Mathematics of computation with infinite objects

Martín Escardó
School of Computer Science, University of Birmingham

Invariants, Oxford, 25th Oct 2011

Let me start with six questions for you

Here is a hint:

2 questions have positive answer.

2 questions have negative answer.

1 question has unknown answer.

1 question doesn't fall in the above three categories.

Question 1

Suppose you have written computer programs for calculating functions

$$f, g \colon \mathbb{Z} \to \mathbb{Z}.$$

Can the computer decide, in finite time, whether or not $f = g$?

Here $\mathbb{Z}$ is the set of integers.

Input: Two functions $f, g \colon \mathbb{Z} \to \mathbb{Z}$.

Output: An element of the set $2 = \{0, 1\}$.

Question 2

Consider the set S of infinite binary sequences $x_0, x_1, x_2, \ldots, x_n, \ldots$ with $x_i \in 2$.

(For example we can use them for representing real numbers in binary.)

Does it make sense to write programs for computing e.g. functions

$f\colon \quad S \to \mathbb{Z}, \qquad$ (given a sequence, calculate an integer);

$g\colon \quad S \to S, \qquad$ (given a real number, calculate another real number).

Question 3

Suppose you have written computer programs for calculating functions

$$f, g \colon S \to \mathbb{Z}.$$

Can the computer decide, in finite time, whether or not $f = g$?

Input: Two functions $f, g \colon S \to \mathbb{Z}$.

Output: An element of the set $2 = \{0, 1\}$.

Question 4

Suppose you have written computer programs for calculating functions

$$f, g \colon S \to S$$

Can the computer decide, in finite time, whether or not $f = g$?

Input: Two functions $f, g \colon S \to S$.

Output: An element of the set $2 = \{0, 1\}$.

Question 5

Can we plot the Mandelbrot set?

Given a point that is not in the boundary, can the computer tell whether it is inside or outside the set?

Question 6

It is easy to write a program that systematically lists all provable statements.

(Let's say formally written in the language of set theory.)

We can also write it so that it stops if it ever lists a statement A and also its negation $\neg A$.

Will this program stop?

Answers

2 questions have positive answer.

2 questions have negative answer.

1 question has unknown answer.

1 question doesn't fall in the above three categories.

(But what else can we say apart from yes, no or don't know???)

Let's leave the questions unanswered for the moment.

A simple computation

Fix a constant $a \in [0, 4]$.

Given $x \in [0, 1]$, it is easy to see that $ax(1 - x) \in [0, 1]$.

Rudimentary model of population growth (Pierre Verhulst (1840s)).

$x = 0$: zero fish in a lake.
$x = 1$: the maximum capacity of fish in the lake.
x_n fish in the nth generation $\implies$ $ax_n(1 - x_n)$ fish in the next generation.

A simple computation

Given $x_0 \in [0, 1]$, define a sequence by the recurrence

$$x_{n+1} = a x_n (1 - x_n).$$

We want to compute x_{60}.

How many fish will the lake have in the 60th generation?

Let's calculate with a computer

Choose $a = 4$ and $x_0 = 43/64 = 0.671875$.

Many other choices will have a similar effect.

Using 7 decimal digits (simple precision) and 16 (double precision) for the calculations, the computer claims that x_{60} is

```
0.934518        0.928604
```

Rounding to two decimal digits, we get $x_{60} \approx 0.93$.

Is that correct?

Let's show intermediate generation counts

This is what the computer claims:

```
 n       simple        double
-----------------------------
 0     0.671875       0.671875
 5     0.384327       0.384327
10     0.313034       0.313037
15     0.022702       0.022736
20     0.983813       0.982892
25     0.652837       0.757549 <------ start to differ
30     0.934927       0.481445 <------ completely different
35     0.848153       0.313159
40     0.057696       0.024009
45     0.991613       0.930892
50     0.042174       0.625693
55     0.108415       0.637033
60     0.934518       0.928604 <------ similar by coincidence
```

Both are completely wrong. The correct answer is $x_{60} = 0.315445$.

How to get this right using a computer

Could use rational arithmetic, with arbitrary-size integers for the numerators and denominators, keeping then in lowest terms.

But: The computation of x_n runs out of memory for $n \geq 30$ or so.

With 1Gb of memory. This is a lot of memory!

Pause to run something in the computer.

How to get this right using a computer

Code numbers using infinitely many digits.

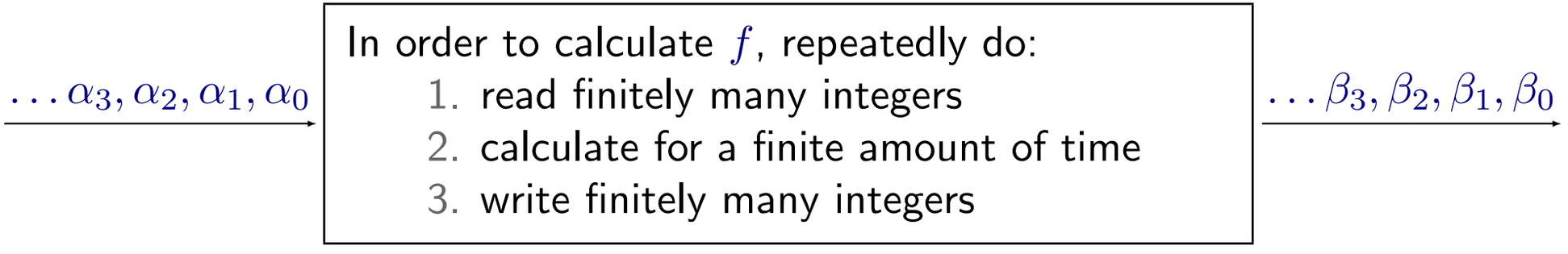
Pause to run something in the computer.

Computing with infinite sequences

$\mathbb{Z}^{\mathbb{N}}$ space of infinite sequences of natural numbers.

Schematic model of computation for functions $f\colon \mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$:

Given $\alpha \in \mathbb{Z}^{\mathbb{N}}$, calculate $\beta = f(\alpha) \in \mathbb{Z}^{\mathbb{N}}$ as follows:

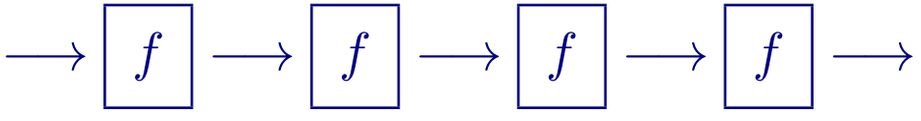


Example

Suppose $f : \mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$ codes the logistic map $x \mapsto 4x(1-x)$ in decimal notation.

Want to calculate $f(f(f(f(x_0))))$.

We get the following pipeline:



Given a pipeline of digits for x_0, this will produce a pipeline for digits for x_4.

NB. In the programming language I am using, you still write

$$x_4 = f(f(f(f(x_0)))).$$

The pipeline is created by the internal mechanisms of the language.

First continuity theorem

If $f : \mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$ is computable in the above fashion, then finite segments of the output can only depend on finite segments of the input.

To make this precise, for $\alpha, \beta \in \mathbb{Z}^{\mathbb{N}}$ and $p \in \mathbb{N}$, define

$$\alpha =_p \beta \iff \alpha_n = \beta_n \text{ for all } i < p.$$

We say that α and β agree with precision p.

First continuity theorem

Theorem. If $f\colon \mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$ is computable in the above fashion, then for every input $\alpha \in \mathbb{Z}^{\mathbb{N}}$ and every desired output precision $p \in \mathbb{N}$ there is a sufficient input precision $q \in \mathbb{N}$ such that

$$\alpha =_q \beta \implies f(\alpha) =_p f(\beta).$$

To calculate $f(\alpha)$ with precision p, it is enough to know α with precision q.

First continuity theorem

Theorem. If $f\colon \mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$ is computable in the above fashion, then for every input $\alpha \in \mathbb{Z}^{\mathbb{N}}$ and every desired output precision $p \in \mathbb{N}$ there is a sufficient input precision $q \in \mathbb{N}$ such that

$$\alpha =_q \beta \implies f(\alpha) =_p f(\beta).$$

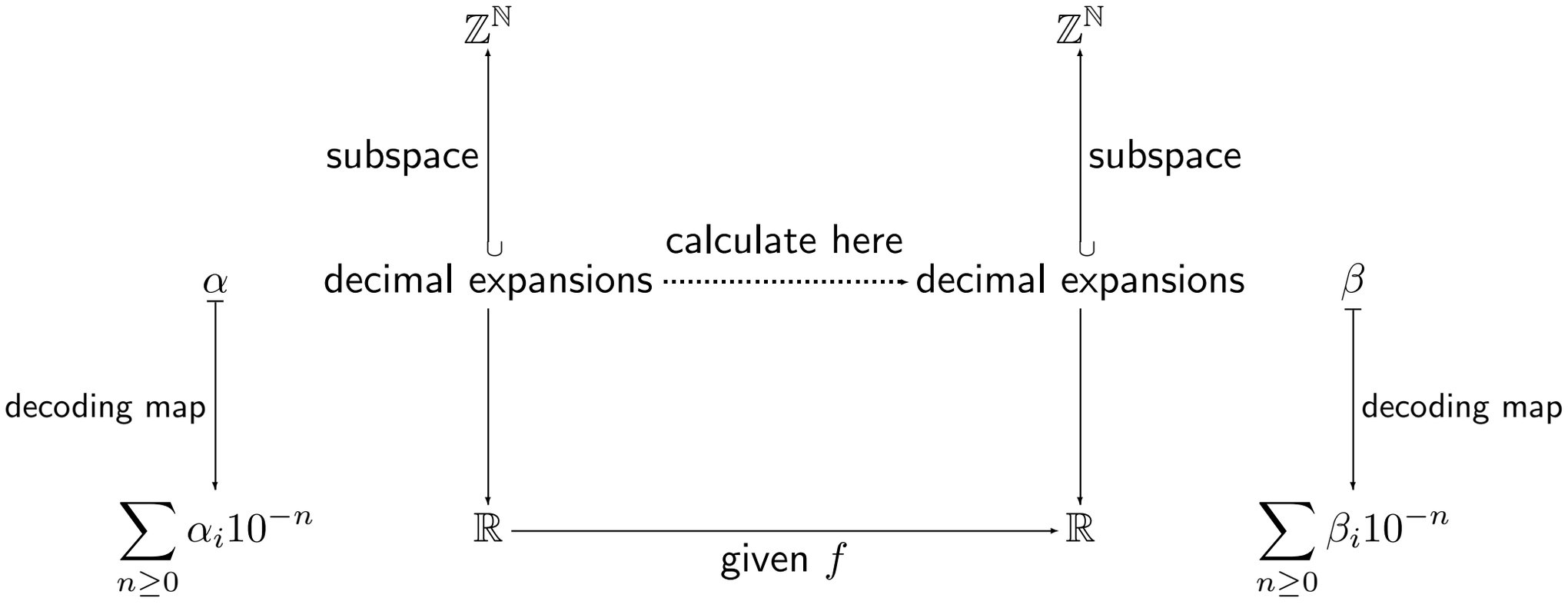
To calculate $f(\alpha)$ with precision p, it is enough to know α with precision q.

Compare with:

Definition. A function $f\colon \mathbb{R} \to \mathbb{R}$ is called continuous iff for every input $x \in \mathbb{R}$ and every desired output precision $\epsilon > 0$ there is a sufficient input precision $\delta > 0$ such that

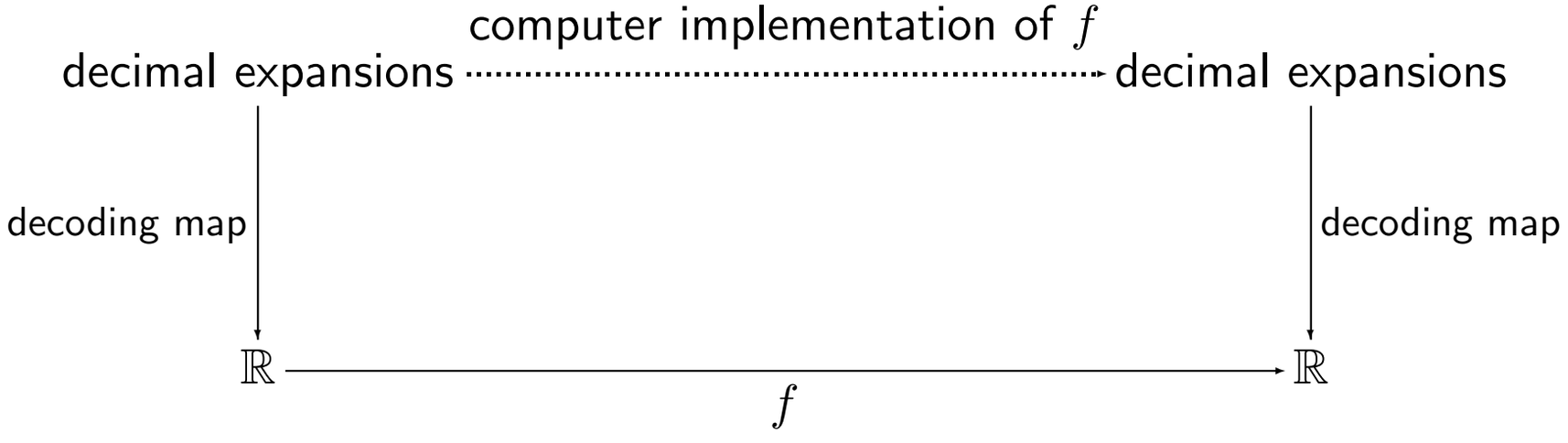
$$|x - y| < \delta \implies |f(x) - f(y)| < \epsilon.$$

Computing with decimal notation



Assume $\alpha_0 \in \mathbb{Z}$ and $\alpha_{i+1} \in \{0, \ldots, 9\}$.

Computing with decimal notation



Second Continuity Theorem. If f has a computer implementation, then it is continuous.

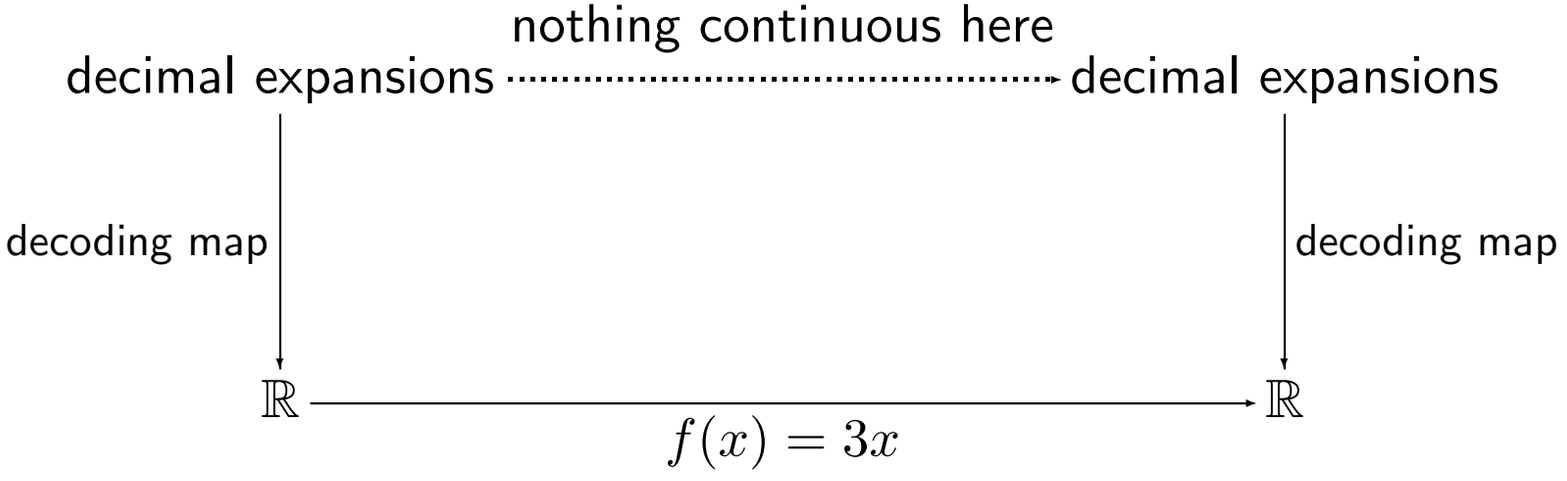
But decimal notation isn't good for infinite computation

This was discovered by Brouwer in the 1920's.

Not even multiplication by three can be computed.

But decimal notation isn't good for infinite computation

This was discovered by Brouwer in the 1920's.



Suppose the input starts 0.333333.

- The first digit of the output must be 0 if we eventually read a digit < 3.
- The first digit of the output must be 1 if we eventually read a digit > 3.
- While we read digits $= 3$, we cannot produce any output for lack of information.

$\Longrightarrow$ If the input is $1/3$, the next output digit depends on *all* input digits.

Other bases have the same problem

So changing to e.g. binary notation doesn't help.

Perhaps shocking, but no reason to abandon hope

Already discussed by Brouwer in the 1920's.

Change the notation for real numbers.

The problem is boundary cases.

There are many theoretical solutions, used in practice.

I will discuss one of them (the one I used above in practice).

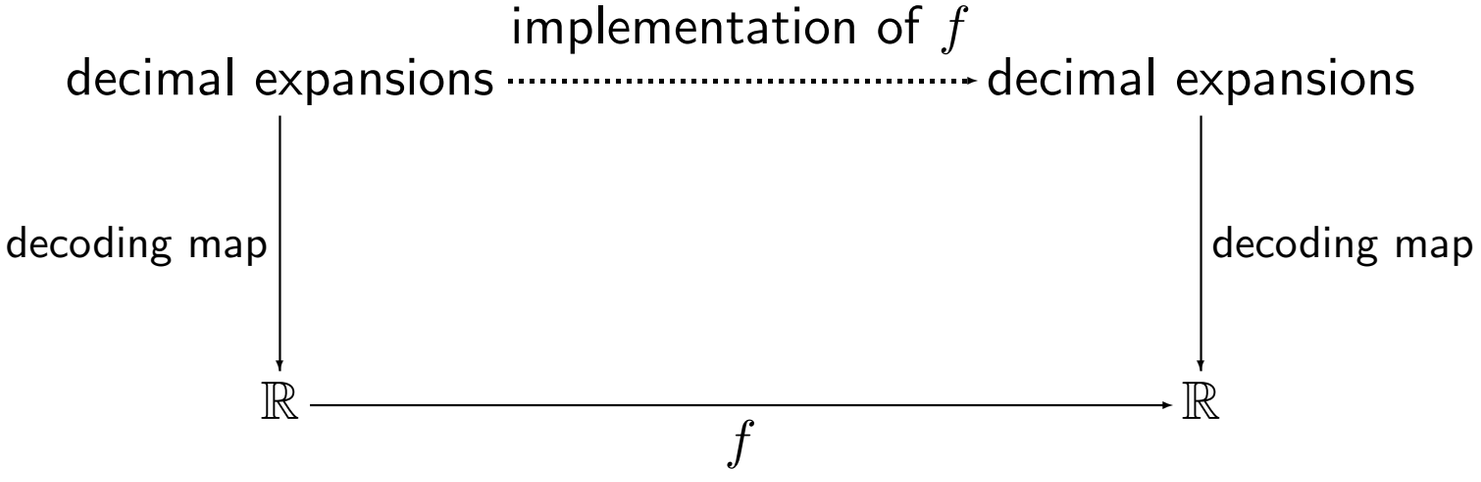
Use binary or decimal notation with negative digits

Actually Cauchy (1840) proposed this to simplify calculation by hand.

$1\bar{8}40 = 1000 - 800 + 40 = 240$.

Intuition. If we are stuck as above, make a reasonable guess, and adjust it later using a negative digit if necessary.

Computing with decimal notation with negative digits



Second Continuity Theorem Still holds. If f has a computer implementation, then it is continuous.

Third Continuity Theorem. If f is continuous, then it has a continuous implementation.

But is this just a hack?

A coding $A \subseteq \mathbb{Z}^{\mathbb{N}}$ with decoding map

$$\text{decode}_A \colon A \to \mathbb{R}$$

is called admissible if given any other coding $B \subseteq \mathbb{Z}^{\mathbb{N}}$ with decoding map

$$\text{decode}_B \colon B \to \mathbb{R},$$

there is a computer program

$$\text{translate} \colon B \to A$$

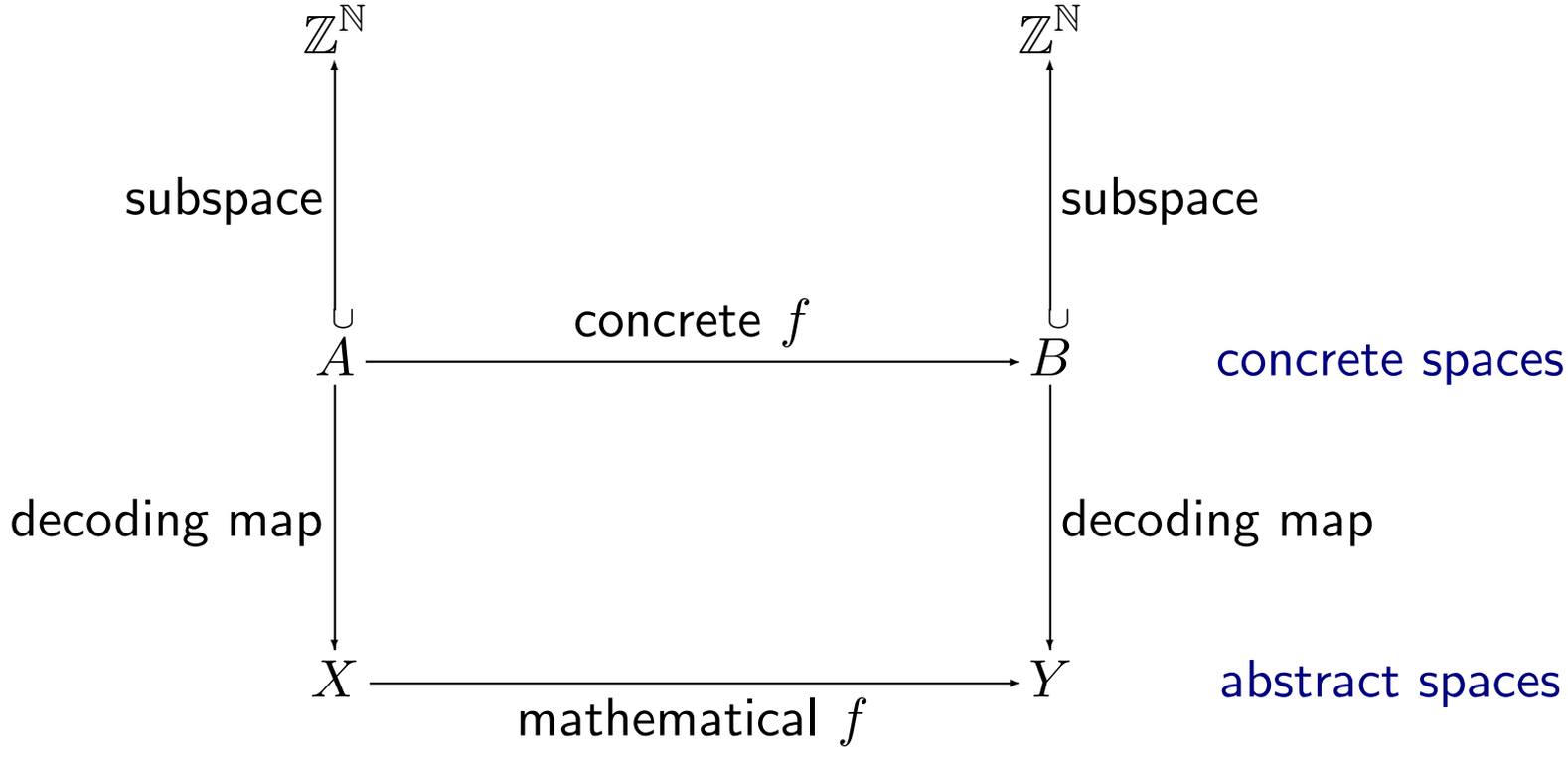
that translates from B codes to A codes.

Theorem.

1. Decimal notation with negative digits is an admissible coding.

2. Any two admissible codings can be translated to each other.

More general picture



Similar continuity theorems.

Notion of admissibility also defined for this more general situation.

Dictionary between topology and computation

Continuous map. Computable function.

Open set. Semi-decidable set.

Closed and open set. Decidable set.

Compact set. Exhaustively searchable set.

Discrete space. Type with decidable equality.

Hausdorff space. Space with semi-decidable apartness.

Take a theorem in topology,
apply the dictionary,
get a theorem in computability theory.

Unfortunately you have to come up with a new proof.

Exhaustive search

Algorithmically check all possibilities in finite time.

Either find something.

Or else report that there is nothing to be found.

Old theorem. A set of natural numbers is exhaustively searchable iff it is finite.

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

Proof: Removed from this intuition (Halting problem, diagonalization).

Computational common wisdom

A set of whatever-you-can-think-of is exhaustively searchable iff it is finite.

Can't always trust common wisdom

E.g. The Cantor space $2^{\mathbb{N}}$ is exhaustively searchable.

Theorem

1. Exhaustively searchable sets are topologically compact.

2. They are precisely the computable images of the Cantor space.

Formal systems for computation

Example 1. Gödel's system T.

Example 2. Platek–Scott–Plotkin's PCF and FPC.

Example 3. Escardó's Real PCF and Real FPC.

Example 4. Practical programming languages based on (2).

ML, OCaml, Haskell, . . .

Robin Milner is mainly responsible for making these theoretical tools available for practical use with many insights.

Some popularization references

1. In Andrej Bauer's *Mathematics and Computation* blog `http://math.andrej.com/`
   (a) Seemingly impossible functional programs
   (b) A Haskell monad for infinite search in finite time
   (c) Running a classical proof with choice in Agda
2. Computing with real numbers represented as infinite sequences of digits in Haskell. `http://www.cs.bham.ac.uk/~mhe/papers/fun2011.lhs`
3. With Paulo Oliva. What Sequential Games, the Tychonoff Theorem and the Double-Negation Shift have in Common. `http://www.cs.bham.ac.uk/~mhe/papers/msfp2010/`

Let's quickly revisit the opening questions and then finish

Question 1

Suppose you have written computer programs for calculating functions

$$f, g \colon \mathbb{Z} \to \mathbb{Z}$$

Can the computer decide, in finite time, whether or not $f = g$?

Here $\mathbb{Z}$ is the set of integers.

Input: Two functions $f, g \colon \mathbb{Z} \to \mathbb{Z}$.

Output: An element of the set $2 = \{0, 1\}$.

Question 2

Consider the set $2^{\mathbb{N}}$ of infinite binary sequences $x_0, x_1, x_2, \ldots, x_n, \ldots$ with $x_i \in 2$.

(For example we can use them for representing real numbers in binary.)

Does it make sense to write programs for computing e.g. functions

$f\colon \quad 2^{\mathbb{N}} \to \mathbb{Z},$ (given a sequence, calculate an integer);

$g\colon \quad 2^{\mathbb{N}} \to 2^{\mathbb{N}},$ (given a real number, calculate another real number).

Question 3

Suppose you have written computer programs for calculating functions

$$f, g \colon 2^{\mathbb{N}} \to \mathbb{Z}$$

Can the computer decide, in finite time, whether or not $f = g$?

Input: Two functions $f, g \colon 2^{\mathbb{N}} \to \mathbb{Z}$.

Output: An element of the set $2 = \{0, 1\}$.

Question 4

Suppose you have written computer programs for calculating functions

$$f, g \colon 2^{\mathbb{N}} \to 2^{\mathbb{N}}$$

Can the computer decide, in finite time, whether or not $f = g$?

Input: Two functions $f, g \colon 2^{\mathbb{N}} \to 2^{\mathbb{N}}$.

Output: An element of the set $2 = \{0, 1\}$.

Question 5

Can we plot the Mandelbrot set?

Given a point that is not in the boundary, can the computer tell whether it is inside or outside the set?

Question 6

It is easy to write a program that systematically lists all provable statements.

(Let's say formally written in the language of set theory.)

We can also write it so that it stops if it ever lists a statement A and also its negation $\neg A$.

Will this program stop?

Concluding discussion

I. General topology is naturally linked with infinite computation because it is a theory of approximation.

II. Category theory is also very useful to understand and apply computation with infinite objects (e.g. final co-algebras, monads).

III. I had to abandon the ambitious plan to also discuss constructive mathematics.

In a nutshell:

1. Give up some logical principles (e.g. excluded middle) and,
2. axioms (e.g. powerset), and
3. develop proofs in the normal mathematical way.
4. Automatically get programs, even if you were not thinking of them (BHK interpretation).
5. Linked to topology: any function you prove to exist will be continuous. (But this is a contentious subject.)