Topology for functional programming

Martín Escardó
University of Birmingham, England

EWSCS, Palmse, Estonia, 26 Feb – 2 Mar 2012

Computational fact

Function types $A \to B$ with A infinite don't have decidable equality.

$$f = g \iff \forall x \in A(f(x) = g(x)).$$

How could one possibly check infinitely many cases in finite time?

Also, equality on B may not be decidable.

Computational fact

Function types $A \to B$ with A infinite don't have decidable equality.

How could one possibly check infinitely many cases in finite time?

This is certainly *not* possible if $A = B = \mathbb{N}$.

Turing, Kleene, . . .

Fact?

There *are* function types with decidable equality!

1. $\mathbb{N} \to \mathbb{N}$ ✗
2. $\mathbb{N} \to \mathrm{Bool}$ ✗
3. $(\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}$ ✓
4. $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ ✗
5. $(\mathbb{N} \to \mathrm{Bool}) \to (\mathbb{N} \to \mathrm{Bool})$ ✗
6. $((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}) \to \mathrm{Bool}$ ✗
7. $(((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}) \to \mathrm{Bool}) \to ((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N})$ ✓

Pause to see and run such a decision algorithm

`http://www.cs.bham.ac.uk/~mhe/.talks/EWSCS2012/`

The programs won't be understandable at this point:

1. They exploit topological ideas to be introduced in the tutorial.

2. I have chosen an algorithm that exponentially reduces the time complexity of simpler algorithms.

In the theory developed in these slides,
I will sacrifice efficiency and show you simpler algorithms.

Topological explanation

In the decidable cases:

i. The domains of the function types are compact.

ii. The codomains are discrete.

Topological explanation

In the decidable cases:

i. The domains of the function types are compact.

ii. The codomains are discrete.

1. $\mathbb{N} \to \mathbb{N}$ ×
2. $\mathbb{N} \to \mathrm{Bool}$ ×
3. $(\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}$ ✓
4. $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ ×
5. $(\mathbb{N} \to \mathrm{Bool}) \to (\mathbb{N} \to \mathrm{Bool})$ ×
6. $((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}) \to \mathrm{Bool}$ ×
7. $(((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}) \to \mathrm{Bool}) \to ((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N})$ ✓

Inductive definition of types with decidable equality

Simultaneously defined with exhaustively searchable types:

$$\begin{aligned} \texttt{discrete} &\approx \text{has decidable equality} \\ \texttt{compact} &\approx \text{exhaustively searchable.} \end{aligned}$$

$$\begin{aligned} \texttt{discrete} &::= 2 \mid \mathbb{N} \mid \texttt{discrete} \times \texttt{discrete} \mid \texttt{compact} \to \texttt{discrete}, \\ \texttt{compact} &::= 2 \mid \texttt{compact} \times \texttt{compact} \mid \texttt{discrete} \to \texttt{compact}. \end{aligned}$$

Here $2 = \{0, 1\}$ is (isomorphic to) the type of booleans.

The types that are simultaneously discrete and compact are finite.

What is topology?

Topology is the field of mathematics that studies spaces and
continuous transformations between them.

Topology comes with its own language:
Topological space, continuous function, open set, closed set, limit, convergence, discrete space, Hausdorff space, compact space, ...

Topology in constructive mathematics (1920-1950)

Brouwer.

Continuity in constructive mathematics

Think of functions $\alpha\colon \mathbb{N} \to \mathbb{N}$ as sequences $\alpha_0, \alpha_1, \alpha_2, \ldots$ of natural numbers.

Consider a function $f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ in constructive mathematics.

Brouwer reasoned that it must be the case that

> Finite parts of the output sequence $f(\alpha)$ can depend only on finite parts of the input sequence α.

This amounts to saying that f is continuous.

Continuity more rigorously

We need a preliminary definition:

For sequences $\alpha, \beta \colon \mathbb{N} \to \mathbb{N}$ and a number $n \in \mathbb{N}$, write

$$\alpha =_n \beta \iff \forall i < n(\alpha_i = \beta_i).$$

The two sequences agree in the first n positions.

Continuity more rigorously

A function $f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ is continuous iff

$$\forall \alpha \, \forall n \, \exists m \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) =_n f(\beta)).$$

If you want to know the first n positions of the output sequence $f(\alpha)$, it is enough to know the first m positions of the input sequence α.

Notice that m depends on both α and n.

Non-continuous functions

A function $f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ is continuous iff

$$\forall \alpha \, \forall n \, \exists m \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) =_n f(\beta)).$$

Counter-example:

$$\boldsymbol{f}(\alpha) = \begin{cases} 0^\omega & \text{if } \alpha = 0^\omega, \\ 1^\omega & \text{otherwise.} \end{cases}$$

Notice that this uses excluded middle.

Without excluded middle you can't define a non-continuous function!

For those of you who already know a bit of topology

1. Endow $\mathbb{N}$ with the discrete topology.

2. Endow $\mathbb{N}^{\mathbb{N}} = (\mathbb{N} \to \mathbb{N})$ with the product topology.

   (Which agrees with the compact-open topology, which gives the categorical exponential.)

   This space is known as the Baire space.

3. Then $f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ is continuous in the above sense if and only if it is topologically continuous.

   (Inverse images of open sets are open.)

The cantor space

Let $2 = \{0, 1\}$ be the set of binary numbers.

The space $2^{\mathbb{N}} = (\mathbb{N} \to 2)$ of binary sequences is known as the Cantor space.

(Because it is homeomorphic to Cantor's third-middle set.)

We can repeat the above story: A function $f : (\mathbb{N} \to 2) \to (\mathbb{N} \to \mathbb{N})$ is continuous iff

$$\forall \alpha \, \forall n \, \exists m \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) =_n f(\beta)).$$

But something new happens: given n, we can find m independently of α.

Uniform continuity

A function $f : (\mathbb{N} \to 2) \to (\mathbb{N} \to \mathbb{N})$ is continuous iff

$$\forall \alpha \, \forall n \, \exists m \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) =_n f(\beta)).$$

A function $f : (\mathbb{N} \to 2) \to (\mathbb{N} \to \mathbb{N})$ is uniformly continuous iff

$$\forall n \, \exists m \, \forall \alpha \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) =_n f(\beta)).$$

Clearly, uniform continuity implies plain continuity.

Uniform continuity

Theorem. Any continuous $f : (\mathbb{N} \to 2) \to (\mathbb{N} \to \mathbb{N})$ is uniformly continuous.

Proof ingredient in classical topology:

Compactness of the Cantor space. Alternatively, König's Lemma.

Proof ingredient in Brouwer's intuitionistic mathematics:

The Fan Theorem, which is proved from the Bar induction axiom.

Another instance of (uniform) continuity

A function $f : (\mathbb{N} \to 2) \to \mathbb{N}$ is continuous iff

$$\forall \alpha \, \exists m \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) = f(\beta)).$$

A function $f : (\mathbb{N} \to 2) \to \mathbb{N}$ is uniformly continuous iff

$$\exists m \, \forall \alpha \, \forall \beta \, (\alpha =_m \beta \implies f(\alpha) = f(\beta)).$$

The two notions are again equivalent.

The smallest such number $m = m(f)$ is called the modulus of uniform continuity of f.

Topology in recursion theory (1950-1980)

Kleene, Kreisel, Lacombe, Myhill, Shepherdson, Rice, Nerode, Normann, Gandy, Hyland, . . .

1. Neighbourhood spaces.
2. Limit spaces.
3. Compactly generated topological spaces.
4. Sequential topological spaces.
5. Stone spaces.

Topology in computer science (1960–present)

Scott.

1. Domain theory: denotational semantics & computability.

2. The main categories of domains are full subcategories of the category of topological spaces.

Domain theory is good for *partial* computation.

In this tutorial we'll tacitly concentrate on *total* functions.

This will allow us to work with more traditional topological spaces.

(But domain theory remains crucial.)

Topology in computer science (1980–present)

Smyth. Predicate transformers: a topological view.

Abramsky. Logic of observable properties.

Vickers. Topology via logic.

Weihrauch. Type 2 computability theory.

Smyth was the first in the computer science community to take the topological view of domains seriously and to advocate the need for more general spaces.

(See his *Topology* chapter in the Handbook of Logic in Computer Science.)

Topology in computer science (2000–present)

Simpson & Schröder:

1. QCB spaces.

   Again a full subcategory of topological spaces, including domains.

2. A richer supply of spaces for semantics and computability.

3. Unifies Scott's and Weihrauch's approaches.

4. Allows explicit treatment of total objects (without bottom elements).

5. Arises from a convergence of ideas from semantics and computability.

Topology in operational semantics (2005)

Escardó & Ho.

1. You don't need a denotational semantics to find and exploit domain theory and topology in programming languages.

2. It can be identified directly in operational terms.

3. Doesn't suffer from the so-called *full abstraction problem*.

Operational domain theory and topology of sequential programming languages. Information and Computation 207 (2009).

Dictionary relating topology and computation

space	$\rightsquigarrow$	type
continuous function	$\rightsquigarrow$	computable function
clopen set	$\rightsquigarrow$	decidable set
open set	$\rightsquigarrow$	semi-decidable set
closed set	$\rightsquigarrow$	set with semi-decidable complement
discrete space	$\rightsquigarrow$	type with decidable equality
Hausdorff space	$\rightsquigarrow$	type with semi-decidable $\neq$
Compact set	$\rightsquigarrow$	exhaustively searchable set, in a finite number of steps

Another connection of topology with computation/constructive mathematics has recently been found in type theory, via homotopy types.

These two connections should be combined!

What is this dictionary good for?

Algorithm for finding theorems in computability theory

1. Go the library.
2. Pick a book on topology.
3. Pick a theorem.
4. Apply the dictionary.
5. Get a theorem in computation.

Synthetic topology of data types and classical spaces, 134 pages, ENTCS, 2004.

Exhaustible sets in higher-type computation.
Logical Methods in Computer Science, 2008.

Some examples

1. A compact subspace of a Hausdorff space is closed.
2. A closed subspace of a compact space is compact.
3. A product of arbitrarily many compact spaces is compact.
4. If X is compact and Y is discrete then $X \to Y$ is discrete.
5. If Y is Hausdorff then $X \to Y$ is Hausdorff.
6. If $K \subseteq X$ is compact and $U \subseteq Y$ is open, then $\{f \in (X \to Y) \mid f(K) \subseteq U\}$ is open.
7. A space X is compact iff for every space Y the projection $X \times Y \to Y$ maps closed sets to closed sets.
8. Any non-empty, countably based, compact Hausdorff space is a continuous image of the Cantor space $(\mathbb{N} \to 2)$.

⋮

Simple example

A compact subspace of a Hausdorff space is closed.

Assumptions: X is a Hausdorff space, $K \subseteq X$ is compact.

Conclusion: K is closed.

$\rightsquigarrow$

Assumptions: X has semi-decidable $\neq$, the set $K \subseteq X$ is searchable.

Conclusion: K has semi-decidable complement.

Algorithm: For given $x \in X$, to semi-decide $x \notin K$,
exhaustively check that $x \neq y$ for every $y \in K$.

What is an exhaustively searchable set?

There are two main versions of the notion:

1. Given a semi-decidable p, semi-decide whether $p(x)$ holds for all x in the set.

   Synthetic topology of data types and classical spaces, ENTCS, 2004.

2. Given a decidable p, decide whether $p(x)$ holds for all x in the set.

   Exhaustible sets in higher-type computation. LMCS, 2008.

In this tutorial I will consider the decidable case.

Exhaustively searchable sets

And there are two natural versions in the decidable case.

Given a type X, a set $K \subseteq X$ and a decidable predicate $p\colon X \to \mathrm{Bool}$, we may wish to:

1. Decide whether $p(k)$ holds for some $k \in K$ or not.

2. Find an example of a $k \in K$ satisfying p, or a counter-example otherwise.

Exhaustively searchable sets

1. Decide whether $p(k)$ holds for some $k \in K$.

$$\boxed{E_K : (X \to \mathrm{Bool}) \to \mathrm{Bool}}$$

Characteristic function of the existential quantifier.

2. Find an example of a $k \in K$ satisfying p, or a counter-example otherwise.

$$\boxed{\varepsilon_K : (X \to \mathrm{Bool}) \to X}$$

Selection function. (Cf. Hilbert's ε-calculus.)

Exhaustively searchable sets

A subset $K \subseteq X$ of a type X is:

1. Exhaustible if there is a computable functional

$$E_K \colon (X \to \mathrm{Bool}) \to \mathrm{Bool}$$

such that $E_K(p) = \mathrm{True} \iff \exists k \in K(p(k) = \mathrm{True})$.

2. Searchable if there is a computable functional

$$\varepsilon_K \colon (X \to \mathrm{Bool}) \to X$$

such that

i. $\varepsilon_K(p) \in K$, and
ii. $p(\varepsilon(p)) = \mathrm{True} \iff \exists k \in K(p(k) = \mathrm{True})$.

Exhaustively searchable sets

1. $K \subseteq X$ is Exhaustible if there is a computable functional

$$E_K \colon (X \to \mathrm{Bool}) \to \mathrm{Bool}.$$

2. $K \subseteq X$ is Searchable if there is a computable functional

$$\varepsilon_K \colon (X \to \mathrm{Bool}) \to X.$$

We can define $E_K(p) = p(\varepsilon_K(p))$, which shows that
if K is searchable then it is exhaustible.

Theorem

Consider the least set of types containing the booleans, natural numbers and closed under products and function types.

1. Any non-empty exhaustible set is searchable.
2. A computable image of an exhaustible set is exhaustible.
3. A decidable subset of an exhaustible set is exhaustible.
4. Searchable sets are closed under finite and countable products.
5. Hence the Cantor space $\prod_{n\in\mathbb{N}} 2 = (\mathbb{N} \to 2)$ is searchable.
6. Any searchable set is a computable image of the Cantor space.

And more.

The proofs

1. They are written in mathematical vernacular in the papers, with explicitly given algorithms.

2. I've also written most of the algorithms in Haskell.

3. I've also written some of the proofs/algorithms in Agda.

I'll sketch one proof to illustrate how topology is used.

Cantor space search by Ulrich Berger (1990)

```
data Bit = Zero | One
type Cantor = [Bit]  -- I only care about the infinite lists

find :: (Cantor -> Bool) -> Cantor
forsome, forevery :: (Cantor -> Bool) -> Bool

find p = if forsome(\a -> p(Zero : a))
         then Zero  : find(\a -> p(Zero : a))
         else One   : find(\a -> p(One  : a))

forsome p = p(find p)
forevery p = not(forsome(\a -> not(p a)))
```

Corollary

Decidability of extensional equality:

```
equal :: (Cantor -> Integer) -> (Cantor -> Integer) -> Bool
equal f g = forevery(\a -> f a == g a)
```

This is exponentially slow in a precise sense.

So I looked for faster algorithms.

Infinite sets that admit fast exhaustive search. LICS'2007.

Exhaustible sets in higher-type computation. LMCS, 2008.

Proof sketch of Berger's algorithm

Let's rewrite the code by inlining one mutual recursive call:

```
find :: (Cantor -> Bool) -> Cantor

find p = if p(Zero : find(\a -> p(Zero : a))
         then Zero : find(\a -> p(Zero : a))
         else One  : find(\a -> p(One  : a))
```

Notice something odd: there is a recursive call in the condition!

This, together with the uniform continuity of `p :: Cantor -> Bool`, is what makes the algorithm work.

Uniform continuity revisited

A function $p : (\mathbb{N} \to 2) \to 2$ is uniformly continuous iff

$$\exists m \, \forall \alpha \, \forall \beta \, (\alpha =_m \beta \implies p(\alpha) = p(\beta)).$$

The number $m = m(p)$ is called the modulus of uniform continuity of p.

Notice that when $m(p) = 0$ the condition $\alpha =_m \beta$ always holds.

Hence in this case we get $\forall \alpha \, \forall \beta \, (p(\alpha) = p(\beta))$. That is, that p is constant.

Proof sketch of Berger's algorithm

```
find :: (Cantor -> Bool) -> Cantor
find p = if p(Zero : find(\a -> p(Zero : a))
         then Zero : find(\a -> p(Zero : a))
         else One  : find(\a -> p(One  : a))
```

The proof that this algorithm is productive proceeds by induction on the modulus of continuity of p.

1. If $m(p) = 0$ then p doesn't depend on its argument.

   In this case the recursive call in the condition doesn't take place, and one bit of output is produced.

2. If $m(p) > 0$ then the two predicates `\a -> p(Zero : a)` and `\a -> p(One  : a)` have modulus of continuity $m(p) - 1$ or smaller.

Technical subtlety

I glossed over some important details in the above sketch for simplicity.

If I get to talk about domain theory, I'll come back to this:

There are two relevant notions of modulus of uniform continuity.

Difference between being constant and "not looking at its argument".

The second condition is captured by a certain *intensional* modulus of continuity, which is what really has to be used in the proof.

Have a look at e.g. Operational domain theory and topology of sequential programming languages. Information and Computation 207 (2009).

Some combinators for searchable sets

```
type J r x = (x -> r) -> x        -- Selection functions.
type K r x = (x -> r) -> r        -- Quantifiers.
```

Here they are used with `r = Bool`

But in related work with Paulo Oliva we use them with r more general.

Type of searchable/exhaustible subsets of the type X:

```
type Searchable  x = J Bool x
type Exhaustible x = K Bool x

image :: (x -> y) -> Searchable x -> Searchable y
times :: Searchable x -> Searchable y -> Searchable(x,y)
prod  :: (N -> Searchable x) -> Searchable(N -> x)
```

Some combinators for searchable sets

```
forsome, forevery :: Searchable x -> Exhaustible x
forsome e p = p(e p)
forevery s p = not(forsome s (\x -> not(p x)))

equal :: Eq y => Searchable x -> (x -> y) -> (x -> y) -> Bool
equal s f g = forevery s (\a -> f a == g a)
```

Agda program/proofs

http://www.cs.bham.ac.uk/~mhe/agda/

Uses?

1. Alex Simpson's *Lazy functional algorithms for exact real functionals*.

   Compute maximum value and integral of a real function, with real numbers represented as infinite sequences of digits.

2. I've also automatically verified some real-number programs by checking infinitely many cases.

Verification of conventional programs? I'll show you one example in Haskell.

Continuous model of dependent types

Johnstone's topological topos.

1. Fully embeds the category of sequential topological spaces.

2. Fully embeds the category of Kuratowski's limit spaces.

3. Validates the above continuity principles (and more).

Remark in passing

The classical set-theoretical model of Gödel's system T is not fully abstract.

But the topological model of sequential spaces is.

Similarly for Martin-Löf type theory.

See my note *Kreisel's counter-example to full abstraction of the set-theoretical model of Goedel's system T.*

Work with Paulo Oliva

The countable product of selection functions, originally developed to realize the Tychonoff theorem from topology, also:

1. Optimally plays sequential games (such as Tic-Tac-Toe, to be mundane).

2. Realizes the double-negation shift from proof theory.

   In turn used to realize the classical axiom of countable (dependent) choice.

`http://www.cs.bham.ac.uk/~mhe/papers/msfp2010/`

`http://www.cs.bham.ac.uk/~mhe/pigeon/`

The only difference between reality and fiction is that fiction needs to be credible. Mark Twain