A coinductive approach to computable analysis

Ulrich Berger

Swansea University

Duality in Computer Science

Dagstuhl, 31 July 2013

Constructive and computational duality

Leading idea (e.g. in Point Free Topology or Type Two Theory of Effectivity):

Replace classical abstract objects such as points of a topological space by constructively more meaningful objects such as basic open sets.

Duality links the two worlds and provides a notion of representation of abstract objects.

In order to obtain computationally meaningful results one has to work with the representations, hence one has to redo mathematics.

The realizability approach to duality

Realizability (originating from work of Kleene, Goedel and Kreisel in the 1940s and 50s) offers an alternative approach to duality where one can stay within abstract mathematics.

Roughly:

- Realizability links classical objects with computations.
- Constructive proofs (about classical objects) provide realizers, i.e. representations.

In this talk:

- Example: coinductive continuous real functions.
- Theory: realizability for Church's Simple Theory of Types.

Formalizing real numbers

We assume that the structure $\mathbb{R}$ of real numbers with $0, 1, +, -, *, /, =, <, \sin, \mathrm{sg}, \ldots$ is given axiomatically (no implementation or computational model provided).

Any true disjunction-free first-order formulas are allowed as axioms.

Since in classical logic disjunction can be expressed by other logical connectives, all classically true statements can be axioms.

In addition true higher-order formulas satisfying certain syntactic criteria (details later) are allowed. For example, completeness:

$$X \text{ nonempty and bounded } \rightarrow X \text{ has l.u.b}$$

Discontinuous and partial fuctions

Discontinuous functions are allowed. E.g. the sign function with the axioms

$$\begin{aligned} x < 0 &\rightarrow \mathrm{sg}(x) = -1 \\ x = 0 &\rightarrow \mathrm{sg}(x) = 0 \\ x > 0 &\rightarrow \mathrm{sg}(x) = 1 \end{aligned}$$

The partial function $1/x$ can be thought of as being totalized, however, without stating anything about $1/0$:

$$x \neq 0 \rightarrow x * 1/x = 1$$

Natural numbers, integers, rational numbers

. . . are defined as subsets of $\mathbb{R}$:

$$
\begin{aligned}
x \in \mathbb{N} &\overset{\mu}{\equiv} x = 0 \vee x - 1 \in \mathbb{N} \\
x \in \mathbb{Z} &\equiv x \in \mathbb{N} \vee -x \in \mathbb{N} \\
x \in \mathbb{Q} &\equiv \exists n \in \mathbb{Z}, m \in \mathbb{N} \setminus \{0\}\ x = n/m
\end{aligned}
$$

where "μ" means that $\mathbb{N}$ is inductively defined, i.e. it is the least set satisfying the equation.

Potential realizers

The space of potential realizers is the Scott domain D defined by the recursive domain equation

$$D = \mathbf{1} + D + D + D \times D + [D \to D]$$

where $+$ is the domain-theoretic sum which adds a new bottom element and $[\cdot \to \cdot]$ is the continuous function space.
The equation is reflected by the following continuous functions:

$$\begin{aligned}
\mathrm{Nil} &: D \\
\mathrm{L}, \mathrm{R} &: D \to D \\
\mathrm{Pair} &: D \to D \to D \\
\mathrm{Fun} &: (D \to D) \to D \\
\\
\mathrm{case} &: (D \to D) \to (D \to D) \to D \to D \\
\mathrm{pr_L}, \mathrm{pr_R} &: D \to D \\
\cdot &: D \to D \to D
\end{aligned}$$

Computational adequacy

The functions $\mathrm{Nil}, \mathrm{L}, \mathrm{R}, \mathrm{Pair}, \mathrm{Fun}, \mathrm{case}, \mathrm{pr}_\mathrm{L}, \mathrm{pr}_\mathrm{R}, \cdot$ can be formally represented as constants in a small (essentially untyped) functional programming language with a natural **denotational semantics** in the domain D.

This language has also a (lazy) **operational semantics** which is adequate in the following sense:

Adequacy Theorem
If a program M denotes a finite data $d \in D$ (built from Nil, L, R, Pair), then M computes (the canonical representation of) d.

Proof-theoretic adequacy

Our programming language can be axiomatized by a set of equations E which are sound in the sense that if $E \vdash M = N$, then $D \models M = N$.

As a corollary we obtain:

Proof-theoretic Adequacy Theorem
If $E \vdash M = d$ for some finite data d, then M computes d.

Realizability

For every formula A we define a formula $d\,\mathbf{r}\,A$ where d is a variable ranging over the domain D:

$$\begin{array}{lcl}
d\,\mathbf{r}\,(A \wedge B) & \equiv & (\mathrm{pr}_{\mathrm{L}}\,d)\,\mathbf{r}\,A \wedge (\mathrm{pr}_{\mathrm{R}}\,d)\,\mathbf{r}\,B \\
d\,\mathbf{r}\,(A \vee B) & \equiv & \exists a\,(d = \mathrm{L}\,a \wedge a\,\mathbf{r}\,A) \vee \exists b\,(d = \mathrm{R}\,b \wedge b\,\mathbf{r}\,B) \\
d\,\mathbf{r}\,(A \to B) & \equiv & \forall a\,(a\,\mathbf{r}\,A \to (d \cdot a)\,\mathbf{r}\,B) \\
d\,\mathbf{r}\,\forall x\,A & \equiv & \forall x\,(d\,\mathbf{r}\,A) \\
d\,\mathbf{r}\,\exists x\,A & \equiv & \exists x\,(d\,\mathbf{r}\,A) \\
d\,\mathbf{r}\,(\vec{t} \in P) & \equiv & \vec{t} \in P \\
d\,\mathbf{r}\,(\vec{t} \in X) & \equiv & (\vec{t}, d) \in \tilde{X} \\
d\,\mathbf{r}\,(\vec{t} \in I) & \equiv & (\vec{t}, d) \in \tilde{I}
\end{array}$$

where P is an atomic predicate, X is a predicate variable, and $I, \tilde{I}$ are inductively defined by

$$\begin{array}{rcl}
\vec{x} \in I & \overset{\mu}{\equiv} & A \\
(\vec{x}, d) \in \tilde{I} & \overset{\mu}{\equiv} & d\,\mathbf{r}\,A
\end{array}$$

Soundness and program extraction

Soundness Theorem.
From a constructive proof of a formula A using assumptions $B_1, \ldots, B_n$, one can extract a program M, possibly containing variables $b_1, \ldots, b_n$, such that $M\,\mathbf{r}\,A$ is provable from the assumptions $b_1\,\mathbf{r}\,B_1, \ldots, b_n\,\mathbf{r}\,B_n$.

The proof yields in fact more: To every sub-program M' of M a formula A' is assigned and a proof of $M'\,\mathbf{r}\,A'$.
This means that one has specifications and correctness proofs of all sub-programs.

Program Extraction Theorem
From a constructive proof of a $\forall{\to}$-free formula A using true $\vee$-free assumptions $B_1, \ldots, B_n$, one can extract a program M such that M computes a data d and $d\,\mathbf{r}\,A$ is provable.

Extracting integer arithmetic

Recall: $x \in \mathbb{N} \stackrel{\mu}{\equiv} x = 0 \vee x - 1 \in \mathbb{N}$.

A realizer of $x \in \mathbb{N}$ is a unary representation of x.

In order to obtain binary representations we define

$$x \in \mathbb{N}_2 \quad \stackrel{\mu}{\equiv} \quad x \in \{0,1\} \vee \exists y > 0\,(y \in \mathbb{N}_2 \wedge \exists d \in \{0,1\}\,(x = 2y + d))$$

Theorem

- $x \in \mathbb{N} \leftrightarrow x \in \mathbb{N}_2$.
- $x \in \mathbb{N}_2 \wedge y \in \mathbb{N}_2 \to x + y \in \mathbb{N}_2$.
- $x \in \mathbb{N}_2 \wedge y \in \mathbb{N}_2 \to x * y \in \mathbb{N}_2$.

The extracted programs provably translate between unary and binary numbers, and compute addition and multiplication on binary numbers.

Approximating real numbers

Two (equivalent) ways of saying that a real number $x \in \mathbb{I} := [-1, 1]$ can be approximated:

$$\begin{array}{rcl} A(x) & \Leftrightarrow & |x| < 1 \wedge \forall n \in \mathbb{N}\, \exists q \in \mathbb{Q}\; x \in \mathrm{B}_n(q) \\ \mathrm{C}_0(x) & \overset{\nu}{\Leftrightarrow} & |x| < 1 \wedge \exists d \in \{-1, 0, 1\}\; \mathrm{C}_0(2 * x - d) \end{array}$$

where $x \in \mathrm{B}_n(q) \equiv |x - q| < 2^{-n}$.

A realizer of $A(x)$ is a fast rational Cauchy sequence converging to x.

A realizer of $\mathrm{C}_0(x)$ is an infinite stream of signed digits, $d_0 : d_1 : \ldots$ representing x, i.e.

$$x = \sum_0^\infty d_i * 2^{i+1}$$

Theorem $A(x) \leftrightarrow \mathrm{C}_0(x)$.

The extracted program translates between Cauchy and signed digit representation.

Extracting exact real number arithmetic

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

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

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

A coinductive definition of continuity

One can define continuous functions by a nested inductive/coinductive definition.

In order to carry this out conveniently, we introduce explicit operators for least and greatest fixed points of monotone operators $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$:

$$\begin{array}{lcl} \mu\, X\, \Phi(X) & := & \text{least fixed point of } \Phi \\ \nu\, X\, \Phi(X) & := & \text{largest fixed point of } \Phi \end{array}$$

Hence, for example $\mathbb{N} = \mu\, X\, .\, \{x \mid x = 0 \vee x - 1 \in X\}$.

We define $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ by a nested inductive/coinductive definition as follows (F, G range over subsets of $\mathbb{I}^{\mathbb{I}}$):

$$\begin{array}{rcl} \mathrm{C}_1 & = & \nu\, F\, .\, \mu\, G\, .\, \{f : \mathbb{I}^{\mathbb{I}} \mid (\exists e \in \mathrm{SD}\, \mathrm{va}_e \circ f \in F) \vee \\ & & \qquad (\forall d \in \mathrm{SD}\, f \circ \mathrm{av}_d \in G)\} \end{array}$$

Memo tries for continuous functions

Theorem h is continuous iff $h \in \mathrm{C}_1$.

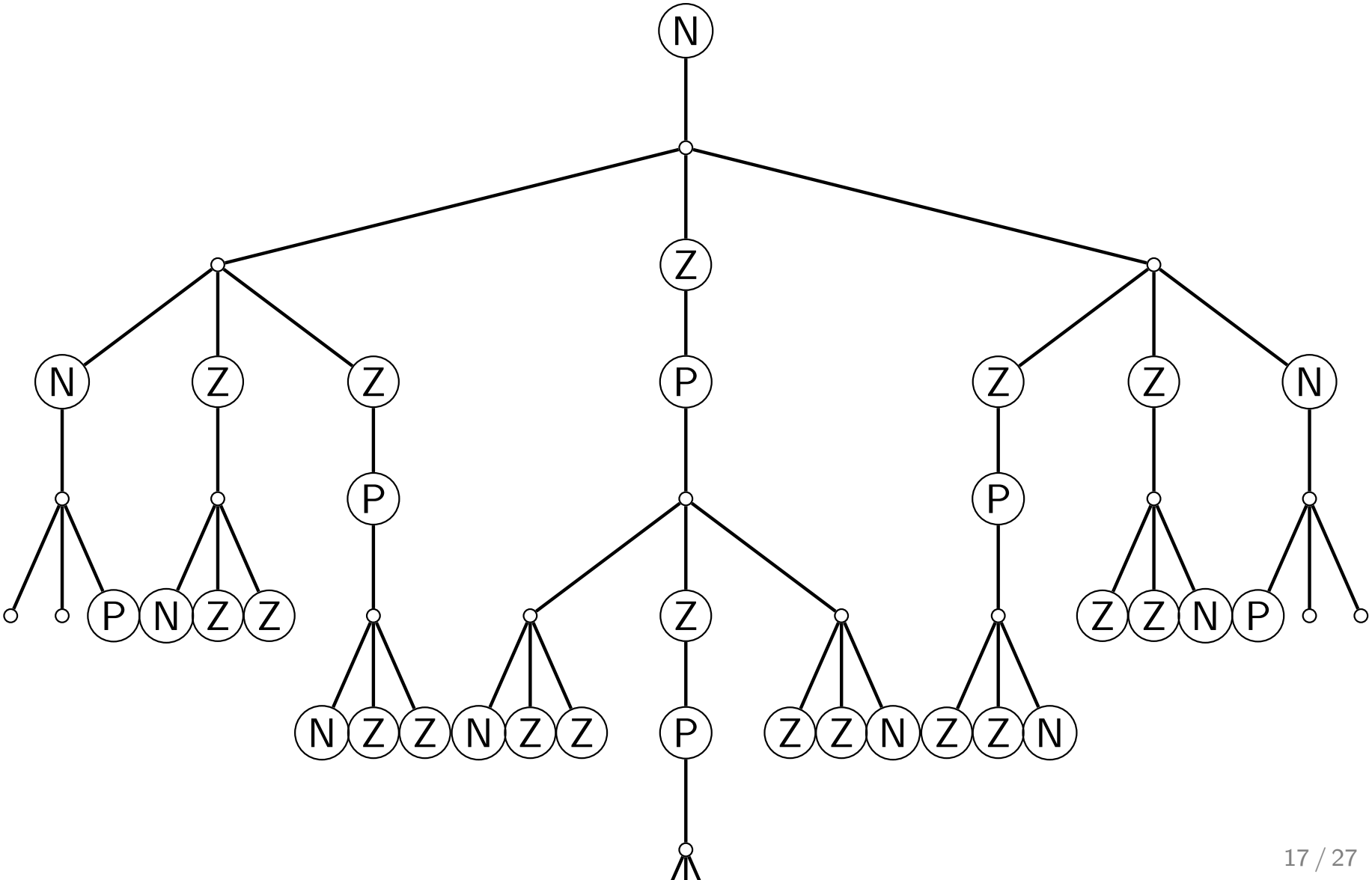
From the proof of this theorem one extracts programs translating between realisers of "f is continuous" (where continuity has to be defined in a constructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

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

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.
P. Hancock, D. Pattinson, N. Ghani. Representations of Stream Processors Using Nested Fixed Points, LMCS 5, 2009.

Tree of the logistic map, $f_a(x) = a(1 - x^2) - 1$, with $a = 2/3$



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 The average function lies in C_2.

Theorem Multiplication lies in C_2.

Theorem If $f \in \mathrm{C}_n$ and $g_1, \ldots g_n \in \mathrm{C}_m$, then $f \circ (g_1, \ldots, g_n) \in \mathrm{C}_m$.

From these Theorems one extracts implementations of addition and multiplication as memo-tries, and of composition as a function on memo-tries.

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

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

We assume the following "axioms" about $\int f$:

(a) $\int f = \frac{1}{2}\int(\mathrm{va}_d \circ f) + d$ where $\mathrm{va}_d(x) := 2x - d$.

(b) $\int f = \frac{1}{2}(\int(f \circ \mathrm{av}_{-1}) + \int(f \circ \mathrm{av}_1))$.

Theorem If $f \in \mathrm{C}_1$, then $\int f \in A$, i.e.
$\forall n \in \mathbb{N}\, \exists q \in \mathbb{Q}\, |\int f - q| \le 2^{-n}$.

The proof is very short and uses only the equations (a), (b) above.
The extracted program is reasonably efficient.

Church's theory of simple types (CST)

Alonzo Church: A Formulation of the Simple Theory of Types. The Journal of Symbolic Logic, Vol. 5, No. 2. 1940).

CST is a formal system for higher order-logic presented as a simply typed lambda calculus.

Types

a set $\mathcal{I}$ of base types for individuals;
the base type o (type of propositions, or truth values);
$\rho \times \sigma$, $\rho \to \sigma$.

Constants

$$\begin{array}{rcl} \to, \wedge, \vee & : & o \to o \to o \\ \forall_\rho, \exists_\rho & : & (\rho \to o) \to o \quad (\rho \text{ arbitrary}) \end{array}$$

Terms

$$\begin{array}{rcl} \mathrm{TER} \ni M, N & ::= & x \in \mathrm{VAR} \mid c \in \mathrm{C} \mid \\ & & \mid \lambda x : \rho . M \mid MN \mid \langle M, N \rangle \mid \pi_0(M) \mid \pi_1(M) \end{array}$$

Semantics of CST

CST admits a straightforward classical semantics:

$\iota =$ a set $(\iota \in \mathcal{I})$
$o = \{0, 1\}$
$\times =$ cartesian product
$\rightarrow =$ full function space

The propositional connective are interpreted as usual:

$$\begin{aligned} \forall_\rho(p) &= \min\{p(x) \mid x \in \rho\} \\ \exists_\rho(p) &= \max\{p(x) \mid x \in \rho\} \end{aligned}$$

Proofs in CST

A typing context is a sequence $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$.

A Γ-formula is a term that has type o in the typing context Γ.

One can define an intuitionistic proof calculus for sequents of the form $\Delta \vdash_\Gamma A$ where Δ is a finite set of Γ-formulas and A is a Γ-formula.

Negation, equality, least and greatest fixed points

Truth, falsity and negation can be defined as

$$\begin{aligned} \top &:= \exists x : o \,.\, x \\ \bot &:= \forall x : o \,.\, x \\ \neg A &:= A \to \bot \end{aligned}$$

Equality can be defined as

$$x =_\rho y \;:=\; \forall p : \rho \to o \;.\; p\, x \to p\, y$$

One can also define least and greatest fixed point operators $\mu_\rho, \nu_\rho : (\rho \to \rho) \to \rho$ for "predicate types" ρ (e.g. $\rho = \iota \to o$).

The expected rules can be derived.

Realizability for CST (brief sketch)

Let δ be a new base type denoting the domain D.

For formulas $A : o$ realizability is clear: the realizability interptetation of A must define a set of realizers, i.e. a term $\mathbf{R_r}(A)$ of type $\delta \to o$.

But what is the realizability interpretation of a term of higher type?

It is obtained by applying a simple substitution that assigns to each constant $c : \rho$ a suitable closed term $\mathbf{R_r}(M) : \mathbf{r}(\rho)$ where $\mathbf{r}(\rho) := \rho[o := \delta \to o]$.

For example,

$$\mathbf{R}(\to) \quad := \quad \lambda\, A, B : \delta \to o\, \lambda d : \delta\, .\, \forall a : \delta . A\, a \to B\, (\mathrm{app}\, d\, a)$$

Although the definition of realizability is now straight forward, program extraction and the proof of the Soundness Theorem aren't.

Remarks

- The realizability interpretation of CST is a vast generalization of existing interpretations, e.g. by M Tatsuta and implementations in Minlog and Coq. I.p. only strictly positive induction and coinduction had been considered so far.

- The soundness proof for monotone induction and coinduction seems to need induction and coinduction for non-mononotone operators.

- A prototype implementation is under development (joint work with T Hou).

Conclusion

Program extraction via realizability . . .

- defines representations implicitely by predicates (e.g. $A \subseteq \mathbb{R}$ instead of $\rho \subseteq: \mathbb{N}^{\mathbb{N}} \to \mathbb{R}$);
- analyses the computational content of proofs of theorems;
- provides a useful guideline for designing data, representations and algorithms, even in the absence of a proof assistant.

Some References

- Andrej Bauer, The Realizability Approach to Computable Analysis and Topology, PhD thesis, 2000.
- Andrej Bauer and Iztok Kavkler, Implementing real numbers with RZ, ENTCS 202, 2008.
- B, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger, Minlog - A Tool for Program Extraction for Supporting Algebra and Coalgebra. LNCS 6859, 2011.
- Helmut Schwichtenberg, Realizability interpretation of proofs in constructive analysis, TOCS 43(3-4), 2008.
- Helmut Schwichtenberg, Inverting monotone functions in constructive analysis, LNCS 3988, 2006.
- B and Monika Seisenberger, Proofs, programs, processes. TOCS 51(3), 2012.
- B, Tie Hou, Typed vs Untyped realizability, ENTCS 286, 2012.
- Andrew Lawrence, B, Monika Seisenberger, Extracting a DPLL Algorithm. ENTCS 286, 2012.
- B, From coinductive proofs to exact real arithmetic: theory and applications. LMCS 7(1), 2011.