Algorithmic topology of program types

Martín Escardó

School of Computer Science, Birmingham University, UK

Wollic'07, Rio de Janeiro, 2-5 July 2007

Input of topology in computation

Very old and well established, going back to Brouwer (1920's).

New: Compact space $\sim$ exhaustively searchable set.

Outrageous: There are infinite examples.

Surprising: Faster than we would expect.

Old topological slogan

A set is compact if it behaves as if it were finite.

So perhaps we can perform exhaustive search on compact sets?

But a topological example is the Cantor space $(\mathbb{N} \to \mathrm{Bool})$.

So this doesn't seem very likely.

But done by Gandy (1970's) and Berger (1990) independently.

Outrageous consequences

In (Kleene–Kreisel) higher-type computation, e.g. the two types

$$(\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}$$

$$(((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}) \to \mathrm{Bool}) \to ((\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N})$$

have decidable equality.

But, of course, the type $\mathbb{N} \to \mathbb{N}$ doesn't.

How fast in practice?

I have recently come up with an algorithm that is fast for surprising examples:

```
type Cantor = N -> Bool

forall :: (Cantor -> Bool) -> Bool
forall p = ...

equal :: (Cantor -> N) -> (Cantor -> N) -> Bool
equal f g = forall(\a -> f a == g a)
```

Experiments

```
f,g,h :: Cantor -> N

f a = a(10*a'(3^400)+100*a'(4^400)+1000*a'(5^400))

g a = a(10*a'(3^400)+100*a'(4^400)+1000*a'(6^400))

h a = a(10 + if a(4^400) then j+100 else j)
    where i = if a(5^400) then 1000 else 0
          j = if a(3^400)) then i else i-10
```

where we abbreviate `a'(i) = if a(i) then 1 else 0`.

Experiments

Using interpreted Haskell (ghci) in a 1.7GHz machine:

```
> equal f g
False
(3.18 secs, 379345012 bytes)

> equal f h
True
(1.50 secs, 178959112 bytes)
```

Efficiency and complexity

Preliminary theorems in talks elsewhere.

Slides available on request.

In this talk I want to focus on computability.

Input of logic and computation in topology

New: λ-calculus transparently proves theorems in topology.

Algorithmic *understanding* of topology.

Algorithmic *content* of topology.

Can run topological proofs.

Plan of the talk

Input of topology and logic in computation.

Input of logic and computation in topology.

(P. Taylor has done some interesting work on the input of topology and computation in logic, not reported here.)

A missing concept in computability theory

A subset K of a computational space X is semi-exhaustible if

for every semi-decidable property p of elements of X,

one can semi-decide, uniformly in p,

whether $p(k)$ holds for all $k \in K$.

Myhill–Shepherdson theorem

Every computable functional is continuous.

Every effectively continuous functional is computable.

Computable $\sim$ continuous.

Rice–Shapiro theorem

Every semi-decidable set is open.

Every effectively open set is semi-decidable.

Semi-decidable $\sim$ open.

Analogue (new)

Every semi-exhaustible set is compact.

Every effectively compact set is semi-exhaustible.

Semi-exhaustible $\sim$ compact.

Folklore

A set with semi-decidable equality is discrete.

An effectively discrete set has semi-decidable equality.

Semi-decidable equality $\sim$ discrete.

Folklore

A set with semi-decidable apartness is Hausdorff.

An effectively Hausdorff set has semi-decidable apartness.

Semi-decidable apartness $\sim$ Hausdorff.

Transfer principles for notions

Continuous map. Computable map.

Open/closed set. Semi-decidable/co-semi-decidable set.

Discrete/Hausdorff space. Semi-decidable equality/apartness.

Compact set. Semi-exhaustible set.

Limit. Computation of an infinite object.

Analysis, topology. Computability theory.

Transfer principles for theorems

New: Theorems tend to have a natural algorithmic reading.

And their proofs too.

Simple example

If X is Hausdorff and $K \subseteq X$ is compact,
then K is closed.

Transfer principles give:

If X has semi-decidable apartness and $K \subseteq X$ is semi-exhaustible,
then K is co-semi-decidable.

Proof. $x \notin K$ iff for all $k \in K$, $x \neq k$.

Anatomy of the proof

Let $\mathbb{S} = \{\bot, \top\}$ be the set of semi-truth-values.

$\bot$ = unobservable false (non-terminating computation).

$\top$ = observable true (terminating computation).

Topologically: $\top$ isolated, $\bot$ limit point. Sierpinski space.

Anatomy of the proof

X Hausdorff $\iff$ $(\neq)\colon X \times X \to \mathbb{S}$ continuous.

$K \subseteq X$ compact $\iff$ $\forall_K \colon \mathbb{S}^X \to \mathbb{S}$ continuous.

$F \subseteq X$ closed $\iff$ $\chi_{X \setminus F} \colon X \to \mathbb{S}$ continuous.

Anatomy of the proof

X effectively Hausdorff $\iff$ $(\neq)\colon X \times X \to \mathbb{S}$ computable.

$K \subseteq X$ semi-exhaustible $\iff$ $\forall_K\colon \mathbb{S}^X \to \mathbb{S}$ computable.

$F \subseteq X$ co-semi-decidable $\iff$ $\chi_{X\setminus F}\colon X \to \mathbb{S}$ computable.

Anatomy of the proof

Any compact subspace K of a Hausdorff space X is closed.

That is, given continuous maps (or higher-type algorithms)

$$\forall_K \colon \mathbb{S}^X \to \mathbb{S},$$

$$(\neq) \colon X \times X \to \mathbb{S},$$

we can find

$$\chi_{X \setminus F} \colon X \to \mathbb{S}.$$

Proof/construction. $\chi_{X \setminus F}(x) = \forall_K(\lambda k.x \neq k)$.

Anatomy of the proof

λ-definability preserves computability.

λ-definability preserves continuity.

$\Longrightarrow$ Previous construction proves the topological and algorithmic readings of the theorem *simultaneously*.

Synthetic topology

Continuity is taken as primitive ("unanalised").

All basic notions are reduced to that of continuity.

Open = continuous membership.

Closed = continuous co-membership.

Discrete = continuous equality.

Hausdorff = continuous apartness.

Compact = continuous universal quantification.

Synthetic topology

Because continuity is regarded as primitive, it can be taken to mean computability (or even definability in a language).

$\Longrightarrow$

Both classical and algorithmic topology are particular cases of synthetic topology.

Synthetic proof

Combine topological notions using λ-calculus.

Synthetic proofs are their own realizers

A synthetic proof is a logical expression.

Written in λ-calculus.

We can run the logical expression.

E.g., in the previous theorem,

To semi-decide $x \notin K$, run the program $\forall k \in K.x \neq k$.

Another example

If X is compact and Y is discrete, then Y^X is discrete.

Synthetic proof.

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

This is again a proof and a program.

(Which we have in fact run at the beginning of the talk.)

Dual example

(If X is overt and) Y is Hausdorff, then Y^X is Hausdorff.

Synthetic proof.

$f \neq g \iff \exists x \in X. f(x) \neq g(x).$

$\exists \colon \mathbb{S}^X \to \mathbb{S}$ is always continuous.

If it is computable, we say that X is overt.

(Joyal, Johnstone, Sambin, Taylor . . .)

Applying this to $X = (\mathbb{N} \to \mathrm{Bool})$ and $Y = \mathbb{N}$

The higher type $(\mathbb{N} \to \mathrm{Bool}) \to \mathbb{N}$ has decidable equality.

I.e. it is both discrete and Hausdorff.

Recall that I have told you that $(\mathbb{N} \to \mathrm{Bool})$ is compact.

Now I assert that it is also overt, again without proof.

Another missing notion in computation

A subset K of a computational space X is called exhaustible if

for every decidable property p of elements of X,

one can decide, uniformly in p,

whether or not $p(k)$ holds for all $k \in K$.

Computatable functional $\forall_K \colon \mathrm{Bool}^X \to \mathrm{Bool}$.

Yet another missing notion

A subset K of a computational space X is called searchable if

for every decidable property p of elements of X,

one can find, uniformly in p, some $k \in K$ such that

if p holds for a point in K, then k is an example.

Computable functional $\varepsilon_K \colon \mathrm{Bool}^X \to X$ with image K.

Characterization

TFAE for a non-empty subset K of a higher type over $\mathbb{N}$:

1. K is exhaustible.
2. K is searchable.
3. K is a computable image of the Cantor space.

Non-trivial.

But bottom-up implications are easy and useful in practice.

Exhaustible implies searchable

$2^{2^X} \to X^{2^X}$ $\qquad \forall_K \mapsto \varepsilon_K.$ $\qquad$ (Here $2 = \mathrm{Bool}$.)

Construction and proof of algorithm use Kleene–Kreisel density (higher-type recursion theory) and Ascoli–Arzela (topology).

Rather brutal approach, useless in practice.

Searchable implies image of Cantor space

$X^{2^X} \to X^{2^N}$ $\qquad \varepsilon_K \mapsto f$ such that $f\left(2^N\right) = K$.

Again non-trivial and useless in practice.

The point was to know that the three notions are the same.

Technology to build exhaustible sets

Exhaustible sets are closed under

1. intersections with decidable sets,
2. computable images,
3. finite products,
4. countable products.

Last one non-trivial.

This time all useful in practice.

Intersections with decidable sets

$$X^{2^X} \times 2^X \to X^{2^X} \qquad (\varepsilon_K, \chi_D) \mapsto \varepsilon_{K \cap D}.$$

Computable images

$$Y^X \times X^{2^X} \to Y^{2^Y} \qquad (f, \varepsilon_K) \mapsto \varepsilon_{f(K)}.$$

Finite products

$$X^{2^X} \times Y^{2^Y} \to (X \times Y)^{2^{X \times Y}} \qquad (\varepsilon_K, \varepsilon_L) \mapsto \varepsilon_{K \times L}.$$

Countable products

1. $\prod_i X_i^{2^{X_i}} \to \left(\prod_i X_i\right)^{2^{\prod_i X_i}}$ needs dependent types.

2. $\left(X^{2^X}\right)^{\mathbb{N}} \to \left(X^{\mathbb{N}}\right)^{2^{X^{\mathbb{N}}}}$ $\quad (\varepsilon_{K_i})_i \mapsto \varepsilon_{\prod_i K_i}$.

Countable products

$$\left(X^{2^X}\right)^{\mathbb{N}} \to \left(X^{\mathbb{N}}\right)^{2^{X^{\mathbb{N}}}} \qquad (\varepsilon_{K_i})_i \mapsto \varepsilon_{\prod_i K_i}.$$

Countable Tychonoff theorem.

Above experiment uses this.

Summary

1. Topological notions $\sim$ algorithmic concepts.

   $\Longrightarrow$ Discovery of new algorithmic concepts.

2. Topological theorems $\sim$ algorithms.

   $\Longrightarrow$ Discovery of new algorithms.

   Proofs are their own realizers.

3. Exhaustive search unfeasible, but fast in surprising examples.

Contributions

1. Algorithmic reading of compactness.
2. Algorithmic reading of topological theorems.
3. Algorithmic proofs of topological theorems.
4. Hope of efficient search over infinite spaces.
5. Not developed here: more scope for topology in computation (e.g. sequential computation).

References

M.H. Escardó. Synthetic topology of data types and classical spaces. ENTCS, Elsevier, volume 87, pages 21-156, Nov 2004.

M.H. Escardó and W.K. Ho. Operational domain theory and topology of a sequential programming language.
LICS 2005, pages 427-436.

M.H. Escardó. Infinite spaces that admit fast exhaustive search.
LICS 2007.