A logic for program extraction with bounded non-determinism

Ulrich Berger
Swansea University

j.w.w.

Hideki Tsuiki
Kyoto University

*Leeds, February 15, 2017*
Constructive mathematics goes back to Brouwer and Kolmogorov who promoted the idea that mathematical objects do not exist in some ideal Platonic world, but are constructions of the mind.

Therefore, the purpose of a proof in constructive mathematics is not establish truth in an ideal world, but to provide a mental construction of the objects involved.

As a consequence, some logical laws are rejected in constructive mathematics, the most famous example being the Law of Excluded Middle,

\[ A \lor \neg A \]
From mental constructions to computation

It was soon observed that mental constructions are closely related to computation. This led to the so-called Curry-Howard isomorphism or Proofs-as-Programs paradigm.

In order to technically exploit the Proofs-as-Programs paradigm it is better to map proofs to much simpler computational objects that only represent their relevant computational aspects. These objects are called realizers.
Realizability

Realizability was introduced by Kleene in 1945 for intuitionistic (constructive) number theory. Kleene defined when a number $e$ (encoding a Turing machine or a partial recursive functions) realizes a formula $A$, written

$$e \ r \ A$$

and showed that every intuitionisitc (constructive) proof yields such a realizer (Soundness Theorem).

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.

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)
- Computable analysis
Non-determinism in program extraction

Programs extracted from proofs are usually
- functional,
- possibly higher-order,
- deterministic.

In this talk we introduce a constructive logic that allows for the specification and extraction of provably correct non-deterministic programs.

We will give two applications in computable analysis:
- Gray code for real numbers
- Gaussian elimination

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

B., Extracting Non-Deterministic Concurrent Programs. LIPICS 26
Gray code for real numbers was introduced by Hideki Tsuiki in

*Real number computation through Gray code embedding.*


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

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

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

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

Note that $\text{tent}^n(x) = 0$ 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 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.
Related work: Extracting pre-Gray code


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.
Gaussian elimination

Given a non-singular (for simplicity quadratic) matrix

\[
\begin{pmatrix}
    a_{11} & \ldots & a_{1n} \\
    \vdots & \ddots & \vdots \\
    a_{n1} & \ldots & a_{nn}
\end{pmatrix}
\]

we pick a non-zero element \( a_{i1} \) in the first column as a pivot and apply elementary matrix operations to obtain

\[
\begin{pmatrix}
    a_{i1} & a_{i2} & \ldots & a_{in} \\
    0 & b_{22} & \ldots & b_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & b_{n2} & \ldots & b_{nn}
\end{pmatrix}
\]

Then we continue with the matrix \((bij)\).

The pivot element \( a_{i1} \) is selected non-deterministically.
Finding the pivot

The problem boils down to finding (constructively) in a non-zero vector of real numbers \((a_1, \ldots, a_n)\) a non-zero element \(a_i\).

The non-zeroness of the vector \((a_1, \ldots, a_n)\) means

\[
\neg(a_1 = \ldots = a_n = 0)
\]

Hence no computational information is contained in that statement.
Realizers

The domain of realizers is a Scott domain $D$ very similar to Scott’s $D_\infty$. It is defined by a recursive domain equation of the form

$$D = C_1(F_1(D)) + \ldots + C_n(F_n(D))$$

where the $+$ denotes the separated sum and the $F_i$ are certain continuous functors on the category of Scott domains with embedding projection pairs. The $C_i$ are constructors, that is, labels naming the components of the separated sum. The least element of $D$ is denoted $\bot$.

<table>
<thead>
<tr>
<th>constructor $C$</th>
<th>functor $F(D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nil</td>
<td>{\bot}</td>
</tr>
<tr>
<td>L</td>
<td>$D$</td>
</tr>
<tr>
<td>R</td>
<td>$D$</td>
</tr>
<tr>
<td>P</td>
<td>$D \times D$</td>
</tr>
<tr>
<td>F</td>
<td>$D \rightarrow D$</td>
</tr>
</tbody>
</table>
Realizability for logic

\[
\begin{align*}
{}^c r A & \equiv A \quad (A \text{ nc, } c \text{ can be anything}) \\
{}^c r A \lor B & \equiv (c = L(a) \land a r A) \lor (c = R(b) \land b r B) \\
{}^c r A \land B & \equiv c = P(a, b) \land a r A \land b r B \quad (A, B \text{ not nc}) \\
{}^c r A \land B & \equiv A \land c r B \quad (A \text{ nc}) \\
{}^c r A \rightarrow B & \equiv c = F(f) \land \forall a (a r A \rightarrow f(a) r B) \quad (A \text{ not nc}) \\
{}^c r A \rightarrow B & \equiv A \rightarrow c r B \quad (A \text{ nc}) \\
{}^c r \exists x A(x) & \equiv \exists x (c r A(x)) \\
{}^c r \forall x A(x) & \equiv \forall x (c r A(x))
\end{align*}
\]

where a formula is non-computational (nc) if it doesn't contain \lor.
Realizability for inductive definitions (by example)

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

$$\mathbb{N}(x) \equiv x = 0 \lor \exists y (\mathbb{N}(y) \land x = y + 1)$$

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:

$$n r \mathbb{N}(x) \equiv (n = L \land x = 0) \lor (n = R(m) \land \exists y (m r \mathbb{N}(y) \land x = y + 1))$$

Hence $n r \mathbb{N}(x)$ means that $n$ is a unary representation of the natural number $x \in R$. 
First (trivial) example of program extraction

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

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

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

Hence $f$ computes addition on unary numbers.

In general, we mean by a *proof* a constructive proof from nc-axioms about real numbers (e.g. the axioms of a real closed field).
Realizability for coinductive definitions (by example)

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

\[
\mathbf{C}(x) \overset{\nu}{=} \bigvee_{d \in SD} x \in \mathbb{I}_d \land \mathbf{C}(2x - d)
\]

\[
s \cdot \mathbf{C}(x) \overset{\nu}{=} \bigvee_{d \in SD} s = d : s' \land x \in \mathbb{I}_d \land s' \cdot \mathbf{C}(2x - d)
\]

where \( d : s' \) means \( P(d, s') \). Hence \( s \cdot \mathbf{C}(x) \) 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** $\forall x, y (C(x) \land C(y) \rightarrow C(x \cdot y))$

From a proof of this theorem one extracts a realizer $g$ of the formula $\forall x, y (C(x) \land C(y) \rightarrow C(x \cdot y))$, that is

$\forall x, y, s, t (s \text{ r } C(x) \land t \text{ r } C(y) \rightarrow g(n, m) \text{ r } C(x \cdot y))$

Hence $g$ computes multiplication on signed digit representations.
Coinductive definition of Gray code

\[
G(x) \triangleq (x \neq 0 \rightarrow x \leq 0 \lor x \geq 0) \land G(\text{tent}(x))
\]

\[
s \cdot r \ G(x) \triangleq s = d : s' \text{ where }
\]

\[
(x \neq 0 \rightarrow d \cdot r (x \leq 0 \lor x \geq 0)) \land s' \cdot r \ G(\text{tent}(x))
\]

Hence \( s \cdot r \ x \in G \) iff \( s \) is a Gray code of \( x \).

We wish to prove constructively \( G = C \).

This will give us a computable equivalence of Gray code and signed digit representation.
Theorem. \( C \subseteq G \).

Proof. By coinduction. We have to show

(1) \( C(x) \rightarrow x \neq 0 \rightarrow x \leq 0 \lor x \geq 0 \) and

(2) \( C(x) \rightarrow C(\text{tent}(x)) \).

For (1) we show \( x \neq 0 \rightarrow C(x) \rightarrow x \leq 0 \lor x \geq 0 \), by Archimedean Induction (next slide).

(2) can be proved directly.
**Archimedean Induction (AI)**

\[
\forall x \neq 0 ((|x| \leq 1/2 \to A(2x)) \to A(x))
\]

\[\forall x \neq 0 A(x)\]

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.

\[
\forall x \neq 0 \forall c ((|x| \leq 1/2 \to c \cdot r A(2x)) \to f(c) \cdot r A(x))
\]

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

\[\forall x \text{ fix}(f) \cdot r A(x)\]
Extracted program: \( \mathbf{C} \subseteq \mathbf{G} \)

\[
stog \ s = f \ s : stog \ (g \ s) \quad \text{where}
\]

\[
f \ (-1:s) = -1 \\
f \ (\ 1:s) = 1 \\
f \ (\ 0:s) = f \ s \quad \text{-- not productive, or guarded!}
\]

\[
g \ (-1:s) = s \\
g \ (\ 1:s) = -s \\
g \ (\ 0:s) = 1 : g \ s
\]

hence

\[
stog \ (-1:s) = -1 : stog \ s \\
stog \ (\ 1:s) = 1 : nh \ (stog \ s) \\
stog \ (\ 0:s) = a : 1 : nh \ t \quad \text{where} \ a : t = stog \ s
\]
We cannot expect to prove $G \subseteq C$ 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 $G \subseteq C_2$, 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 $S_n(A)$. Realizability for $S_n(A)$ is defined as follows:

$$a \text{ r } S_n(A) \overset{\text{Def}}{=} a = \text{Amb}(a_1, \ldots, a_n) \land$$
$$\exists i \text{Def}(a_i) \land \forall i((\text{Def}(a_i) \rightarrow a_i \text{ r } A))$$

Def$(a)$ means that $a$ begins with a constructor. The new constructor $\text{Amb}$ is a variant of McCarthy’s $\text{amb}$ operator


The operational semantics of $\text{Amb}(a_1, \ldots, a_n)$ is that the (potentially non-terminating) processes $a_1, \ldots, a_n$ 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:

$$a \text{r} (A | B) \overset{\text{Def}}{=} (B \rightarrow \text{Def}(a)) \land (\text{Def}(a) \rightarrow a \text{r} A)$$

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

$$a \text{r} (B \rightarrow A) \overset{\text{Def}}{=} B \rightarrow (a = L(a_0) \land a_0 \text{r} A_0) \lor (a = R(a_1) \land a_1 \text{r} A_1)$$

The problem is that if, say, $L(a_0)$ realizes $B \rightarrow A$, then it suggests that $a_0$ realizes $A_0$, but we cannot conclude this unless $B$ is true.

On the other hand, if $L(a_0)$ realizes $A | B$, then we are sure that $a_0$ realizes $A_0$ without knowing anything about $B$. 

Proof rules for $S_n(A)$ and $A \mid B$

\[
\begin{array}{c}
\frac{B}{S_n(B)} \\
\frac{A \rightarrow B}{S_n(A) \rightarrow S_n(B)}
\end{array}
\]

for strict $B$, that is formulas $B$ that are not realized by $\bot$.

\[
\begin{array}{c}
\frac{A}{A \mid B} \\
\frac{A \mid B \quad A \rightarrow (A' \mid B)}{A' \mid B} \\
\frac{A \mid B \quad B' \rightarrow B}{A \mid B'} \\
\frac{A \mid B \quad B}{A}
\end{array}
\]

\[
\begin{array}{c}
B \rightarrow A_0 \lor A_1 \\
\neg B \rightarrow A_0 \land A_1
\end{array}
\]

\[
\begin{array}{c}
(A_0 \lor A_1) \mid B
\end{array}
\]

where $A_0, A_1, B$ must be nc

\[
\begin{array}{c}
\frac{A \mid B \quad A \mid \neg B}{S_2(A)} \\
\frac{\neg (\neg B_1 \land \ldots \land \neg B_n) \quad A \mid B_1 \quad \ldots \quad A \mid B_n}{S_n(A)}
\end{array}
\]
Non-deterministic signed digits

\[ C_2(x) \ 
\overset{\nu}{=} \ S_2( \bigvee_{d \in SD} x \in I_d \land C_2(2x - d)) \]

**Theorem** \( G \subseteq C_2 \).

**Proof.** By coinduction. To show

\[ G(x) \rightarrow S_2(\exists d \in SD (x \in I_d \land G(2x - d))) \]

This follows from the following lemmas.

**Lemma 1.** \( G(x) \land x \in I_d \rightarrow G(2x - d) \), for all \( d \in SD \).

**Lemma 2.** \((x \neq 0 \rightarrow x \leq 0 \lor x \geq 0) \iff (x \leq 0 \lor x \geq 0 \mid x \neq 0)\).

**Lemma 3.** \( G(x) \rightarrow S_2(\exists d \in SD x \in I_d) \).
Extracted program: \( G \subseteq C_2 \)

\[
\begin{align*}
gtos (-1:s) & = -1 : gtos s \\
gtos (1:b:s) & = 1 : gtos (swap b : s) \\
gtos (a:1:c:s) & = 0 : gtos (a : swap c : s)
\end{align*}
\]

Note the overlapping patterns!
Cauchy reals

Moving on to Gaussian elimination, we work, for convenience, with a Cauchy-representation of real numbers:

\[ A(x) \overset{\text{Def}}{=} \forall n \in \mathbb{N} \exists q \in \mathbb{Q} \ |x - q| \leq 2^{-n} \]

A realizer of \( A(x) \) is a fast rational Cauchy-sequence converging to \( x \), i.e. a function \( f : \mathbb{N} \rightarrow \mathbb{Q} \) such that \( |x - f(n)| \leq 2^{-n} \) for all \( n \in \mathbb{N} \).
Gaussian elimination

We will extract Gaussian elimination from a proof of the following theorem:

**Gaussian Elimination Theorem.** For every non-singular $n \times n$-matrix with coefficients in $\mathbf{A}$ there exists, $n$-non-deterministically, an equivalent diagonal one with coefficients in $\mathbf{A}$.

As explained earlier the main problem is to find a non-zero vector of real numbers, $(a_1, \ldots, a_n)$, a non-zero element $a_i$, or even better, a positive rational number $q$ such that $|a_i| \geq q$.

Hence we define

$$x \not\equiv y \overset{\text{Def}}{=} \exists k \in \mathbb{N}(|x - y| \geq 2^{-k}) \quad (x \text{ is apart from } y)$$

A realizer of $x \not\equiv y$ is a natural number $k$ such that $|x - y| \geq 2^{-k}$. 


Picking the pivot

**Pivot Lemma**

\[ \forall x_1, \ldots, x_n \in A (\lnot (x_1 = \ldots = x_n = 0) \rightarrow S_n (x_1 \neq 0 \lor \ldots \lor x_n \neq 0)) \]

**Proof (sketch).** First one shows

\[ (*) \quad \forall x (A(x) \rightarrow (x \neq 0 \mid x \neq 0)) \]

using a variant of Archimedean Induction. Now fix arbitrary \( x_1, \ldots, x_n \in A \) with \( \lnot (x_1 = \ldots = x_n = 0) \). Using (*) and the monotonicity rules for restriction on obtains

\[ (x_1 \neq 0 \lor \ldots \lor x_n \neq 0) \mid x_i \neq 0 \]

for each \( i = 1, \ldots, n \). The result follows with non-deterministic disjunction elimination (slide 26).

**Extracted program (for \( n = 2 \))**

\[ \varphi(f, g) = \text{Amb}(L(\varphi_*(f)), R(\varphi_*(g))) \]
Monadic non-determinism

To finish the proof of the Gaussian Elimination Theorem, the Pivot Lemma has to be iterated. This means that we have to feed the result of one non-deterministic proof into another one. In other words, we need that $S_n$ is a monad.

But it isn’t!

To solve this problem we weaken $S_n$ by an inductive definition:

$$S^\omega_n(A) \overset{\mu}{=} S_n(A \lor S^\omega_n(A))$$

Realizability of $S^\omega_n(A)$ works out as follows:

$$a \triangleright S^\omega_n(A) \overset{\mu}{=} a = \text{Amb}(a_0, \ldots, a_n) \land \exists i \text{Def}(a_i) \land \forall i (\text{Def}(a_i) \rightarrow (a_i = L(b) \land b \triangleright A) \lor (a_i = R(b) \land b \triangleright S^\omega_n(A)))$$
Derivable rules for monadic non-determinism

\[
\frac{A}{S_n^\omega(A)} \quad \frac{S_n^\omega(A)}{A \rightarrow S_n^\omega(A')} \quad \frac{S_n(A)}{S_n^\omega(A)}
\]
Gaussian elimination, formally

The formal statement of Gaussian elimination is now:

**Gaussian Elimination Theorem.**

∀\(M \in A^{n \times n}(\text{non-singular}(M) \rightarrow S_\omega^n(\exists N \in A^{n \times n}(M \simeq N \land \text{diagonal}(N))))\)

Note that the non-singularity of \(M\) carries no computational information since it says that the column vectors of \(M\) can only trivially be linearly combined to the zero-vector.

Hence the extracted program takes as input only a matrix of Cauchy-sequences and produces the desired matrix whose coefficients are computed non-deterministically requiring at most \(n\)-processes to run in parallel.
Conclusion

We introduced two new logical operators that make it possible to specify nondeterministic programs at an abstract logical level and extract such programs together with formal correctness proofs.

This is part of a general project to extend program extraction programming paradigms beyond the purely functional. So far, besides non-determinism, also imperative programming has been considered: