Realizability: a short course

Chambéry, June 2011

John Longley
Laboratory for Foundations of Computer Science
University of Edinburgh

Outline of the course

I will structure the course as four ‘lectures’ of 1.5 hours each.

The first lecture will be a broad introduction and overview of the field. The other three will be rather more specialized, reflecting my personal interest in ‘notions of computability’.

1. The many faces of realizability.

2. A realizability framework for models of computation.

3. Realizability and higher type computability.

4. Models of sequential computation.

Lecture 1: The many faces of realizability

There's no general definition of what realizability is. Rather, there are a bunch of things known as 'realizability interpretations' with a common flavour. One understands the general (informal) concept of realizability by seeing some examples.

In this lecture, we'll look at realizability interpretations for . . .

- logics such as Heyting Arithmetic (where it all began),
- type systems such as Girard's System F,
- programming languages such as Plotkin's PCF.

In the remaining lectures, I'll develop a fourth strand, applying realizability to the study of 'models of computation' in general.

Origins of realizability: Clarifying intuitionistic meaning

In the 1920s, L.E.J. Brouwer expounded his intuitionistic philosophy of mathematics. According to Brouwer, the 'meaning' of mathematical statements resided in mental constructions, rather than in their reference to a supposed platonic reality.

Many found Brouwer's exposition of his ideas obscure and lacking in the clear definitions one is used to in mathematics.

The idea of realizability was first introduced by Kleene in a paper of 1945. One can see Kleene's work as an attempt to recast (some aspects of) Brouwer's thought in more accessible terms. (Though Kleene never mentions Brouwer in the paper!)

The Brouwer-Heyting-Kolmogorov 'interpretation'

Consider the language of first order logic (say for the theory of natural numbers, with $0, S, +, *, =$). In intuitionism, to say what a (closed) formula 'means' is to say what counts as a 'proof' of it. Somewhat informally, we can say:

- An atomic formula has a trivial 'proof' iff it's (verifiably) true.
- If p, q are proofs of P, Q respectively, the pair (p, q) is a proof of $P \wedge Q$.
- If p is a proof of P then $(0, p)$ is a proof of $P \vee Q$. Likewise for $(1, q)$.
- A proof of $P \Rightarrow Q$ is a constructive operation that transforms any proof of P into a proof of Q $(\ldots)$.
- A proof of $\exists x. P$ is a pair (n, p) where p is a proof of $P[n/x]$.
- A proof of $\forall x. P$ is a constructive operation that transforms a value n into a proof of $P[n/x]$ $(\ldots)$.
- There's no proof of $\perp$ (falsity).

Kleene '1945' realizability

With a little hindsight, we can see Kleene's definition as one way of making the BHK interpretation precise.

The informal notion of an constructive operation is replaced by the precise notion of a Church-Turing computable function.

To make this idea work, proofs are replaced by natural numbers. (N.B. Kleene hadn't invented higher type computability yet!)

So fundamentally, Kleene is defines a relation:

$n \Vdash P$ (the number n *realizes* the formula P)

Kleene exploits the existence of *pairing* and *application* operations on $\mathbb{N}$. E.g.

$$\langle n, m \rangle = 2^n.3^m$$

$$n \bullet m = \text{result (if any) of running } n\text{th Turing machine on } m.$$

Kleene's definition

- For P atomic, $0 \Vdash P$ iff P is true.
- $n \Vdash P \wedge Q$ iff $n = \langle p, q \rangle$ where $p \Vdash P$, $q \Vdash Q$.
- $n \Vdash P \vee Q$ iff $n = \langle 0, p \rangle$ where $p \Vdash P$ or $n = \langle 1, q \rangle$ where $q \Vdash Q$.
- $n \Vdash P \Rightarrow Q$ iff for all $m \Vdash P$, $n \bullet m$ is defined and $n \bullet m \Vdash Q$.
- $n \Vdash \exists x.P$ iff $n = \langle m, p \rangle$ where $p \Vdash P[m/x]$.
- $n \Vdash \forall x.P$ iff for all m, $n \bullet m$ is defined and $n \bullet m \Vdash P[m/x]$.
- $n \Vdash \bot$ never.

We say P is *realizable* if some $n \Vdash P$.

Realizability and intuitionism

Kleene realizability doesn't buy us anything philosophically as an explication of intuitionistic 'meaning': how are we meant to 'understand' the definition of $\Vdash$?

Rather, realizability is a technical tool that's useful for investigating questions of intuitionistic provability.

Any sentence provable in Heyting Arithmetic (say) is realizable (easy induction on structure of HA proofs).

But not conversely. E.g. consider

$$\forall n.\, H(n) \vee \neg H(n)$$

where $H(n)$ expresses '$n \bullet n$ is defined', and $\neg P$ means $P \Rightarrow \bot$.

This is unrealizable, because the halting problem is undecidable. So the negation of the above formula *is* realizable, though it's clearly unprovable even in Peano Arithmetic.

(There are even purely propositional examples — Rose 1953.)

Unprovability and consistency results

Since HA-provable sentences $\overset{\neq}{\subset}$ Kleene-realizable sentences, we can sometimes use realizability to show that a given sentence is unprovable in HA. Example: $\forall n.\, H(n) \vee \neg H(n)$.

What's more, we can turn the mismatch into advantage. certain *classically false* principles are seen to be consistent with HA.

Example: the following sentence, known as Church's Thesis CT_0.

$$(\forall n.\exists m.\, P(n,m)) \;\Rightarrow\; (\exists k.\forall n.\, `P(n, k \bullet n)\text{'})$$

Note that such principles were actually accepted by the Russian school of constructive mathematics (Markov *et al.*)!

Unprovability/consistency results are typical proof-theoretic applications of realizability.

More subtle view: Relative consistency and conservativity

Any consistency proof is ultimately a *relative* consistency proof.

Everything I've said so far can itself be formalized within HA. So if HA+CT_0 turned out to be inconsistent, then HA would be inconsistent! Moreover, our argument yields an explicit effective method Δ for transforming proofs:

$$\pi \text{ proves } \perp \text{ in HA+CT}_0 \;\Rightarrow\; \Delta(\pi) \text{ proves } \perp \text{ in HA}$$

That can be proved in *very* weak systems (PRA or even less)!

With a bit more work, we can replace $\perp$ here by e.g. any $\exists$-free formula. So HA+CT_0 is conservative over HA for this class of formulae.

Another proof-theoretic application

A typical feature of intuitionistic systems like HA is the existence property: if $\vdash \exists x.P$, then there's some m such that $P[m/x]$.

Realizability gives a nice proof of this for HA. Let's tweak the definition of $\Vdash$ slightly in the $\Rightarrow$ case:

- $n \Vdash P \Rightarrow Q$ iff for all $m \Vdash P$, $n \bullet m$ is defined and $n \bullet m \Vdash Q$, and $P \Rightarrow Q$ is also true.

This ensures that $(n \Vdash P) \Rightarrow P$ (and HA can prove this). (*)

Suppose now HA $\vdash \exists x.P$. Then for some particular n, we have $n \Vdash \exists x.P$ (and HA proves this). So $n = \langle m, p \rangle$ where $p \Vdash P[m/x]$ (and HA proves this). So by (*), HA $\vdash P[m/x]$.

Realizability for logics: the general pattern

The definition and applications of Kleene '1945' realizability are the prototype for all other realizability interpretations of logic. The general pattern is that one defines a relation $p \Vdash P$, where

- P is a formula in some logic (e.g. HA),
- n is an entity with some computational or algorithmic content (e.g. a natural number),
- $\Vdash$ is some relation of 'providing constructive evidence for' (e.g. Kleene's $\Vdash$ which closely parallels BHK).

Each of these three things is asking to be generalized/varied! This gives a host of realizability interpretations for different (typically constructive) systems, leading a wealth of unprovability/consistency results and other proof-theoretic applications.

Generalization 1: Extending the logic

We can extend HA to a 'higher type' version HA^ω, in which variables have types generated by $\sigma ::= \iota \mid \sigma \to \tau$. This lets us formalize not just number theory but lots of analysis too.

What's new here is the definition of when n 'represents' an object of type σ. The key idea is to define a partial equivalence relation (PER) $\sim_\sigma$ on $\mathbb{N}$ for each σ:

- $n \sim_\iota n'$ iff $n = n'$,
- $n \sim_{\sigma\to\tau} n'$ if whenever $m \sim_\sigma m'$, we have $n \bullet m \sim_\tau n' \bullet m'$.

We can then say n 'realizes' a σ object if $n \sim_\sigma n$.

The definition of $\Vdash$ can now proceed as before, e.g.:

- $n \Vdash \exists x^\sigma . P$ iff $n = \langle m, p \rangle$ where $m \sim_\sigma m$ and $p \Vdash P[m/x]$.

Extending the logic (continued)

Our interpretation of HA^ω supports some interesting 'counter-classical' principles (cf. Russian recursive analysis).

E.g. 'all functions from $(\mathbb{N} \to \mathbb{N})$ to itself (or from $\mathbb{R}$ to itself) are continuous'. This is seen to be realizable via the Kreisel-Lacombe-Shoenfield theorem (1959).

The system HA^ω is 'predicative' in spirit and doesn't include e.g. full comprehension principles.

However, it's possible to give Kleene-style realizability interpretations for systems all the way up to Intuitionistic ZF (even with large cardinals). We can thus obtain a version of the existence property, plus consistency with various counter-classical principles, even for these systems (Friedman-Scedrov 1984).

Generalization 2: Other kinds of realizing object

What abstract features of $\mathbb{N}$ does Kleene realizability rely on? All we really needed was the structure of a partial combinatory algebra (PCA): that is, a set A equipped with an 'application' operation $\bullet : A \times A \rightharpoonup A$, in which there are elements k, s satisfying

$$k \bullet x \bullet y = x \qquad s \bullet x \bullet y \downarrow \qquad s \bullet x \bullet y \bullet z \succeq x \bullet z \bullet (y \bullet z)$$

(N.B. *Pairing* is definable in this setting!) Other examples:

- The set of closed terms of pure untyped lambda calculus modulo β-equality. Here e.g. CT_0 isn't realizable.
- Kleene's 'second model' K_2 (Kleene-Vesley 1965). This is a certain PCA with underlying set $\mathbb{N}^{\mathbb{N}}$, in which application is 'continuous' w.r.t. the usual Baire topology.
  This gives a realizability interpretation closer to Brouwerian flavours of intuitionism: e.g. it validates the fan theorem.

Other kinds of realizing object (continued)

PCAs are untyped structures: elements of A serve as both ‘data’ and ‘operations’. But we can also generalize to typed structures.

Let $\mathcal{T}$ be a set of *types*, endowed with binary operations $\times, \rightarrow$. A typed PCA A over $\mathcal{T}$ is a family of sets $(A_\sigma \mid \sigma \in \mathcal{T})$ with application operations $\bullet_{\sigma\tau} : A_{\sigma\rightarrow\tau} \times A_\sigma \rightharpoonup A_\tau$, and containing elements $k_{\sigma\tau}$, $s_{\rho\sigma\tau}$, $\mathit{pair}_{\sigma\tau}, \mathit{fst}_{\sigma\tau}, \mathit{snd}_{\sigma\tau}$ satisfying certain axioms.

Realizers for a formula P will then have a type determined by the structure of P.

Typed PCAs are perhaps the ‘natural’ framework for realizability in the spirit of BHK. If all the $\bullet_{\sigma\tau}$ are *total* operations, we get what is known as modified realizability (Kreisel 1962).

Generalization 3: Other realizability relations

There is a bewildering array of alternative ways of defining a 'realizability' relation $\Vdash$ or something similar. We've already seen one: the concept of realizability-with-truth. Others include:

- Slash relations, **q**-realizability (Kleene, Aczel, Friedman)
- Dialectica interpretation (Gödel 1958)
- Lifschitz realizability (Lifschitz 1979)
- Extensional realizability (van Oosten 1990)

For more on these (and on everything else we've covered so far), see the works of Troelstra and van Oosten. For now, standard realizability over typed PCAs will be plenty to be going on with.

A recent development: Krivine realizability

In the past decade, J.-L. Krivine has shown how to give a realizability interpretation of classical logic — in fact, for all of ZF set theory and beyond!

Idea: Krivine's 'realizers' are terms in a λ-calculus with `callcc`. In typed settings, `callcc` often has type $((A \to B) \to A) \to A$. Read as a proposition, this is just Peirce's law, which is valid classically but not intuitionistically (Griffin 1990).

The operational rules of Krivine's calculus involve not just terms but stacks (lists of terms). Both terms and stacks play a role in the realizability definition. See OL's lecture for more!

Goal: Extend this to all of ZFC. Krivine's philosophy is that new programming concepts should be motivated by their need to realize important axioms.

Extracting programs from proofs

Another 'face' of realizability interpretations:

Suppose the intended behaviour of some program is specified by a logical formula $P(x, y)$, giving the desired relationship between the 'input' x and the 'output' y.

Suppose too we have a proof of $\forall x. \exists y. P(x, y)$. This yields a realizer for this formula, that is, a 'program' mapping any x to a suitable pair $\langle y, p \rangle$. From this, we can extract a program mapping x to a suitable y.

More on program extraction in SB's lecture!

The model-theoretic view (Hyland c. 1980)

An interpretation of a logic ('P is satisfied if . . .') can often be cast as a model: a mathematical structure given independently of the logic in which formulae can be assigned denotations: $P \mapsto [\![P]\!]$.

Example: Interpreting formulae P with one free variable as predicates on a set A.

Interpretation: Define a relation $a \models P$ for $a \in A$.

Model: Define a mapping $P \mapsto [\![P]\!] \in \mathcal{P}(A)$ (a Boolean algebra).

Hyland showed how to treat realizability in terms of categorical models. This isolates a rich structure that can be studied in advance of choosing a logic. It also turns out that this structure provides a natural home for many other things besides logics . . .

Realizability models

Let's work with an arbitrary PCA $(A, \bullet)$.

Hyland defined a realizability topos RT(A), a universe for 'intuitionistic set theory'. RT(K_1) is known as the effective topos.

For now, we'll work with a simpler category PER$(A) \hookrightarrow$ RT(A).

- Objects: PERs (i.e. symmetric transitive relations) on A.
- Morphisms $R \to S$: define a PER S^R by

$$a\ S^R a' \quad \Leftrightarrow \quad (\forall b, b'.\ b\,R\,b \Rightarrow a \bullet b\ S\ a' \bullet b')$$

  A morphism $R \to S$ is an equivalence class for S^R.

Intuition: PERs are 'datatypes' implementable on the 'abstract machine' A. Elements a with aRa are 'machine representations' ('realizers') of data values. Elements a, b with aRb realize the *same* data value. Morphisms are machine-computable functions.

Structure in PER(A)

Any PCA admits a representation of natural numbers: $n \mapsto \bar{n}$. So in PER(A) we have a natural number object N: $\bar{n}N\bar{n}$ for every n and that's all.

PER(A) is cartesian closed (exponentials S^R as on previous slide). The finite types over N are exactly those we saw earlier.

Actually, PER(A) is locally cartesian closed and regular. In any such category, one can interpret first order logic over whatever types are around, using standard ideas from categorical logic. In the case of PER(A), this agrees precisely with the standard realizability interpretation (e.g. for HA$^\omega$).

A predicate P on type σ is modelled as a subobject $[\![P]\!]$ of $[\![\sigma]\!]$.

(For full higher order logic, we need the whole of RT(A).)

PER(A) **as a model for type systems**

In fact, Girard (1972) had already used PERs to model his polymorphic typed lambda calculus ('System F'):

$$\sigma \ ::= \ X \mid \sigma \to \tau \mid \forall X.\sigma$$

The impredicative polymorphism here can't be modelled using classical sets (Reynolds 1984). But in PER(A), it can:

$$\begin{aligned} [\![X]\!]_\nu &= \nu(X) \\ [\![\sigma \to \tau]\!]_\nu &= [\![\tau]\!]_\nu{}^{[\![\sigma]\!]_\nu} \\ [\![\forall X.\sigma]\!]_\nu &= \bigcap_R [\![\sigma]\!]_{\nu(X \mapsto R)} \end{aligned}$$

There's also a nice interpretation of subtyping, as in System $\mathsf{F}_{<:}$.

$$\sigma <: \tau \ \Rightarrow \ [\![\sigma]\!] \subseteq [\![\tau]\!]$$

In fact, PER(A) can model quite complex type systems that can't (at present) be modelled semantically in any other way. (Conceptually nice; technical usefulness somewhat unclear.)

Languages with recursion

Languages like System F can only express total functions (and our PER semantics reflects this). However, most programming languages allow partial functions to be defined using iteration and/or general recursion.

Consider the simple types over ι, interpreted in PER(K_1) by setting $[\![\,\iota\,]\!] = N_\perp$, $[\![\,\sigma \to \tau\,]\!] = [\![\,\tau\,]\!]^{[\![\,\sigma\,]\!]}$, where

$$m\, N_\perp\, n \;\Leftrightarrow\; m \bullet 0 \simeq n \bullet 0$$

It turns out that every $[\![\,\sigma\,]\!]$ admits a fixed point operator: a morphism $Y_\sigma : [\![\,\sigma\,]\!]^{[\![\,\sigma\,]\!]} \to [\![\,\sigma\,]\!]$. (Cf. Myhill-Shepherdson theorem.) This means we can interpret Plotkin's language PCF (simply typed λ-calculus with arithmetic and general recursion).

Synthetic domain theory

In fact, for many A, there are quite rich subcategories of PER(A) — hence of RT(A) — which enjoy these 'fixed points for free'. (Yet another counter-classical feature of realizability universes!)

Objects of these subcategories may be viewed as carrying an intrinsic domain structure. This contrasts with extrinsic (e.g. CPO) structure as in classical domain theory.

These 'categories of domains' are able to model extensions of PCF with strong polymorphism, recursive types, subtyping, . . .

As with much other work in denotational semantics, a long-term hope is that these models should assist with the design and validation of useful program logics for such languages. In practice, though, PERs are often hard to get a good mathematical handle on.

Differences between realizability models

Consider two PCAs:

- Kleene's first model $K_1 = (\mathbb{N}, \bullet)$.
- Closed untyped λ-terms modulo β-equality: Λ^0/β. (Here natural numbers can be realized by Church numerals, and $\perp$ by unsolvable terms.)

Both of these give PER models for PCF. However, PER(K_1) also contains *parallel-or* and *exists* operations, while PER(Λ^0/β) doesn't (Berry sequentiality theorem).

Conjecture (Longley/Phoa): Every element of simple type in PER(Λ^0/β) is PCF-definable. (Hence the simple type structure in PER(Λ^0/β) coincides with PCF/ $\approx_{obs}$.)

'Notions of computability'

This suggests that there is some difference in the 'computational power' of K_1 and Λ^0/β, even though they're both Turing complete.

Intuitively, a K_1 realizer is like a program whose source code can be inspected. A Λ^0/β realizer is more like a 'black box'.

In any case, at the level of functionals of simple type, we have at least two notions of computability: 'parallel' and 'sequential'.

All this suggests a general programme of mapping out interesting computability notions and the relationships between them. In the remaining lectures, we'll see how realizability provides a useful tool for doing this. (E.g. can K_1 be 'realized by' Λ^0/β and vice versa?)