Program extraction from proofs

Ulrich Berger - Swansea

MFPS, Pitsburgh, USA, 25 May 2011

Outline

Proofs as programs

Program extraction in computable analysis

Towards memoized functionals in higher types

Conclusion

Proofs as programs

Program extraction in computable analysis

Towards memoized functionals in higher types

Conclusion

The Curry-Howard correspondence (or Brouwer-Heyting-Kolmogorov interpretation)

Formulas correspond to *data types*
Proofs correspond to *programs*

$A \vee B$	disjoint sum
$A \wedge B$	cartesian product
$A \to B$	function space
$\exists x\, A$	cartesian product
$\forall x\, A$	function space

A proof of a formula A correponts to a program constructing an element of A.

- What is a function?
- What if the quantified x ranges over abstract objects?
- How do we interpret logical axioms, e.g. $A \vee \neg A$?
- How do we interpret maths axioms, e.g. induction, choice?
- Why is it interesting and useful?

Why Curry-Howard is interesting and useful

Foundations

Constructive foundation of Mathematics (Brouwer, Heyting, Kolmogorov, Gödel, Kleene, Kreisel, Martin-Löf). Properties of logical and mathematical systems (Realizability $\Rightarrow$ existence and disjunction property; Dialectica Interpretation $\Rightarrow$ consistency)

Programming

Program extraction (Minlog, Coq, Isabelle, Agda). In Minlog, realizability is used to automatically extract from a proof a program and its correctness proof ($\Rightarrow$ Monika's talk).

Mathematics

Approximation-, fixpoint- and ergodic-theory (Kohlenbach, DI). The study of function spaces has led to new developments in computability theory, topology, domain theory. The problem of C-H interpreting classical choice axioms has led to new recursion principles such as bar recursion and products of selection functions ($\Rightarrow$ Martin's and Paulo's talks).

What is a function and when is it a proof of an implication?

BHK-interpretation: A proof of $A \to B$ is a function f mapping proofs of A to proofs of B.

- f should be computable. What does this mean if A itself consists of functions? ($\Rightarrow$ computability in higher types)
- Don't we need a *proof* that f does it's job? (circularity!)

Realizing an implication

Realizability (Kleene, Kreisel)

$$f\,\mathbf{r}\,(A \to B) \quad \equiv \quad \forall a\,(a\,\mathbf{r}\,A \to f(a)\,\mathbf{r}\,B)$$

Dialectica Interpretation (Gödel)

$$(f,g)(A \to B) \quad \equiv \quad \forall a, v\,(ag(a,v)A \to f(a)vB)$$

where $aA \equiv \forall u\,(auA)$ and $bB \equiv \forall v\,(bvB)$ are purely universal formulas.

The idea is that for the conclusion, $f(a)vB$, the premise, $\forall u\,(auA)$ is used for finitely many u only (continuity argument), in fact, a single $u = g(a,v)$ suffices.

Both interpretations extract from a proof of A a term M and a proof of $M\,\mathbf{r}\,A$ (Soundness Theorem).

In the DI the proof of $M\,\mathbf{r}\,A$ takes place in a quantifier free system!

Realizing quantifiers

$$\begin{aligned}(x, a)\,\mathbf{r}\,\exists x\,A(x) &\equiv a\,\mathbf{r}\,A(x)\\ f\,\mathbf{r}\,\forall x\,A(x) &\equiv \forall x\,(f(x)\,\mathbf{r}\,A(x))\end{aligned}$$

x may range over abstract object (reals, real functions, . . .).

This seems to require a realizing programming language with data types for such abstract objects.

Alternative: uniform realization of quantifiers

$$\begin{aligned}a\,\mathbf{r}\,\exists x\,A(x) &\equiv \exists x\,(a\,\mathbf{r}\,A(x))\\ a\,\mathbf{r}\,\forall x\,A(x) &\equiv \forall x\,(a\,\mathbf{r}\,A(x))\end{aligned}$$

For concrete objects we may relativize the quantifiers:

$$\forall x\,(\mathbb{N}(x) \to \exists y\,(\mathbb{N}(y) \wedge (x = 2y \vee x = 2y + 1)))$$

where $\mathbb{N}$ is defined such that $n\,\mathbf{r}\,\mathbb{N}(x)$ means that n is a representation of the natural number x.
The extracted program computes integer division by 2.

Program extraction and the law of excluded middle

Realizing, say, $\forall x\,(\mathbb{N}(x) \to A(x) \vee \neg A(x))$ would mean to construct a program computing for every (representation of) a natural number x a realizer of $A(x)$ or a realizer or $\neg A(x)$. This is impossible, in general.

But, one can eliminate LEM in proofs of formulas of the form

$$\forall x\,(\mathbb{N}(x) \to \exists y\,(\mathbb{N}(y) \wedge A_0(x, y))$$

where $A_0(x, y)$ is decidable, using Gödel's negative translation and the Friedman/Dragalin A-translation ($\Rightarrow$ Monika's talk).

Other approaches to program extraction from classical proofs

- ϵ-substitution calculus (Hilbert).
- Interpretation of $\neg\neg A \to A$ by continuations (Felleisen).
- Direct computational interpretation of classical sequent calculus ($\lambda\mu$-calculus, Parigot).
- Interpretation of restricted forms of LEM by learning based realizability (Berardi, Aschieri)
- Realizability interpretation of classical systms via stacks and processes (Krivine).

Interpreting induction

Induction on natural numbers

$$A(0) \wedge \forall x\,(A(x) \to A(x+1)) \to \forall x\,(\mathbb{N}(x) \to A(x))$$

is a special case of induction on an inductively defined predicate: Set $\Phi(X) := \{0\} \cup \{x+1 \mid x \in X\}$, then $\mathbb{N} = \mu\Phi = \mu X.\Phi(X)$

In general, one has for a monotone predicate transformer Φ an induction schema for its least fixed point $\mu\Phi$:

$$\Phi(\mathcal{P}) \subseteq \mathcal{P} \to \mu\Phi \subseteq \mathcal{P}$$

The data type associated with $\mu\Phi$ is the initial algebra $\mathrm{In}_\varphi : \varphi(\mu\varphi) \to \mu\varphi$ of a functor φ derived from Φ. The induction scheme is realized by the iterator $\mathbf{It}_\varphi$ that iterates any "step function" (i.e. φ-algebra) $f : \varphi(\alpha) \to \alpha$ to an algebra morphism $\mathbf{It}_\varphi(f) : \mu\varphi \to \alpha$ with computation rule (i.e. morphism equation)

$$\mathbf{It}_\varphi(f)\,\mathrm{In}_\varphi(m) = f(\mathbf{map}_\varphi(\mathbf{It}_\varphi(f))(m))$$

Example: Natural numbers

Recall $\mathbb{N} = \mu\Phi$ where

$$\begin{aligned}\Phi(X) &= \{0\} \cup \{x+1 \mid x \in X\} \\ &= \{y \mid y = 0 \vee \exists x\,(y = x+1 \wedge x \in X)\}\end{aligned}$$

The functor associated with Φ is obtained by removing all first-order parts from Φ: $\varphi(\alpha) = 1 + \alpha$ The initial algebra $\mathrm{In}_\varphi : \varphi(\mu\varphi) \to \mu\varphi$ is the familiar structure of unary natural numbers $\mathbb{N} := \mu\varphi$ generated by zero and successor.

A step function $f : \varphi(\alpha) \to \alpha$ consists of $f_0 : \alpha$ and $f_1 : \alpha \to \alpha$. The iteration $g := \mathbf{It}_\varphi(f) : \mathbb{N} \to \alpha$ is defined recursively by $g(0) = f_0$, $g(\mathrm{S}(n)) = f_1(g(n))$.

Remarks: 1. The variables x, y may range over abstract objects, for example the real numbers. 2. Category theory is only needed to explain realizability. The "user" doesn't have to know anything about this.

Interpreting choice axioms

The (constructive) axiom of choice

$$\forall x\, \exists y\, A(x, y) \to \exists f\, \forall x\, A(x, f(x))$$

has a trivial realizer, namely the identity (both with the traditional and the uniform interpretation of quantifiers)

Much harder is the *classical* axiom of choice which is obtained by double negation translation of the constructive axiom of choice. Even classical *countable* classical choice is hard to realize:

$$\forall x \in \mathbb{N}\, (\neg\neg \exists y\, A^{\neg\neg}(x, y) \to \neg\neg \exists f\, \forall x\, A^{\neg\neg}(x, f(x)))$$

Classical countable choice is the main stumbling block in extending program extraction from classical proofs to analysis.

More about this in the other three talks.

In contrast, the negative translation of an induction axiom is unproblematic, since it is again instances of an induction axiom.

Proofs as programs

Program extraction in computable analysis

Towards memoized functionals in higher types

Conclusion

Coinduction

Coinduction is dual to induction. Given a monotone predicate transformer Φ we hav a coinduction scheme for its greatest fixed point $\nu\Phi$:

$$\mathcal{P} \subseteq \Phi(\mathcal{P}) \to \mathcal{P} \subseteq \nu\Phi$$

The associated data type is the final coalgebra $\mathrm{Out}_\varphi : \nu\varphi \to \varphi(\mu\varphi)$.

The coinduction scheme is realized by the coiterator $\mathbf{Coit}_\varphi$ that coiterates any "step function" (i.e. φ-coalgebra) $f : \alpha \to \varphi(\alpha)$ to a coalgebra morphism $\mathbf{Coit}_\varphi(f) : \alpha \to \mu\varphi$ with computation rule (i.e. morphism equation)

$$\mathrm{Out}_\varphi(\mathbf{Coit}_\varphi(f)(a)) = \mathbf{map}_\varphi(\mathbf{Coit}_\varphi(f))(f(a))$$

Equivalently, using the fact that Out_φ has an inverse In_φ,

$$\mathbf{Coit}_\varphi(f)(a) = \mathrm{In}_\varphi(\mathbf{map}_\varphi(\mathbf{Coit}_\varphi(f))(f(a)))$$

Example: Signed digit representation

We are after a signed digit representation of real numbers x in the compact interval $\mathbb{I} := [-1, 1]$, i.e. we want

$$x = \sum_{n=0}^{\infty} d_n \cdot 2^{-(n+1)} \tag{1}$$

where $d_i \in \mathrm{SD} := \{-1, 0, 1\}$.

Note (1) is equivalent to the fact that there are $x_0, x_1, \ldots \in \mathbb{I}$ such that $x = 1/2(d_0 + x_0) = 1/2(d_0 + 1/2(d_1 + x_1)) = \ldots$

This suggests the following coinductive predicate on $\mathbb{I}$:

$$\mathrm{C}_0 = \nu X.\{x \mid \exists d \in \mathrm{SD}\, \exists x_0\, (x = \frac{d + x_0}{2} \wedge X(x_0)\}$$

The data type associated with C_0 is type of infinite streams of signed digits. A stream $d_0, d_1, \ldots$ realizes $\mathrm{C}_0(x)$ precisely when (1) holds.

Extracting exact real number algorithms

Using coinduction one can prove, for example:

Theorem 1 $x \in \mathrm{C}_0 \quad \text{iff} \quad \forall n \in \mathbb{N}\, \exists q \in \mathbb{Q} \cap \mathbb{I}\, |x - q| \leq 2^{-n}$.

Theorem 2 If $x, y \in \mathrm{C}_0$ then $\frac{x+y}{2} \in \mathrm{C}_0$.

Theorem 3 If $x, y \in \mathrm{C}_0$ then $xy \in \mathrm{C}_0$.

From the proofs of these theorems one extracts a program translating between the signed-digit- and the Cauchy-representation, as well as implementations of addition and multiplication w.r.t. the signed digit representation.

Similar implementations were studied by Edalat, Potts, Heckmann, Escardo, Marcial-Romero, Ciaffaglione, Gianantonio, ...

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

Characterizing uniform continuity by induction/coinduction

Recall the coinductive defintition of reals in $\mathbb{I}$ that have a signed digit representation:

$$\mathrm{C}_0 = \nu X.\{x \mid \exists d \in \mathrm{SD}\, \exists x_0\, (x = \mathrm{av}_d(x_0) \wedge X(x_0)\}$$

where $\mathrm{av}_d(x_0) := \frac{d+x_0}{2}$.

We generalize this to a characterization of (uniformly) continuous functions $f : \mathbb{I} \to \mathbb{I}$:

$$\begin{aligned}\mathrm{C}_1 \quad = \quad \nu X.\mu Y.\{f \mid \exists d \in \mathrm{SD}\, \exists f_0\, (f = \mathrm{av}_d \circ f_0 \wedge X(f_0)) \\ \vee \ \forall d \in \mathrm{SD}\ Y(f \circ \mathrm{av}_d)\}\end{aligned}$$

The left disjunct is analoguous to C_0 and means that f *emits* a digit.

The right disjunct means that f *absorbs* a digit from the input.

Memo trees (tries?) for continuous functions

Theorem 4 $f \in \mathbb{I}^{\mathbb{I}}$ is continuous iff $f \in \mathrm{C}_1$.

From the proof of this theorem one extracts programs translating between realisers of "f is continuous" (where continuity has to be defined in a contructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

What is a realiser of "$f \in \mathrm{C}_1$"?

It is a finitely branchning 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 $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

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 $f \in \mathrm{C}_1$, then $\int f \in \mathrm{C}_0$.

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

Proofs as programs

Program extraction in computable analysis

Towards memoized functionals in higher types

Conclusion

Proofs as programs

Program extraction in computable analysis

Towards memoized functionals in higher types

Conclusion