Scott Domains for Denotational Semantics and Program Extraction

Ulrich Berger
Swansea University

Workshop Domains

Oxford, 7-8 July 2018

Overview

1. Domains
2. Computability
3. Denotational semantics
4. Program extraction
5. Brouwer's thesis
6. Concurrency and the law of excluded middle

Domains

From the abstract of Dana Scott's

DOMAINS FOR DENOTATIONAL SEMANTICS (1982)

"The purpose of the theory of domains is to give models for spaces on which to define computable functions. . . .

. . . There are several choices of a suitable category of domains, but the basic one which has the simplest properties is the one sometimes called *consistently complete algebraic cpo's*. . . ."

Scott domains

A *Scott domain* (*domain*, for short) is a partial order $(X, \sqsubseteq)$ with the following properties:

- There is a least element $\bot \in X$, and every directed set $A \subseteq X$ has a supremum $\sqcup A \in X$
  (X is a dcpo).
- Every bounded set $B \subseteq D$ has a supremum $\sqcup B \in X$
  (X is bounded complete).
- Every element of X is the directed supremum of compact elements, where $x \in X$ is called compact if whenever $x \sqsubseteq A$ for some directed set A, then $x \sqsubseteq a$ for some $a \in A$
  (X is algebraic).
- The set X_0 of compact elements of X is countable
  (X is countably based)

The *Scott topology* on X is generated by the basic open sets

$$\overset{\vee}{a} = \{x \in X \mid x_0 \sqsubseteq x\} \quad (x_0 \in X_0)$$

Continuous functions

A function $f : X \to Y$ is continuous (w.r.t. the Scott topolgy) iff it is monotone and respects directed suprema, that is,

- $\forall x, y \in X (x \sqsubseteq y \to f(x) \sqsubseteq f(y))$
- $f(\sqcup A) = \sqcup f[A]$ for every directed set $A \subseteq X$

The set $[X \to Y]$ of continuous functions from X to Y with the pointwise order is a domain.

By algebraicity,

$$f(x) = \sqcup \{y_0 \in Y_0 \mid \exists x_0 \in X_0, x_0 \sqsubseteq x, y_0 \sqsubseteq f(x_0)\}$$

Hence, continuous functions are no more complicated than domain elements: both are given by countable information.

The category of Scott domains

Scott domains and continuous functions form a cartesin closed category.

Cartesian closure essentially means the homeomorphism

$$[X \times Y \to Z] \simeq [X \to [Y \to Z]]$$

Due to the presence of $\perp$ the category of domains doesn't have co-products but there are 'approximations' to the co-product such as the separated sum $X + Y$ that adds a new bottom element to the disjoint sum of X and Y.

Fixed points

Fixed point combinator

Every continuous endofunction $f : X \to X$ has a least fixed point

$$Y(f) = \sqcup_{n \in \mathbb{N}} f^n(\bot) \in X$$

Moreover, $Y : [X \to X] \to X$ is continuous.

Recursive domain equations

In the category $\mathrm{DOM}^{\mathrm{e}}$ of domains with embeddings every continuous endofunctor has a least fixed point up to isomorphism.

Reflexive domains

Scott was the first to construct a non-trivial domain D_∞ isomorphic to its own function space:

$$D_\infty \simeq [D_\infty \to D_\infty]$$

This construction can be generalized using the fact that in $\mathrm{DOM}^{\mathrm{e}}$ the continuous function space operation

$$(X, Y) \mapsto [X \to Y]$$

is a continuous (co-variant!) functor in both arguments.

From *DOMAINS FOR DENOTATIONAL SEMANTICS*:

" . . . This category of domains is studied in this paper from a new, and it is to be hoped, simpler point of view incorporating the approaches of many authors into a unified presentation. Briefly, the domains of elements are represented set theoretically with the aid of structures called *information systems*. These systems are very familiar from mathematical logic, and their use seems to accord well with intuition. . . ."

Information systems

Information systems, roughly speaking, treat compact elements as the primary objects and view the points of a domain as a derived concept (ideals of compacts).

Advantages (from my point of view):

- No category theory needed.
- 'Information system equations' can be solved up to equality.
- Constructions like the universal domain become very easy.
- The finiteness of compact elements becomes obvious and equally obvious become the:
  - notion of a continuous function,
  - notion of a computable domain element,
  - effectiveness of domain constructions,
  - effectiveness of the solutions to recursive domain equations.

Information system considerably influenced the foundations of constructive mathematics (e.g. in point-free topology).

Beyond Scott domains

Many variants of domains have been studied.

Weakening the axioms allows for more domain constructions, e.g.

- continuous domains (real interval domain),
- SFP-domains (power domains),

. . . strengthening them or adding structure yields refinements, e.g.

- coherence spaces (linear logic/functions),
- stable domains (sequentiality)
- qualitative domains
- probabilistic domains
- richer topology (negative information, Lawson Topology)

Other directions, e.g.:

- Domain-theoretic models of exact real number computation
- Stone duality
- Synthetic domain theory
- Domain theory in logical form
- Equilogical spaces

Computability

$x \in X$ is *computable* if the set of its compact approximations

$$\{x_0 \in X_0 \mid x_0 \sqsubseteq x\}$$

is recursively enumerable (w.r.t. some coding of the compact elements).

Ershov (1977) related this notion of computability to his theory of numberings and showed its remarkable robustness:

- The computable elements of a domain admit a principle numbering.
- Rice-Shapiro Theorem (1959): A set of computable domain elements is completely enumerable iff it is effectively open.
- Myhill-Sheperdson Theorem (1959): A function on the computable elements of a domain is an effective operation iff it is effectively continuous

Partial continuous functionals

Due to cartesian closure, domains provide a natural model of partial higher-type functionals:

$$D(0) = \mathbb{N}_\perp = \text{the flat domain of natural numbers.}$$
$$D(\rho \to \sigma) = [D(\rho) \to D(\sigma)]$$

Plotkin 1977: A partial continuous functional is effectively continuous (computable as a domain element) iff it can be defined in PCF (basic arithmetic, λ-calculus, recursion (Y)) extended by the functions

- *parallel or* ($1 \tilde{\vee} \perp = \perp \tilde{\vee} 1 = 1$, $0 \tilde{\vee} 0 = 0$)
- *continuous existential* ($\exists^2(f) = 1$ if $f(n) = 1$, $\exists^2(f) = 0$ if $f(\perp) = 0$).

Total continuous functionals

Ershov 1977: The hereditarily total continuous functionals coincide with the Kleene-Kreisel countable/continuous functionals.

Kreisel-Lacombe-Shoenfield 1959/Ershov 1976: The hereditarily computably total continuous functionals coincide with hereditarily effective operations (HEO). See also Spreen/Young 1984 for this result in a topological setting.

Normann 2000: A total continuous functional is computable (as a domain element) iff it is PCF-definable.

Program semantics

Consider a programming language with a given *operational semantics*, e.g. LCF.

Denotational semantics interprets a program M as an element $[\![M]\!]$ of a domain.

Goals:

- **Computational Adequacy:** If $[\![M]\!] = d$ for some *data*, that is, discrete defined value d, then the computation of M terminates with result d.

- **Full abstraction** If M and N are operationally equivalent in all contexts, then $[\![M]\!] = [\![N]\!]$.

Program semantics (some results)

- Plotkin 1977: Scott domains are computationally adequate for PCF $+ \tilde{\forall} + \exists^2$ with a call-by-name operational semantics.

- Plotkin 1977: Scott domains are fully abstract for PCF $+ \tilde{\forall}$.

  [Proof: The functions $[\![M]\!]$ and $[\![N]\!]$ are continuous and hence complete determined by their values at compact arguments. The latter are definable in PCF$+\tilde{\forall}$]

- Fully abstract models of PCF: Milner (syntactic, 1977), Abramsky, Jagadeesan, Malacaria, Hyland, Ong, Nickau (games, 1994), Bucciarelli, Ehrhard, Curien, Berry, Jung, Stoughton, McCusker, ...

Program semantics (II)

- B 2005: Strong computational adequacy for PCF with strict domain semantics: If $[\![M]\!]_{\mathrm{s}} \neq \bot$, then M is strongly normalizing.
- Coquand, Spiwack 2006: Strong computational adequacy for Dependent Type Theory using a reflexive domain.
- B 2009/2018: Computational adequacy for extensions of type free PCF using a suitable reflexive domain.

The proofs use compact domain elements as a substitute for finite types. Induction on types is replaced by induction on the **rank** of a compact element, $\mathbf{rk}(x_0) \in \mathbb{N}$.

(1) $\mathbf{rk}(\mathbf{Pair}(x_0, x_1)) > \mathbf{rk}(x_i)$.

(2) $\mathbf{rk}(\mathbf{Fun}(f_0)) > \mathbf{rk}(f_0(x))$ and $f_0(x) = f_0(x_0)$ for some compact $x_0 \sqsubseteq x$.

Program Extraction

The *Curry-Howard correspondence* states that intuitionistic proofs correspond to programs.

Kleene's realizability:

From a proof of a formula A one can extract a number e such that $\{e\}$ *realizes* A.

($\{e\}$ is the partial recursive fuction with index e)

We work with a similar notion of realizability but our realizers are elements of the domain

$$D \simeq \mathbf{Nil}\{\} + \mathbf{Pair}(D \times D) + \mathbf{Fun}([D \to D]).$$

Soundness

Soundness Theorem

From a proof of A one can extract a program M (in untyped PCF) such that $[\![M]\!]$ ($\in D$) realizes A.

Proof. Induction on proofs using the equational theory of D and the denotational semantics of programs.

Program Extraction Theorem

From a proof of a Σ-formula A one can extract a program M evaluating to a data d realizing A.

Proof. Take the program from the Soundness Theorem and apply computational adequacy.

Induction and Coinduction

Traditionally, program extraction via realizability is done in intuitionistic number theory (HA or HA^ω).

For applications it is better to include abstract spaces, specified by disjunction-free axioms, and *inductive and coinductive definitions* as least and greatest fixed points of strictly positive operators:

$$\frac{}{\Phi(\mu(\Phi)) \subseteq \mu(\Phi)}\ \mathsf{cl} \qquad \frac{\Phi(P) \subseteq P}{\mu(\Phi) \subseteq P}\ \mathsf{ind}$$

$$\frac{}{\nu(\Phi) \subseteq \Phi(\nu(\Phi))}\ \mathsf{cocl} \qquad \frac{P \subseteq \Phi(P)}{P \subseteq \nu(\Phi)}\ \mathsf{coind}$$

If s realizes $\Phi(P) \subseteq P$, then $a \overset{\mathrm{rec}}{=} s \circ (\mathbf{mon}_\Phi\, a)$ realizes $\mu\,\Phi \subseteq P$.
If s realizes $P \subseteq \Phi(P)$, then $a \overset{\mathrm{rec}}{=} (\mathbf{mon}_\Phi\, a) \circ s$ realizes $P \subseteq \nu\,\Phi$.

Infinite data

Realizers of coinductive definitions are infinite data like streams. These exist in D, due to consistent completeness, as suprema of finite lists with $\bot$ at the end, e.g.

$$\mathbf{Pair}(d_0, \mathbf{Pair}(d_1, \mathbf{Pair}(d_2, \bot)))$$

For example, [1]

$$\mathrm{C}(x) \overset{\nu}{=} \exists d \in \{-1, 1, 1\}(|x| \leq 1 \wedge \mathrm{C}(2x - d))$$

defines a predicate on the compact interval [-1,1] such that $a \in D$ realizes $\mathrm{C}(x)$ iff a is a signed digit representation of x.

This coinductive style of formalization has the advantage that infinite streams do not need to be formalized (realizability and domains take care of this in the background).

[1] Officially, $\mathrm{C} = \nu\Phi$ where $\Phi(X) = \lambda x\, \exists d \in \{-1, 1, 1\}(|x| \leq 1 \wedge X(2x - d))$

PE in constructive analysis

Based on coinductive specifications of real numbers and continuous real functions parts of constructive analysis have been fomalized, implemenmted and programs extracted.

See for example:

B, Kenji Miyamoto, Helmut Schwichtenberg, Monika Seisenberger. *Minlog - A Tool for Program Extraction for Supporting Algebra and Coalgebra.* LNCS 6859, 2011.

Fredric Forsberg, Kenji Miyamoto, Helmut Schwichtenberg. *Program Extraction from Nested Definitions.* LNCS 7988, 2013.

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

B., Dieter Spreen. *A Coinductive Approach to Computing with Compact Sets.* Journal of Logic and Analysis 8, 2016.

Computational content

To measure complexity one counts

- alternations of quantifiers in *classical logic*;
- nestings of universal implications in *intuitionistic logic*;
- alternations of bounded quantifiers in computational complexity.

These measures are of little relevance in PE since even very low classes contain (practically) infeasible programs.

What matters more is which formulas have computational content.

A general strategy is to organize axioms and rules into

- a few with computational content (for which we need realizers);
- a majority without computational content (which only need to be true).

The latter are characterized by the lack of disjunctions at strictly positive positions (a version of Harrop formulas).

Brouwer's Thesis

A predicate P on natural numbers is a *bar* if

$$\forall \alpha \in \mathbb{N}^{\mathbb{N}} \, \exists n \in \mathbb{N} \, P(\bar{\alpha}\, n).$$

P is an *inductive bar* if (the code of) the empty sequence is *inductively barred* by P where s being inductively barred by P ($\mathbf{IB}_P(s)$) is inductively defined by the rules:

(i) if $P(s)$, then $\mathbf{IB}_P(s)$,

(ii) if $\forall n \in \mathbb{N}\, \mathbf{IB}_P(s * n)$ then $\mathbf{IB}_P(s)$.

In other words $\mathbf{IB}_P(s) \stackrel{\mu}{=} P(s) \vee \forall n \in \mathbb{N}\, \mathbf{IB}_P(s * n)$.

Brouwer's Thesis (BT): Every bar is inductive.

An immediate consequence of Brouwer's thesis is **Bar Induction**.

Under additional conditions on P (decidability, or monotonicity), these prinicples are generally accepted as intuitionisitcally valid, at least, they are realizable.

Problems with **BT**

Brouwer's thesis

$$\forall \alpha \in \mathbb{N}^{\mathbb{N}}\, \exists n \in \mathbb{N}\, P(\bar{\alpha}\, n) \;\to\; \mathbf{IB}_P(\langle\rangle)$$

has a few disadvantages (from the viewpoint of PE):

1. It has computational content, which, however, is useless.
2. It is confined to arithmetic.
3. Its premise and conclusion are both too strong for many applications.

An abstract and non-computational version of **BT**

The *accessible part* of a binary relation $\prec$, is defined inductively by

$$\mathbf{Acc}_{\prec}(x) \stackrel{\mu}{=} \forall y \prec x\, \mathbf{Acc}_{\prec}(y)$$

Dually, the property of *having a path through* $\prec$ is defined coinductively by

$$\mathbf{Path}_{\prec}(x) \stackrel{\nu}{=} \exists y \prec x\, \mathbf{Path}_{\prec}(y)$$

The *wellfounded part* of $\prec$ is defined as the complement of $\mathbf{Path}_{\prec}$,

$$\mathbf{Wf}_{\prec}(x) \stackrel{\mathrm{Def}}{=} \neg\mathbf{Path}_{\prec}(x)$$

Connection with **BT**: Set $s \prec t \stackrel{\mathrm{Def}}{=} \neg P(s) \wedge \exists n \in \mathbb{N}\, s = t * n$.

- If P is decidable, then $\mathbf{IB}_P = \mathbf{Acc}_{\prec}$.
- Classically, if $\mathbf{Wf}_{\prec}(\langle\rangle)$, then P is a bar.

Hence, we propose the axiom:

$\mathbf{BT}_0 \quad \forall x\, (\mathbf{Wf}_{\prec}(x) \to \mathbf{Acc}_{\prec}(x))$

Virtues of $\mathbf{BT}_0$

$\mathbf{BT}_0 \quad \forall x\,(\mathbf{Wf}_{\prec}(x) \to \mathbf{Acc}_{\prec}(x))$

1. is true but non-computational;
2. is general (not confined to arithmetic) and doesn't mention infinite sequences;
3. has a weaker premise than $\mathbf{BT}$, thus permitting more applications.

Bar Induction

Brouwer's Bar induction, **BI** (follows from **BT**): If

(0) P is decidable (or monotone),

(1) P is a bar,

(2) $P \subseteq Q$,

(3) $\forall s \in \mathbb{N}\,(\forall n \in \mathbb{N}\, Q(s * n) \;\rightarrow\; Q(s))$,

then $Q(\langle\rangle)$.

Abstract Bar induction, $\mathbf{BI}_0$ (follows from $\mathbf{BT}_0$): Let $\prec^*$ be the reflexive transitive closure of $\prec$, let 0 be arbitrary. If

(1) $\mathbf{Wf}_{\prec}(0)$,

(2) $\forall x \prec^* 0\,(\neg P(x) \vee Q)$,

(3) $\forall x \prec^* 0\,(\forall y \prec x Q(y) \;\rightarrow\; Q(x))$,

then $Q(0)$.

Further useful consequences of $\mathbf{BT}_0$

Markov's principle: If P is decidable, then $\neg\neg\exists n \in \mathbb{N}\, P(n) \;\to\; \exists n \in \mathbb{N}\, P(n)$

[set in $\mathbf{BT}_0$, $m \prec n :\overset{\text{Def}}{=} \neg P(n) \wedge m = n+1$ and $Q \overset{\text{Def}}{=} \exists n \in \mathbb{N}\, P(n)$]

Wellfounded induction: If $\forall x(\forall y \prec x\, P(y) \;\to\; P(x))$, then $\forall x \in \mathbf{Wf}_{\prec} P(x)$.

Archimedean induction: If $\forall x \neq 0\,((|x| \leq 1 \to Q(2x)) \to Q(x))$, then $\forall x \neq 0\, Q(x)$.

[set $y \prec x :\overset{\text{Def}}{=} |x| \leq 1 \wedge y = 2x$ and use the (non-computational) Archimedean property $(\forall n \in \mathbb{N}|x| \leq 2^{-n} \;\to\; x = 0)$ to show that $\forall x \neq 0\, \mathbf{Wf}_{\prec}(x)$.]

Feeling: All useful realizable principles can be split into a non-computational part (such as $\mathbf{BT}_0$) and an instance of induction or coinduction.

Concurrency and LEM (j.w.w. Hideki Tsuiki)

The starting point for this work was:

Hideki Tsuiki. Real Number Computation through Gray Code Embedding. TCS 284(2):467–485, 2002.

- Infinite Gray code for real numbers admits one undefined digit.
- This requires programs with two concurrently operating reading heads with possibly nondeterministic results (IM2 machines).
- Can such programs be extracted from proofs?

Parallelism in Exact Real Number Computation

- Potts, Edalat, Escardo noticed in 1997 that computing with the interval domain as a model of real numbers appears to require a parallel if-then-else operation.

- In fact, this parallelism is unavoidable (Escardo, Hofmann, Streicher, 2004).

  Computing with TTE representations (e.g. Cauchy- or signed digit representation) does *not* require parallelism, while Gray code (though very similar to signed digits) requires parallelism.

- Denotational models of nondeterministic computation are well-known in Domain Theory (starting with Plotkin's powerdomain 1976) and Relational Semantics (Bucciarelli, Ehrhard, Manzonetto 2011).

Concurrency and partiality

Given: Processes p_1, p_2 such that

- at least one p_i is guaranteed to terminate,
- each terminating p_i will produce a correct result

Task: Combine the p_i to obtain a correct result.

Solution: Run p_1, p_2 concurrently. As soon as one p_i terminates, deliver the result and kill p_{3-i}.

We will introduce an extension of intuitionistic logic enabling the extraction of such kind of programs (together with correctness proofs).

Logic and program for concurrency

- We add a new formula construct $A_1 \overset{\mathsf{p}}{\vee} A_2$ which admits concurrent processes as realizers ...
- ... and add a new program constructor $\mathbf{Amb}(a_1, a_2)$ for the concurrent execution of the processes a_i (motivated by McCarthy's Amb).
- $\mathbf{Amb}(a_1, a_2)$ realizes $A_1 \overset{\mathsf{p}}{\vee} A_2$ iff at least one a_i is defined and, if defined, a_i realize A_i.

The law of excluded middle (LEM) as a disjunction introduction rule

$$\frac{}{B \vee \neg B}\ \text{LEM}$$

is equivalent to

$$\frac{B \to A_1 \qquad \neg B \to A_2}{A_1 \vee A_2}\ \text{LEMD}$$

(for LEMD $\Rightarrow$ LEM set $A_1 \stackrel{\text{Def}}{=} B$, $A_2 \stackrel{\text{Def}}{=} \neg B$)

Concurrent law of excluded middle (failed attempt)

The following form of the law of excluded middle seems to be realizable provided B is non-computational:

$$\frac{B \to A_1 \quad \neg B \to A_2}{A_1 \overset{\mathsf{p}}{\vee} A_2}$$

If	$a_1 \,\mathbf{r}\, (B \to A_1)$,	which means	$B \to a_1 \,\mathbf{r}\, A_1$,
and	$a_2 \,\mathbf{r}\, (\neg B \to A_1)$,	which means	$\neg B \to a_2 \,\mathbf{r}\, A_2$

one might believe (classically) that $\mathbf{Amb}(a_1, a_2)$ realizes $A_1 \overset{\mathsf{p}}{\vee} A_2$.

However, if, for example, B is false, then the formula $B \to a_1 \,\mathbf{r}\, A_1$ says nothing about a_1, but a_1 might still be defined and be delivered as a result of $\mathbf{Amb}(a_1, a_2)$.

Hence, there is no guarantee that $\mathbf{Amb}(a_1, a_2)$ realizes $A_1 \overset{\mathsf{p}}{\vee} A_2$.

We need a variant of implication that avoids this.

Restriction $A \parallel B$ (a variant of $B \to A$)

$$a\,\mathbf{r}\,(A \parallel B) \stackrel{\text{Def}}{=} (B \to \mathbf{Def}(a)) \wedge (\mathbf{Def}(a) \to a\,\mathbf{r}\,A)$$

where B is nc and $\mathbf{Def}(a)$ means that a is defined (i.e. $\{a\}$ terminates). Realizable rules:

$$\frac{A}{A \parallel B} \qquad \frac{A \parallel B \quad A \to (A' \parallel B)}{A' \parallel B} \qquad \frac{A \parallel B \quad B' \to B}{A \parallel B'}$$

$$\frac{A \parallel B \quad B}{A} \qquad \frac{\neg B}{A \parallel B}$$

$$\frac{B \to (A_0 \vee A_1) \quad \neg B \to (A_0 \wedge A_1)}{(A_0 \vee A_1) \parallel B}$$

where A_0, A_1 must be nc

Concurrent law of excluded middle (correct)

$$\frac{A_1 \parallel B \quad A_2 \parallel \neg B}{A_1 \overset{\mathsf{p}}{\vee} A_2}$$

If a_1 realizes $A_1 \parallel B$ and a_2 realizes $A_2 \parallel \neg B$,

then $\mathbf{Amb}(a_1, a_2)$ realizes $A_1 \overset{\mathsf{p}}{\vee} A_2$.

Program extraction revised

Programs are extended by a construct for nondeterminism or concurrency $\mathbf{Amb}(a_1, a_2)$.

Our domain-theoretic denotational semantics interprets $\mathbf{Amb}(a_1, a_2)$ simply as a pair (with a marker to distinguish it from $\mathbf{Pair}(a_1, a_2)$ (no powerdomains needed).

The interpretation of $\mathbf{Amb}(a_1, a_2)$ as nondeterminism is only reflected in the operational semantics (see next slide).

Concurrent Program Extraction Theorem

From a proof of a data formula A one can extract a terminating program M such that whenever M reduces to a data d, then d realizes A^-.

Where:

- data formulas roughly correspond to Σ^0_1-formulas,
- A^- is obtained from A by replacing $\overset{\mathsf{p}}{\vee}$ by $\vee$ and $\|$ by $\leftarrow$.

Operational semantics of **Amb**

$$\text{(i)}\quad \frac{c \longrightarrow (C(M_1,\ldots,M_k),\eta) \qquad (M_i,\eta) \Longrightarrow d_i \quad (i=1,\ldots,k)}{c \Longrightarrow C(d_1,\ldots,d_k)}\ (C \text{ a data constructor})$$

$$\text{(ii)}\quad \frac{c \longrightarrow (\mathbf{Amb}(M,N),\eta) \qquad (M,\eta) \Longrightarrow d}{c \Longrightarrow \mathbf{L}(d)} \qquad \frac{c \longrightarrow (\mathbf{Amb}(M,N),\eta) \qquad (N,\eta) \Longrightarrow d}{c \Longrightarrow \mathbf{R}(d)}$$

$c \longrightarrow c'$ is the usual (deterministic) call-by-name big-step head reduction of closures $c = (M,\eta)$ treating **Amb** like an ordinary pairing constructor.

$c \Longrightarrow d$ is a non-deterministic 'print' relation that completely normalizes under constructors.

Infinite Gray code for exact real numbers

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

From Gray code to signed digits

Gray code has the remarkable property that each real number $x \in [-1, 1]$ has exactly one representation.

In contrast, the well-known *signed representation*, which represents a real number $x \in [-1, 1]$ by an infinite stream of digits $d_i \in \mathrm{SD} \stackrel{\mathrm{Def}}{=} \{-1, 0, 1\}$ such that

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

is highly redundant (as are all other known admissible total representations of the reals).

We sketch how to extract a concurrent program that translates infinite Gray code into signed representation.

From Gray code to signed digit representation

We write $\mathrm{S}(A)$ for $A \overset{\mathsf{p}}{\vee} A$.

$$\begin{aligned} \mathrm{SD} &\overset{\mathrm{Def}}{=} \{-1, 0, 1\} \\ \mathbb{I}_d &\overset{\mathrm{Def}}{=} [d/2 - 1/2, d/2 + 1/2] \end{aligned}$$

$$\begin{aligned} \mathrm{C}(x) &\overset{\nu}{=} \exists d \in \mathrm{SD}\,(x \in \mathbb{I}_d \wedge \mathrm{C}(2x - d)) \\ \mathrm{C}_2(x) &\overset{\nu}{=} \mathrm{S}(\exists d \in \mathrm{SD}\,(x \in \mathbb{I}_d \wedge \mathrm{C}_2(2x - d))) \end{aligned}$$

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

$s\,\mathbf{r}\,\mathrm{C}$ iff s is a signed digit representation of x.

$s\,\mathbf{r}\,\mathrm{G}$ iff s is an infinite Gray code of x.

$s\,\mathbf{r}\,\mathrm{C}_2$ iff s is a non-deterministic signed digit rep. of x.

$\mathrm{C} \subseteq \mathrm{G}$ is easy. Our main goal is to show $\mathrm{G} \subseteq \mathrm{C}_2$.

$\mathrm{G} \subseteq \mathrm{C}_2$ (main step)

Assume $\mathrm{G}(x)$.

$A(x) \overset{\mathrm{Def}}{=} \exists d \in \mathrm{SD}\, x \in \mathbb{I}_d$

We show $\mathrm{S}(A(x))$, i.e. the first digit of the signed digit representation exists, nondeterministically.

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

$$\dfrac{\dfrac{\dfrac{\mathrm{G}(x)}{x \neq 0 \to (x \leq 0 \vee x \geq 0)} \quad x = 0 \to (x \leq 0 \wedge x \geq 0)}{x \leq 0 \vee x \geq 0 \parallel x \neq 0}}{A(x) \parallel x \neq 0} \qquad \dfrac{\dfrac{\dfrac{\mathrm{G}(x)}{\mathrm{G}(\mathbf{tent}(x))}}{\vdots}}{A(x) \parallel x = 0}$$

$$\mathrm{S}(A(x))$$

Conclusion

Domain theory is a work horse in program semantics and program extraction.

Without domains most of the work in program semantics and program extraction would have not been possible.

Thanks

Dear Professor Scott

Thanks for giving us domains!

Happy Birthday !!