Non-determinism and partiality in realizability: Pure Gray code

Ulrich Berger
Swansea University

j.w.w.

Hideki Tsuiki
Kyoto University

Interval Analysis and Constructive Mathematics
CMO-BIRS workshop 16w5099
Oaxaca, November 13-18, 2016

Gray code for real numbers

Gray code for real numbers was introduced by Hideki Tsuiki in

> *Real number computation through Gray code embedding. Theoretical Computer Science, 284:467–485, 2002.*

Pure Gray code represents a real number in $[-1, 1]$ by its itinerary of the *tent map*

$$\mathbf{tent}(x) = 1 - 2|x|$$

That is, $x \in [-1, 1]$ is represented by the stream $d_0 : d_1 : \ldots$ where

$$d_n = \begin{cases} 1 & \text{if } \mathbf{tent}^n(x) > 0 \\ \perp & \text{if } \mathbf{tent}^n(x) = 0 \\ -1 & \text{if } \mathbf{tent}^n(x) < 0 \end{cases}$$

Note that $\mathbf{tent}^n(x) = 0$ can happen for at most one n.

Gray code requires partiality and non-determinism

By definition, (pure) Gray code is *partial*.

Moreover, as shown by Tsuiki, computation with Gray code requires *non-determinism*.

The intuitive reason is as follows:

- Because one digit of of Gray code may be undefined, a (Turing) machine reading or writing Gray code must have two heads, since one head might get stuck at an undefined digit.
- Since the two heads act independently the machine's behaviour is non-deterministic.

Logic for partiality and non-determinism

- In this talk, we present a *logic* that captures partiality and non-determinism in a constructive way. The logic allows for the *specification* and *extraction* of *provably correct* partial and non-deterministic programs.

- We apply this system to extract transformations between Gray code and the signed digit representations of real numbers.

- An earlier version of the system was presented at CSL 2016.

  B., Extracting Non-Deterministic Concurrent Programs. LIPICS 26

Related work: Extracting pre-Gray code

B., Kenji Miyamoto, Helmut Schwichtenberg, Hideki Tsuiki: Logic for Gray-code computation. In: Concepts of Proof in Mathematics, Philosophy, and Computer Science, de Gruyter, 2016.

gives a realizability interpretation and Minlog implementation of an intensional version of Gray code, called pre-Gray code, using a conventional constructive system and conventional program extraction.

Realizability

- We base program extraction from proofs on *realizability*.
- Realizability explains what it means to solve the computational problem expressed by a formula (can be viewed as a formalization of the BHK-interpretation of constructive logic).
- We let a, b, c range over (denotations of) programs, for example natural numbers viewed as codes of Turing machines, or elements of a Scott domain.
- For each formula A we define a formula

$$c\,\mathbf{r}\,A$$

to be read "c realizes A" or
"c solves the computational problem expressed by A".

Realizability for logic

$$
\begin{array}{llll}
c & \mathbf{r} & A & \equiv \quad A \quad (A \text{ nc, } c \text{ can be anything}) \\
c & \mathbf{r} & A \vee B & \equiv \quad (c = \mathrm{L}(a) \wedge a\,\mathbf{r}\,A) \vee (c = \mathrm{R}(b) \wedge b\,\mathbf{r}\,B) \\
c & \mathbf{r} & A \wedge B & \equiv \quad c = a : b \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B \quad (A, B \text{ not nc}) \\
c & \mathbf{r} & A \wedge B & \equiv \quad A \wedge c\,\mathbf{r}\,B \quad (A \text{ nc}) \\
c & \mathbf{r} & A \to B & \equiv \quad \forall\, a(a\,\mathbf{r}\,A \to c(a)\,\mathbf{r}\,B) \quad (A \text{ not nc}) \\
c & \mathbf{r} & A \to B & \equiv \quad A \to c\,\mathbf{r}\,B \quad (A \text{ nc}) \\
c & \mathbf{r} & \exists x\, A(x) & \equiv \quad \exists x\,(c\,\mathbf{r}\,A(x)) \\
c & \mathbf{r} & \forall x\, A(x) & \equiv \quad \forall x\,(c\,\mathbf{r}\,A(x))
\end{array}
$$

where a formula is *non-computational (nc)* if it doesn't contain $\vee$.

Realizability for inductive definitions (by example)

Assume operations and nc axioms for a real closed field R.

$$\mathbb{N}(x) \quad \stackrel{\mu}{=} \quad x = 0 \vee \exists y\,(\mathbb{N}(y) \wedge x = y + 1)$$

This defines $\mathbb{N}$ (inductively) as the least subset of R that contains 0 and is closed under successor.

Realizability for $\mathbb{N}$ is given by an analogous inductive definition:

$$\begin{aligned} n\,\mathbf{r}\,\mathbb{N}(x) \quad \stackrel{\mu}{=} \quad & (n = \mathrm{L} \wedge x = 0) \vee \\ & (n = \mathrm{R}(m) \wedge \exists y\,(m\,\mathbf{r}\,\mathbb{N}(y) \wedge x = y + 1)) \end{aligned}$$

Hence $n\,\mathbf{r}\,\mathbb{N}(x)$ means that n is a unary representation of the natural number $x \in R$.

First (trivial) example of program extraction

Theorem $\forall x, y\,(\mathbb{N}(x) \land \mathbb{N}(y) \to \mathbb{N}(x + y))$

From a (constructive) proof of this theorem one extracts a realizer f of the formula $\forall x, y\,(\mathbb{N}(x) \land \mathbb{N}(y) \to \mathbb{N}(x + y))$, that is

$$\forall x, y, n, m\,(n\,\mathbf{r}\,\mathbb{N}(x) \land m\,\mathbf{r}\,\mathbb{N}(y) \to f(n, m)\,\mathbf{r}\,\mathbb{N}(x + y))$$

Hence f computes addition on unary numbers.

Realizability for coinductive definitions (by example)

$\mathbb{I} = [1, -1] \subseteq R$
$\mathbb{I}_d = [d/2 - 1/2, d/2 + 1/2]$ for $d \in \mathrm{SD} = \{-1, 0, 1\}$

$$\mathrm{C}(x) \stackrel{\nu}{=} \bigvee_{d \in \mathrm{SD}} x \in \mathbb{I}_d \wedge \mathrm{C}(2x - d)$$

$$s \,\mathbf{r}\, \mathrm{C}(x) \stackrel{\nu}{=} \bigvee_{d \in \mathrm{SD}} s = d : s' \wedge x \in \mathbb{I}_d \wedge s' \,\mathbf{r}\, \mathrm{C}(2x - d)$$

Hence $s \,\mathbf{r}\, \mathrm{C}(x)$ means that s is an infinite stream of signed digits $d_0 : d_1 : \ldots$ such that

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

i.o.w., s is a signed digit representation of $x \in [-1, 1]$.

Second (slightly less trivial) example of program extraction

Theorem $\forall x, y\,(\mathrm{C}(x) \wedge \mathrm{C}(y) \to \mathrm{C}(x\,y))$

From a (constructive) proof of this theorem one extracts a realizer g of the formula $\forall x, y\,(\mathrm{C}(x) \wedge \mathrm{C}(y) \to \mathrm{C}(x\,y))$, that is

$$\forall x, y, s, t\,(s\,\mathbf{r}\,\mathrm{C}(x) \wedge t\,\mathbf{r}\,\mathrm{C}(y) \to g(n, m)\,\mathbf{r}\,\mathrm{C}(x\,y))$$

Hence g computes multiplication on signed digit representations.

Soundness

Program extraction is based on the soundness theorem for realizability which makes explicit the computational nature of constructive proofs:

Theorem (Soundness)

From a constructive proof of a formula A one can extract a program realizing A, that is, some term M (denoting a computation) such that $M\,\mathbf{r}\,A$ is provable.

The proof may use axioms which are either

- nc and true in the intended model, or else
- provided with realizers (for example induction axioms which are realized by recursion operators).

Constructivism for classical mathematicians

The Soundness Theorem may be a motivation for classical mathematicians to study constructive proofs, since there is a tangible advantage of constructive over classical proofs:

A constructive proof of A not only confirms that A is true, but also provides a solution to the computational problem expressed by A.

Origins of realizability

- Realizability was introduced by Kleene in 1945 for intuitionistic (constructive) number theory. His realizers are numbers encoding Turing machines or partial recursive functions.
- Kreisel introduced *modified realizability* for analysis (second-order number theory) in 1959. His realizers are continuous functionals of higher types.
- In the 1970s and 80s Kleene's number realizers were generalized to structures called *Partial Combinatory Algebras (PCAs)*, and since then many variants of realizability were studied.

Uses of realizability

The main uses of realizability are to

- make explicit the computational content of constructive mathematics,
- show the constructive unprovability of certain statements by showing that they are not realizable,
- provide models for constructive systems (including systems that are classically inconsistent),
- extract provably correct programs from constructive proofs.

Implementations and applications

Program extraction (via realizability or related methods) is implemented in many proof systems, e.g., Agda, Coq, Isabelle, Minlog, Nuprl.

Constructive analysis is a rich field of applications, but not the only one.

PE has also been applied to, for example,

- Lambda calculus (normalization by evaluation)
- Infinitary combinatorics (Higman's lemma)
- Parsing (monadic parser combinators)
- Imperative programming (in-place sorting)
- Satisfiability testing (extraction of a SAT solver)

Coinductive definition of Gray code

$$\mathrm{G}(x) \quad \overset{\nu}{=} \quad (x \neq 0 \to x \leq 0 \vee x \geq 0) \wedge \mathrm{G}(\mathbf{tent}(x))$$

$$s\,\mathbf{r}\,\mathrm{G}(x) \quad \overset{\nu}{=} \quad s = d : s' \text{ where}$$

$$(x \neq 0 \to d\ \mathbf{r}\ (x \leq 0 \vee x \geq 0)) \wedge s'\ \mathbf{r}\ \mathrm{G}(\mathbf{tent}(x))$$

Hence $s\ \mathbf{r}\ x \in \mathrm{G}$ iff s is a Gray code of x.

We wish to prove constructively $\mathrm{G} = \mathrm{C}$.

This will give us a computable equivalence of Gray code and signed digit representation.

$\mathrm{C} \subseteq \mathrm{G}$

Theorem. $\mathrm{C} \subseteq \mathrm{G}$.

Proof. By coinduction. We have to show

(1) $\mathrm{C}(x) \to x \neq 0 \to x \leq 0 \vee x \geq 0$ and

(2) $\mathrm{C}(x) \to \mathrm{C}(\mathbf{tent}(x))$.

For (1) we show $x \neq 0 \to \mathrm{C}(x) \to x \leq 0 \vee x \geq 0$, by Archimedean Induction (next slide).

(2) can be proved directly.

Archimedean Induction (AI)

$$\frac{\forall x \neq 0\,((|x| \leq 1/2 \to A(2x)) \to A(x))}{\forall x \neq 0\,A(x)}$$

Expresses that the partial order $([-1,1] \setminus \{0\}, \prec)$, where $y \prec x \equiv y = 2x$, is wellfounded, i.e. if $x \neq 0$ then $|2^n x| \geq 1$ for some $n \in \mathbb{N}$.

AI is realized by general recursion. This means, if f realizes the premise, i.e.

$$\forall x \neq 0\,\forall c\,((|x| \leq 1/2 \to c\,\mathbf{r}\,A(2x)) \to f(c)\,\mathbf{r}\,A(x))$$

then the least fixed point of f realizes $\forall x\,A(x)$, i.e.

$$\forall x\,\mathbf{fix}(f)\,\mathbf{r}\,A(x)$$

$\mathrm{G} \subseteq \mathrm{C}$?

We cannot expect to prove $\mathrm{G} \subseteq \mathrm{C}$ constructively, since this would give us a deterministic program to convert Gray code into signed digit representation, which is impossible by Tsuiki's analysis.

However, we *can* prove $\mathrm{G} \subseteq \mathrm{C}_2$, where C_2 is a non-deterministic variant of C.

In order to achieve this, we extend the logic by two new operators, one for *bounded non-determinism*, the other for *restriction*, a strict version of implication.

Bounded non-determinism

For every formula A we introduce a new formula $\mathrm{S}_n(A)$.
Realizability for $\mathrm{S}_n(A)$ is defined inductively.

$$
\begin{aligned}
a \,\mathbf{r}\, \mathrm{S}_n(A) \quad \stackrel{\mu}{=} \quad & a = \mathrm{Amb}(a_1, \ldots, a_m) \text{ where } m \leq n \wedge \\
& \exists i, b\, (a_i = \mathrm{Res}(b) \vee a_i = \mathrm{Cont}(b)) \ \wedge \\
& \forall i, b\, ((a_i = \mathrm{Res}(b) \to b \,\mathbf{r}\, A) \wedge \\
& \qquad (a_i = \mathrm{Cont}(b) \to b \,\mathbf{r}\, \mathrm{S}_n(A)))
\end{aligned}
$$

Amb is a variant of McCarthys *amb* operator

John McCarthy, *A Basis for a Mathematical Theory of Computation*,
IFIP Congress 62, N-H, 1963

The operational semantics of $\mathrm{Amb}(a_1, \ldots, a_m)$ is that the (potentially non-terminating) processes $a_1, \ldots, a_m$ are run in parallel. As soon as some a_i terminates, its result is taken and the other processes are abolished.

Restriction

For every formula A and nc formula B we introduce a new formula $A \,|\, B$ ("A restricted to B").

Realizability is defined as follows:

$$c \,\mathbf{r}\, (A \,|\, B) \quad \stackrel{\text{Def}}{=} \quad (B \to \exists a\, c = \mathrm{Res}(a)) \wedge \forall a\, (c = \mathrm{Res}(a) \to a \,\mathbf{r}\, A)$$

Compare this with realizability of $B \to A$ where B is nc and $A = A_0 \vee A_1$:

$$c \,\mathbf{r}\, (B \to A) \stackrel{\text{Def}}{=} B \to \exists a\, (c = \mathrm{L}(a) \wedge a \,\mathbf{r}\, A_0 \vee c = \mathrm{R}(a) \wedge a \,\mathbf{r}\, A_1)$$

The problem is that if, say, $\mathrm{L}(a)$ realizes $B \to A$, then it suggests that a realizes A_0, but we cannot conclude this unless B is true.

On the other hand, if $\mathrm{Res}(a)$ realizes $A \,|\, B$, then we are sure that a realizes A without knowing anything about B.

Non-deterministic signed digits

$$\mathrm{C}_2(x) \overset{\nu}{=} \mathrm{S}_2\Big(\bigvee_{d \in \mathrm{SD}} x \in \mathbb{I}_d \wedge \mathrm{C}_2(2x - d)\Big)$$

Theorem $\mathrm{G} \subseteq \mathrm{C}_2$.

Proof. By coinduction. To show

$$\mathrm{G}(x) \to \mathrm{S}_2(\exists d \in \mathrm{SD}\,(x \in \mathbb{I}_d \wedge \mathrm{G}(2x - d)))$$

This follows from the following lemmas.

Lemma 1. $\mathrm{G}(x) \wedge x \in \mathbb{I}_d \to \mathrm{G}(2x - d)$, for all $d \in \mathrm{SD}$.

Lemma 2. $(x \neq 0 \to x \leq 0 \vee x \geq 0) \leftrightarrow (x \leq 0 \vee x \geq 0 \mid x \neq 0)$.

Lemma 3. $\mathrm{G}(x) \to \mathrm{S}_2(\exists d \in \mathrm{SD}\, x \in \mathbb{I}_d)$.

The lemmas can be proven using the (realizable) rules on the next slide.

Logic for non-determinism

$$\frac{A}{\mathrm{S}_n(A)} \qquad \frac{\mathrm{S}_n(A) \quad A \to \mathrm{S}_n(A')}{\mathrm{S}_n(A')} \qquad \frac{A}{A \mid B} \qquad \frac{A \mid B \quad A \to (A' \mid B)}{A' \mid B}$$

$$\frac{A \mid B \quad B}{A} \qquad \frac{B \to A_0 \vee A_1 \quad \neg B \to A_0 \wedge A_1}{A_0 \vee A_1 \mid B} \; (A_0, A_1, B \text{ nc})$$

$$\frac{A \mid B \quad A \mid \neg B}{\mathrm{S}_2(A)}$$

Extracted program: $\mathbb{C} \subseteq \mathbb{G}$

```
stog s = f s : t1 (g s)  where

  f (-1:s) = -1
  f ( 1:s) =  1
  f ( 0:s) =  f s

  g (-1:s) =  s
  g ( 1:s) = -s
  g ( 0:s) =  1 : g s
```

hence

```
stog (-1:s) = -1 : stog s
stog ( 1:s) =  1 : nh (stog s)
stog ( 0:s) =  a : 1 : nh t   where a : t = stog s
```

Extracted program: $\mathrm{G} \subseteq \mathrm{C}_2$

```
gtos (-1:s)       = -1 : gtos s
gtos ( 1:b:s)     =  1 : gtos (swap b : s)
gtos ( a:1:c:s)   =  0 : gtos (a : swap c : s)
```