

# Computational efficiency of coinduction

Ulrich Berger  
Swansea University

*LCC 2016*

*September 2, Marseille*

## Approaches to implicit complexity

Given: a class  $P$  of bounding functions.

### *Programs*

High-level programming languages or term calculi that do not mention run-time bounds explicitly, but define exactly the  $P$ -time computable functions (bounded recursion on notation, safe recursion, ramification, special term-rewriting systems, ...).

### *Totality*

Proof systems that can derive the totality (i.e. termination) of a partial recursive function  $f$  exactly if  $f$  is in  $P$ -time (Intrinsic theories, applicative theories, ...).

### *Extraction*

Proof systems whose extracted programs are exactly the  $P$ -time functions (Bounded Arithmetic, Linear Arithmetic, ...).

## Linking theory and practice

The approaches above mainly aim at characterising or separating complexity classes.

On the other hand, program extraction has been applied to synthesize useful programs from proofs such as sorting, parsing, SAT-solving, exact real number computation, however, so far without considering precise calibrations of computational complexity.

It is the aim of this talk to hint at possibilities to link the theoretical and practical work in order to obtain useful programs that are not only provably correct, but also efficient.

## Infinite data, corecursion, coinduction

We will concentrate on corecursive computation on infinite data and the corresponding proof principle of coinduction, which received growing interest, both on the theoretical and the practical side.

Leivant, Ramyaa: Ramified Corecurrence and Logspace. ENTCS 276, 2011.

B: From coinductive proofs to exact real arithmetic: theory and applications. LMCS 7(1), 2011.

We will use examples from real number computation to illustrate coinduction and corecursion.

## Formal framework

- ▶ Intuitionistic many-sorted logic in finite types.
- ▶ Sorts represent abstract mathematical structures given by  $\forall$ -free axioms.
- ▶ Inductive and coinductive definitions of predicates as least and greatest fixed points of monotone predicate transformers.

$$\frac{}{\Gamma \vdash \Phi(\mu \Phi) \subseteq \mu \Phi} \text{ Closure}$$

$$\frac{\Gamma \vdash \Phi(\mathcal{P}) \subseteq \mathcal{P}}{\Gamma \vdash \mu \Phi \subseteq \mathcal{P}} \text{ Induction}$$

$$\frac{}{\Gamma \vdash \nu \Phi \subseteq \Phi(\nu \Phi)} \text{ Coclosure}$$

$$\frac{\Gamma \vdash \mathcal{P} \subseteq \Phi(\mathcal{P})}{\Gamma \vdash \mathcal{P} \subseteq \nu \Phi} \text{ Coinduction}$$

# Realizability

- ▶ Realizers are untyped recursive programs.
- ▶ The definition of realizability is usual, e.g. a function  $f$  realizes the formula  $A \rightarrow B$  if it maps realizers of  $A$  to realizers of  $B$ .

Exception: quantifiers are interpreted uniformly:

- ▶  $\mathbf{ar} \exists x A(x)$  means  $\exists x (\mathbf{ar} A(x))$ .
- ▶  $\mathbf{ar} \forall x A(x)$  means  $\forall x (\mathbf{ar} A(x))$ .

## Real, natural and rational numbers

The structure of the *real numbers*  $\mathbb{R} = (0, 1, +, *, -, /, <, | \cdot |)$  is treated as a sort specified by  $\forall$ -free axioms.

The *natural numbers* are defined as the least subset of  $\mathbb{R}$  that contains 0 and is closed under successor:

$$\mathbb{N} \stackrel{\mu}{=} \{0\} \cup \{x + 1 \mid x \in \mathbb{N}\}$$

*Realizability* automatically associates with this definition the unary representation of natural numbers and with proofs of closure properties of  $\mathbb{N}$  programs operating on that representation.

Alternatively:

$$\mathbb{B} \stackrel{\mu}{=} \{0\} \cup \{2x + d \mid x \in \mathbb{N}, d \in \{0, 1\}\}$$

Clearly,  $\mathbb{N} = \mathbb{B}$ , but the realizers of elements of  $\mathbb{B}$  are binary words representing natural numbers in binary.

Note: Words occur as realizers only, not in the formal system.

## Signed digit representation of real numbers

Let  $SD = \{-1, 0, 1\}$  be the set of signed digits and  $\mathbb{I}$  the closed real interval  $[-1, 1]$ .

Furthermore,  $\mathbb{I}_{-1} := [-1, 0]$ ,  $\mathbb{I}_0 := [-1/2, 1/2]$ ,  $\mathbb{I}_1 := [0, 1]$  (left, middle and right part of  $\mathbb{I}$ ).

We define  $C$  (coinductively) as the largest subset of  $\mathbb{R}$  satisfying

$$C \stackrel{\nu}{=} \{x \mid \exists d \in SD . x \in \mathbb{I}_d \wedge 2x - d \in C\}$$

Classically,  $C = \mathbb{R}$ , but the point is that the realizers of  $x \in C$  are infinite streams of signed digits  $d_0 : d_1 : \dots$  representing  $x$ , i.e.

$$x = \sum_{i \in \mathbb{N}} d_i * 2^{-(i+1)}$$

## Extracting real arithmetic

From proofs of closure properties like

$$x, y \in \mathbb{C} \rightarrow x + y \in \mathbb{C}$$

one can extract corecursive implementations of arithmetic operations processing stream representations of real numbers.

Note again that neither streams nor corecursion are part of the formal language.

## Project:

Transfer Leivant's and Ramyaa's ramified approach to log-space stream computation to the proof-theoretic level.

Needed: Suitable restrictions of induction and coinduction (and the logic?) to mimic safe corecursion.

## Continuous functions

We define a predicate  $C_1$  on the set of functions  $f \in \mathbb{I}^{\mathbb{I}}$  by a nested inductive/coinductive definition.

$$C_1 := \nu \lambda F \mu \lambda G \{ g \mid (\exists d \in \text{SD} \exists f \in F . g = \text{av}_d \circ f) \\ \vee \bigwedge_{d \in \text{SD}} g \circ \text{av}_d \in G \}$$

One can show that  $C_1$  consists exactly of the uniformly continuous functions.

Nested inductive/coinductive definitions similar to  $C_1$  were also considered by Hancock, Ghani, Pattinson (2009) and Nakata and Uustalu (2010).

## Continuous functions as trees

A realizer of " $f \in C_1$ " is a non-wellfounded tree that encodes an algorithm for  $f$  acting on realizers of arguments  $x \in C$ , i.e. on signed digit representations.

From a proof that the functions  $f \in C_1$  map  $C$  to  $C$  one extracts a program that interprets these trees as stream transformers implementing the function  $f$  with respect to the signed digit representation.

Moreover one can prove that  $C_1$  is closed under composition. The extracted program composes trees, and one can observe that composing trees is more efficient than composing the corresponding functions.

## Example: The logistic map

$$f_a : \mathbb{I} \rightarrow \mathbb{I}, \quad f_a(x) = a(1 - x^2) - 1 \quad (a \in [0, 2])$$

$f_a$  can be shown to be in  $C_1$ , hence a tree implementing  $f_a$  can be extracted. Our goal is to compute iterations of  $f_a$ .

$f_a^n$  is a polynomial of degree  $2^n$  that wildly oscillates in the interval  $\mathbb{I}$ , with  $2^n$  local maxima. Hence, it is rather hard to compute the value of, say,  $f_2^{100}(0.7)$  with high precision.

## Efficiency through memoization

It turns out that computing the expression  $f_2^{100}(0.7)$  with the tree representing  $f_2^{100}$  is far more efficient than iterating the function  $f_2$  100 times.

The speed-up is particularly significant if  $f_2^{100}(b)$  is computed for a large number of rationals  $b$  close to each other.

The reason is that during the computation larger and larger initial segments of the infinite tree representing  $f_2^{100}$  are computed which are reused. Hence these trees act as memoization tables for continuous real functions.

The structure of the memoization table is completely determined by the logical form of the predicate  $C_1$  and its construction is purely driven by the proof that  $f_2^{100} \in C_1$ .

## Project: Complexity of real functions

The tree representation of continuous real functions induced by the predicate  $C_1$  can be viewed as the stream of its finite levels and hence suggests a similar correspondence.

However, it is not hard to see that every computable tree representing a real function can be transformed into a tree of very low complexity that represents the same function, simply by introducing unnecessary reading steps, i.e. choosing an unnecessarily big modulus of continuity.

This can be remedied by an effective “normalization” procedures for trees that removes unnecessary reading steps.

Is it possible to strengthen the predicate  $C_1$  so that realizers of  $x \in C_1$  are automatically normalized?