisation: digit spaces

A *digit space* (X, D) consists of a set X and a set $D \subseteq X^X$.

This generalises the structure $(\mathbb{I}, \{\text{av}_d \mid d \in \text{SD}\})$.

Given digit space (X, D) and (Y, E) we define the set $C \subseteq X^Y$ of *digital maps* by

$$C := \nu F. \mu G. \{e \circ f \mid e \in E, f \in F\} \cup \{h : X \rightarrow Y \mid \forall d \in D h \circ d \in G\}$$

This generalises $C_1 \subseteq \mathbb{I}^{\mathbb{I}}$.

Fundamental results about digit spaces and their application

Theorem 8 Digit spaces and digital maps form a category with finite products.

The extracted program corresponds to the main results by Ghani/Hancock/Pattinson.

The generality of digit spaces can be used to obtain new algorithm in computable analysis (for example, power series using higher-order digit spaces).

What have we achieved?

- ▶ Programs with correctness proofs extracted (some new, some more efficient).
- ▶ Simple formalisation: Abstract classical reals, no streams, no trees,
- ▶ Simple proofs (you have to believe me).
- ▶ In some cases first hacking the program and then verifying it would have been much harder than the extraction of the program from a proof (for example, the proof that digit spaces have finite products).

Finite sets

Let $\mathcal{P}(X)$ be the classical powerset of X .

Define $\mathcal{P}_0(X) \subseteq \mathcal{P}(X)$ by a constructive inductive definition:

(i) $\emptyset \in \mathcal{P}_0(X)$

(ii) If $E \in \mathcal{P}_0(X)$ and $x \in X$, then $\{x\} \dot{\cup} E \in \mathcal{P}_0(X)$

where $A \dot{\cup} B := \{x \mid x \in A \vee x \in B\}$ and $\dot{\cup}$ is classical disjunction (we assume comprehension for classical properties, hence $A \dot{\cup} B$ exists).

In other words, $\mathcal{P}_0(X)$ is the least subset of $\mathcal{P}(X)$ such that

$$\mathcal{P}_0(X) = \{F \mid F = \emptyset \vee \exists x \exists E \in \mathcal{P}_0(X) F = \{x\} \dot{\cup} E\}$$

A realiser of “ $E \in \mathcal{P}_0(X)$ ” is a finite list $[a_1, \dots, a_n]$ such that a_i realises “ $x_i \in X$ ” and $E = \{x_1, \dots, x_n\}$.

In particular, if X is a “concrete” set, that is, its elements realise themselves, then a realiser of “ $E \in \mathcal{P}_0(X)$ ” is simply a listing of the elements of E .

Labelled transition systems

Let S, A be sets (states and labels). For simplicity let's assume both are concrete.

$$LTS_{S,A} := \mathcal{P}(S \times A \times S).$$

Finitely branching LTS Let $P \in LTS_{S,A}$.

$$FB_{S,A}(P) := \forall s \in S P(s) \in \mathcal{P}_0(A \times S)$$

where $P(s) := \{(a, t) \mid (s, a, t) \in P\}$.

A realiser of " $FB_{S,A}(P)$ " is a function $p: S \rightarrow [A \times S]$ such that $p(s)$ is a listing of all (a, t) with $P(s, a, t)$.

Constructing finitely branching LTS

$$\text{return}(a) := \{(s, a, s) \mid s \in S\} \text{ (for } a \in A\text{).}$$

$$\text{fail} := \emptyset$$
Lemma(a) $\text{FB}_{S,A}(\text{return}(a))$ (b) $\text{FB}_{S,A}(\text{fail})$ (c) If $\text{FB}_{S,A}(P)$ and $\text{FB}_{S,A}(Q)$, then $\text{FB}_{S,A}(A\tilde{\cup}B)$

If $P \in \text{LTS}_{S,A}$ and $Q_a \in \text{LTS}_{S,B}$ for $a \in A$, then we define

$$P >>= Q := \{(s, b, t) \mid \exists a, r (P(s, a, r) \wedge Q_a(r, b, t))\}$$

Lemma If $\text{FB}_{S,A}(P)$ and $\text{FB}_{S,B}(Q_a)$ for all $a \in A$, then $\text{FB}_{S,B}(P >>= Q)$.

From these lemmas the corresponding monadic parsers and parser combinators can be extracted.

For more parser combinators the set S must be instantiated by the set of strings.

What have we achieved?

- ▶ The well-known parser combinators by Hutton/Meijer have been extracted – with correctness and in particular termination proofs!
- ▶ In the (source) proofs no lists or higher-order functions occur.

Related work on realisability for (co)induction

M. Tatsuta, Realizability of Monotone Coinductive Definitions and Its Application to Program Synthesis. Proc. MPC, LNCS 1422, 338–364, 1998

F. Miranda-Perea, Realizability for Monotone Clausular (Co)Inductive Definitions, ENTCS 123, 179–193, 2005

H. Schwichtenberg, Minlog system,
<http://www.mathematik.uni-muenchen.de/~minlog/minlog/>

B., Realisability for induction and coinduction, Proceedings of Computability and Complexity in Analysis (CCA), 2009

Case studies in program extraction

- ▶ NbE extracted from Tait's SN proof for the simply typed λ -calculus.
Berghofer (Isabelle), Letouzey (Coq), Schwichtenberg, B. (Minlog).
- ▶ Program extracted from Nash-Williams classical proof of Dickson's Lemma and Higman's Lemma
Seisenberger, B. (Minlog).
- ▶ Programs extracted from Intermediate Value Theorem and Inverse Function Theorem for continuous real functions.
Schwichtenberg (Minlog).

Proof-theoretic strength

- ▶ Without classical axioms our formal system is an intuitionistic first-order version of the μ -calculus (or fixed point logic).
- ▶ With classical logic the system has the proof-theoretic strength of Π_2^1 -comprehension (Möllerfeld 2007).
- ▶ The intuitionistic version, however with non-strictly positive inductive definitions, has the same strength.

[S. Tupailo, On the intuitionistic strength of monotone inductive definitions, JSL 69(3), 790–798, 2004]

Conclusion

- ▶ Program extraction turned out to be very helpful (not a burden) in the example areas covered.
- ▶ Can we apply it to areas that are of less mathematical nature?
- ▶ Can we address resource issues?
- ▶ We need *much* smarter proof assistants to sell this.