Abstract Computability

Robin Cockett

Department of Computer Science
University of Calgary

April 2016

A presentation of work with:

Ximo Boils
Jonathan Gallagher
Pavel Hrubes
Pieter Hofstra
Steve Lack

Outline

The Church-Turing thesis ...

All reasonable notions of computation are equivalent ...

- *Turing*: A function is said to be *effectively calculable* if its values can be found by some purely mechanical process.
- *Kleene*: Every effectively calculable function is *general recursive* in the sense of Gödel.
- *Gödel*: "The correct definition of *mechanical computability* was established beyond any doubt by Turing."
  (Gödel's collected works.)

λ-definable = Turing computable = General recursive
(Turing complete)

The Church-Turing thesis ...

SO everyone agrees:

....

Because all these different attempts at formalizing the concept of “effective calculability/computability” have yielded equivalent results, it is now generally assumed that the Church-Turing thesis is correct.

....

https://en.wikipedia.org/wiki/Church-Turing_thesis

END OF STORY

The Church-Turing thesis goes viral ...

- An axiom which states that all functions are computable in constructive mathematics
  Constructive Church-Turing thesis
- A function on the natural numbers is computable by a human being iff it is computable by a Turing machine
  Human Church-Turing thesis
- A probabilistic Turing machine can efficiently simulate any realistic model of computation
  Complexity Church-Turing thesis
- A quantum Turing machine can efficiently simulate any realistic model of computation
  Quantum Church-Turing thesis
- All physically computable functions are Turing-computable
  Physical Church-Turing Thesis
- The universe is equivalent to a Turing machine
  Strong Church-Turing Thesis

The Church-Turing thesis ...

This is weird:

it is not science

it is not mathematics

it is not computer science

maybe it *is* religion?

... it certainly deters curiosity in foundations!

Equivalence of computability?

Question:

Is it useful to believe that all notions of computability are equivalent?

By the way:
Mathematically all the evidence points to *inequivalent* notions of computability ...

Equivalence of computability?

Mathematically all the evidence points to *inequivalent* notions of computability ...

- Computing with an oracle (jumps) ...
- Computation within a Topos ...
- Computation within an arithmetic universe ...
- Feasible computing ...
- Abstract computability ...

Slogan: Different notions of computability determined by different notions of partiality ...

Abstract Computability

This talk follows:

[1] J.R.B. Cockett, S. Lack
Restriction categories I: categories of partial maps
TCS 1-2(6) (2002) 223-259

[2] J.R.B. Cockett, P.J.W. Hofstra
Introduction to Turing categories
APAL 156 (2) (2008)183-209

[3] J.R.B. Cockett, P.J.W. Hofstra, P. Hrubes
Total Maps of Turing Categories
ENTCS 308 (2014) 129-146

[4] J.R.B. Cockett
Estonia notes: Categories and Computability (Website)

[5] J.R.B. Cockett, J. Diaz-Boils, J. Gallagher, P. Hrubes
Timed Sets, Functional Complexity, and Computability
ENTCS 286 (2012) 117-137

Partiality

First problem: need a decent description of partial maps!

A **restriction category**, $\mathbb{X}$, is a category equipped with a combinator:

$$\frac{f : A \to B}{\overline{f} : A \to A}$$

such that

[**R.1**] $\overline{f}\, f = f$

[**R.2**] $\overline{f}\,\overline{g} = \overline{g}\,\overline{f}$

[**R.3**] $\overline{f}\,\overline{g} = \overline{\overline{f}\, g}$

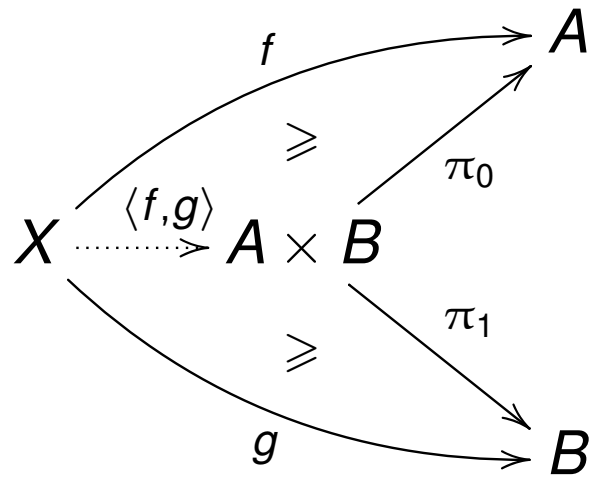
[**R.4**] $f\overline{g} = \overline{fg}\, f$

Think sets and partial functions!

$$\overline{f}\,(x) = \begin{cases} x & f(x) \text{ is defined.} \\ \uparrow & \text{otherwise} \end{cases}$$

Partiality

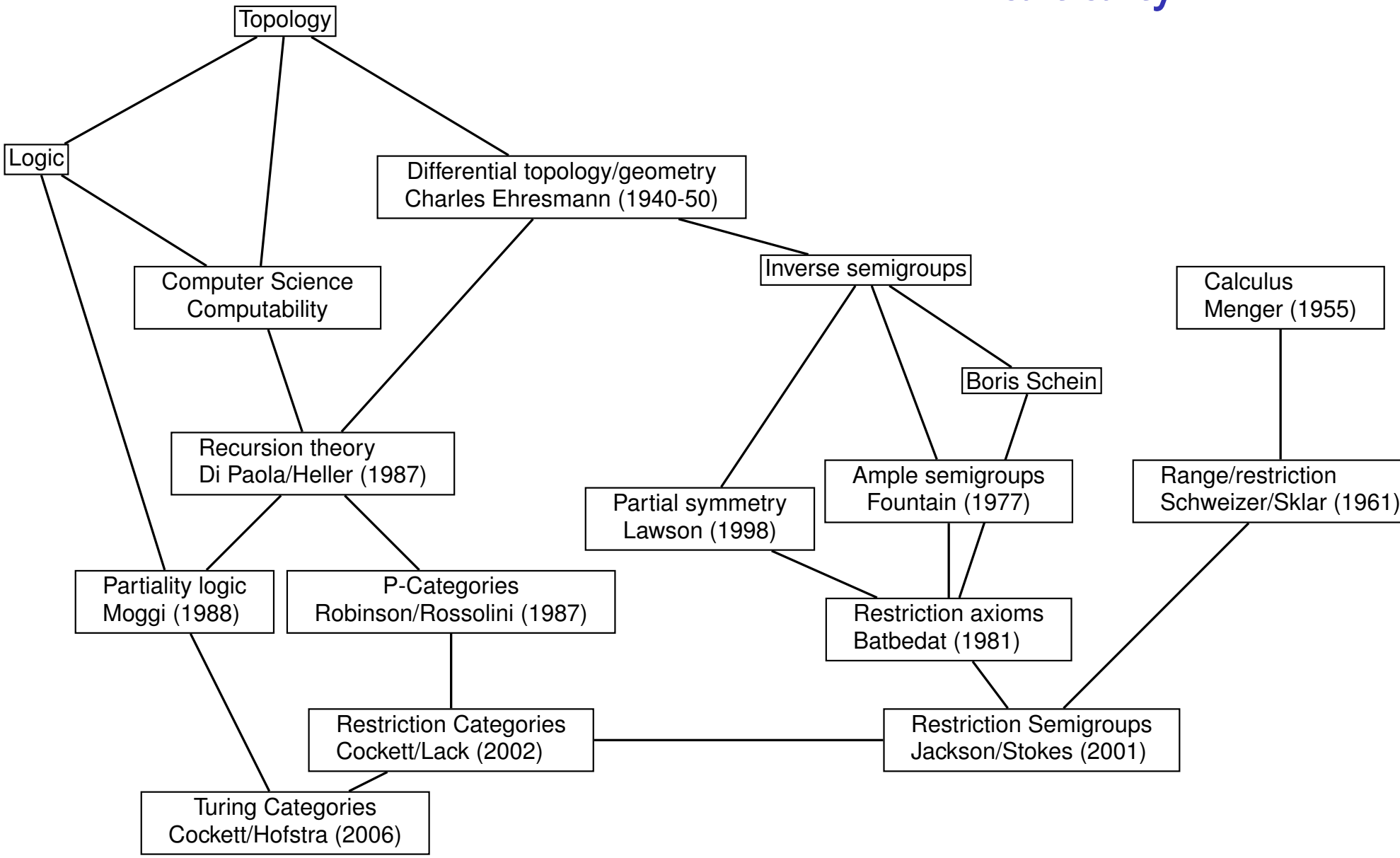
A **Cartesian restriction category** (CRC) has partial products. That is **binary partial products** $A \times B$:



where $\langle f, g\rangle$ is unique such that $\langle f, g\rangle\pi_0 = \overline{g}\, f$ and $\langle f, g\rangle\pi_1 = \overline{f}\, g$.

and a **partial terminal object**, 1 such that there is a unique total map $!_A : A \to 1$ such that any map $f : A \to 1$ has $f = \overline{f}\, !_A$.

Partiality


Topology
Logic
Differential topology/geometry
Charles Ehresmann (1940-50)
Computer Science
Computability
Inverse semigroups
Calculus
Menger (1955)
Boris Schein
Recursion theory
Di Paola/Heller (1987)
Ample semigroups
Fountain (1977)
Range/restriction
Schweizer/Sklar (1961)
Partial symmetry
Lawson (1998)
Partiality logic
Moggi (1988)
P-Categories
Robinson/Rossolini (1987)
Restriction axioms
Batbedat (1981)
Restriction Categories
Cockett/Lack (2002)
Restriction Semigroups
Jackson/Stokes (2001)
Turing Categories
Cockett/Hofstra (2006)

Motivated by calculus ...

Karl Menger was worried by (apparently quite reasonable) equations such as

$$x^2 \frac{1}{x} = x$$

The problem: what is the value at $x = 0$?
What is the antiderivative of this ...

$$\int_{t=0}^{x} t^2 \frac{1}{t} \mathrm{d}t = \begin{cases} \frac{1}{2}x^2 + c_1 & \text{when } x < 0 \\ x^2 + c_2 & \text{when } x > 0 \\ \uparrow & \text{when } x = 0 \end{cases}$$

This is different from the antiderivative of $\int_{t=0}^{x} t \mathrm{d}t$.

How do you keep track of this?

.... and differential geometry!

Charles Ehresmann (1940-50) noted that there was a correspondence between étale groupoids occurring in differential geometry/topology and inverse semigroups.

He realized the importance of partial symmetries to the subject and started to develop their theory ...

He did so using the language of category theory.

Charles Ehresmann ...

Dieudonné describes Ehresmann as:

> *... distinguished by forthrightness, simplicity, and total absence of conceit or careerism. As a teacher he was outstanding, not so much for the brilliance of his lectures as for the inspiration and tireless guidance he generously gave to his research students ...*

Abstract Computability

Second problem: need a decent description of computable maps!

R.A. Di Paola, A. Heller
Dominical categories: recursion theory without elements
JSL, 52 (1987) 595-635

- A programme to understand computability categorically ...
- Establish a (minimal) categorical setting for recursion theory ...

E. Robinson, G. Rossolini, F. Lengyel, L. Birkedal,
G. Longo, E. Moggi, ...

Abstract Computability

A key notion is that of a *partial combinatory algebra* (PCA):

- sometimes attributed to Fefferman (1979)
- full definition appeared in Andy Pitts' PhD. thesis (1981)

$$\bullet : A \times A \to A, \qquad S, K : 1 \to A$$

"application", or $\bullet$, is a partial operation, S and K total elements such that:

$$((S \bullet x) \bullet y) \bullet z = (x \bullet z) \bullet (y \bullet z) \tag{1}$$

$$(K \bullet x) \bullet y = x \tag{2}$$

$$z_{|(S \bullet x) \bullet y)} = z \quad \text{(i.e. } (S \bullet x) \bullet y \text{ is total)} \tag{3}$$

A PCA (in sets) expresses every computable function ...

Abstract Computability

Tip-toe into abstraction:

First step:

View a model of computability as the maps which are computable by a PCA
(Captures all the “classical models” ...)

Second step:

Allow the setting in which a PCA lives to vary ....

Third step:

Directly describe the maps computed by a PCA living somewhere.

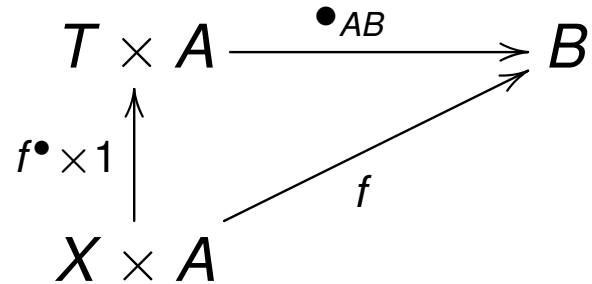
The maps computed by a PCA living somewhere form a **Turing category** ...

Turing categories

A **Turing category**, $\mathbb{C}$, is a Cartesian restriction category with a Turing object T such that for each A and B there is an "application"

$$\bullet_{AB} : T \times A \to B$$

such that for every $f : X \times A \to B$ there is a total map, $f^\bullet : X \to T$, sometimes called an **index** for f, such that



commutes.

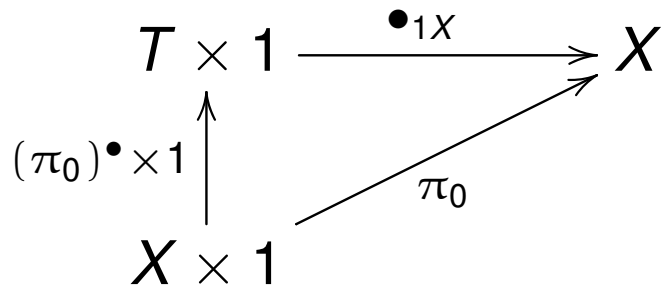
Note: $f^\bullet$ is not necessarily unique!

Turing categories

Lemma

In a Turing category, $\mathbb{C}$, with Turing object T, every object $X \in \mathbb{C}$ is a retract of T.

Consider the diagram:



An object T in a category is said to be **universal** in case every object is a retract of it.

Turing categories

Theorem

(Recognition of Turing categories)
For a cartesian restriction category, $\mathbb{C}$, the following are equivalent:

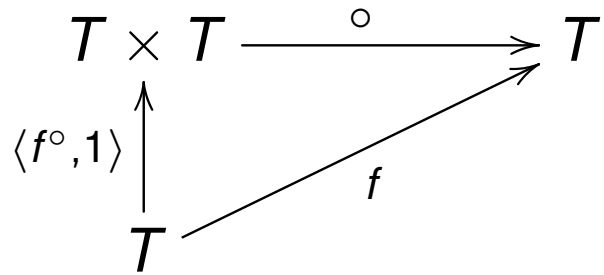
(i) *$\mathbb{C}$ is Turing category;*

(ii) *There is an object T of which is universal, and for which there exists a Turing morphism $\bullet : T \times T \to T$;*

(iii) *There is a universal object T of which has a coding morphism $\circ : T \times T \to T$.*

Turing categories

- $\bullet : T \times T \to T$ is a **Turing morphism** in case for every $f : T \times T \to T$ there is an index $f^\bullet$ as above.
- $\circ : T \times T \to T$ is a **coding morphism** in case for every map $f : T \to T$ there is a total element $f^\circ : 1 \to T$ such that



commutes.

Turing categories

Theorem

Every Turing category is (an idempotent splitting) of the category of computable maps of a partial combinator algebra living somewhere.

- In any Turing category T is a PCA with application $\bullet : T \times T \to T$.
- The computable functions of this PCA generate the maps of the Turing category.

SLOGAN: Turing categories are equivalent to PCAs!

Are PCAs equivalent to Turing machines?

Turing categories

Turing categories are compared by functors:

- A functor $F : \mathbb{X} \to \mathbb{Y}$ between Turing categories is just a Cartesian restriction functor.
- Such a functor F induces a simulation of the PCA, $F(T_{\mathbb{X}})$, by the PCA, $T_{\mathbb{Y}}$, (John Longley).
- Special functors have this simulation an equivalence (i.e. essentially preserve the Turing object) ...

Turing categories

Question:
What do the total maps of a Turing category look like?

Recall:
Every PCA can represent *all* recursive functions ....
.... but this notion of representation is very weak.

Turing categories

The rather technical (necessary and sufficient) answer is in:

[3] J.R.B. Cockett, P.J.W. Hofstra, P. Hrubes
Total Maps of Turing Categories
ENTCS 308 (2014) 129-146

A useable corollary is:

Theorem
Every countable cartesian category with a weak universal object, U, which has more than one element, is the total maps of a Turing category.

A **weak universal object** is an object which has every other object as a subobject.

Turing categories

Examples of total map categories:

(a) PTIME (polynomial time) functions.

(b) LTIME (linear time) functions.

$$U := \mathrm{List}(\mathrm{Bool})$$

To be universal need a linear time encoding $U \times U \to U$:

$$\iota(b : bs, b' : bs') = 0 : b : 1 : b' : \iota(bs, bs')$$
$$\iota(b : bs, []) = 0 : b : \iota(bs, [])$$
$$\iota([], b : bs) = 1 : b : \iota([], bs)$$
$$\iota([], []) = []$$

(powerful objects later ...)

WOW!

There are Turing categories which express the functional complexity classes!

Hitting the functional complexity classes

Question: how do you construct a Turing category whose total maps are *precisely* those of a given functional complexity class?

Size monoids ...

Where is complexity measured?

A **size** monoid (in Set) is a partially ordered commutative monoid $(M, 0, +, \leqslant)$ such that

- $0 \leqslant x$ for all $x \in M$,
- $x \leqslant x'$ and $y \leqslant y'$ implies $x + y \leqslant x' + y'$.

Examples: $\mathbb{N}$,$\mathbb{R}_{\geqslant 0}$, ...

In fact, given any commutative monoid A set $x \leqslant y$ if there is a z with $x + z = y$ then $x \sim y \equiv x \leqslant y \& y \leqslant x$ then $\text{size}(A) = A/\sim$ is the universal size monoid associated with A.

Note: size monoids are orthogonal to commutative groups.

The basic category of timed sets

TSet(M):

Objects: Sets: X, Y, ..

Maps: $(f, |\_|_f) : X \to Y$ where:

- $f : X \to Y$ is a partial function of sets;
- $|\_|_f : X \to M$ is a "timing" map;
- $f(x) \downarrow$ if and only if $|x|_f \downarrow$.

Composition: $(fg, |\_|_{fg})$ where $|x|_{fg} = |x|_f + |f(x)|_g$.

Identity: $(1_X, |\_|_{1_X})$ where $|x|_{1_X} = 0$ for every $x \in X$.

Note: this category is partial order enriched with $f \leqslant g$ meaning $f(x) \downarrow$ whenever $g(x) \downarrow$ and $|x|_f \leqslant |x|_g$.

Intuitively f is faster and more defined than g.

Complexity orders

Complexity theory only measures “time” up to “order” ...

A **complexity order** on a size monoid, M, is a class of monotone functions $\mathcal{C} \subseteq \mathrm{Mon}(M, M)$ such that:

- $\mathcal{C}$ is **down-closed** ($P \in \mathcal{C}$ and $Q(x) \leqslant P(x)$ for all $x \in M$ implies $Q \in \mathcal{C}$);
- $\mathcal{C}$ is closed to **composition** ($I(x) = x \in \mathcal{C}$ and if $P, Q \in \mathcal{C}$ then $PQ \in \mathcal{C}$ where $(PQ)(x) = Q(P(x))$);
- $\mathcal{C}$ is **laxly additive** ($P_1, P_2 \in \mathcal{C}$ implies $\exists Q \in \mathcal{C}$ such that $P_1(x) + P_2(x) \leqslant Q(x)$).

Complexity orders

Special properties of complexity orders ...

A complexity order $\mathcal{C}$ is:

Pointed: if, for all $P \in \mathcal{C}$, $P(0) = 0$.

Additive: in case $P, Q \in \mathcal{C} \Rightarrow P + Q \in \mathcal{C}$, where $P + Q(x) = P(x) + Q(x)$.

Laxly generated: if, for all $P \in \mathcal{C}$ there is a $Q \in \mathcal{C}$ such that $P \leqslant Q$ and $Q(x) + Q(y) \leqslant Q(x + y)$ (lax property).

$$\text{Laxly generated} \Rightarrow \text{Additive} \Rightarrow \text{Laxly additive}.$$

A complexity order is additive iff it contains $I + I$.

Examples of complexity orders

A complexity orders $\mathcal{C}$ is **generated** by a class of functions $\mathcal{F}$ if it is the down-closure of that class, that is

$$\mathcal{C} = \Downarrow \mathcal{F} = \{P | \exists Q \in \mathcal{F}. P \leqslant Q\}.$$

(1) $\Downarrow \{I\} \subseteq \mathrm{Mon}(M, M)$ the **non-increasing complexity order** – this is the smallest complexity order!

(2) $\mathcal{K} = \Downarrow \{\lambda x.x + k | k \in M\} \subseteq \mathrm{Mon}(M, M)$ the **constant complexity order** $-\mathcal{E}_0$.

(3) $\mathcal{L} = \Downarrow \{\lambda x.n \cdot x | n \in \mathbb{N}\} \subseteq \mathrm{Mon}(M, M)$ the **linear functions** (Pointed, laxly generated) – big "O".

(4) $\mathcal{P} = \Downarrow \{\lambda x. \sum_{i=0}^{m} a_i \cdot x^i\} \subseteq \mathrm{Mon}(\mathbb{N}, \mathbb{N})$ the **polynomial functions**. (additive) – $\mathcal{E}_2$.

(5) $\mathcal{P}^* = \Downarrow \{\lambda x. \sum_{i=1}^{m} a_i \cdot x^i\} \subseteq \mathrm{Mon}(\mathbb{N}, \mathbb{N})$ the **pointed polynomial functions**. (Pointed, laxly generated).

Examples of complexity orders

(7) ELEMENTARY: the **Kalmar elementary** functions (ELEMENTARY $= \mathcal{E}_3$).

(8) $\mathcal{E}_n$: the n^{th} class in the **Grzegorczyk hierarchy**.

(9) Prim: the primitive recursive functions, Prim $= \bigcup_{n \in \mathbb{N}} \mathcal{E}_n$.

All laxly generated ...

Complexity equivalence

We say that $f \in \mathrm{TSet}(M)$ has **better $\mathcal{C}$-complexity** than g, $f \leqslant_{\mathcal{C}} g$, when:

- f is at least as defined as g, that is $g(x)\downarrow$ implies $f(x)\downarrow$;
- f is as $\mathcal{C}$-fast as g, that is $|x|_f \leqslant P(|x|_g)$ for some $P \in \mathcal{C}$.

Lemma

$\_ \leqslant_{\mathcal{C}} \_$ is a preorder on parallel M-timed maps.

For transitivity suppose $|x|_f \leqslant P(|x|_g)$ and $|x|_g \leqslant Q(|x|_h)$ then as P is monotone:

$$|x|_f \leqslant P(|x|_g) \leqslant P(Q(|x|_h))$$

We say that f is $\mathcal{C}$**-equivalent** to g, $f \equiv_C g$, when $f \leqslant_{\mathcal{C}} g$ and $g \leqslant_{\mathcal{C}} f$.

Complexity equivalence

In fact $f \equiv_{\mathcal{C}} g$ is a congruence of $\mathrm{TSet}(M)$ because $f \leqslant_{\mathcal{C}} g$ is a preorder enrichment:

Lemma

In $\mathrm{TSet}(M)$*:*

(i) *If* $f \leqslant_{\mathcal{C}} g$ *and* $h \leqslant_{\mathcal{C}} k$ *then* $fh \leqslant_{\mathcal{C}} gk$*;*

(ii) $f \equiv_{\mathcal{C}} g$ *is a congruence.*

If $|x|_f \leqslant P(|x|_g)$ and $|y|_h \leqslant Q(|x|_k)$ then

$$|x|_f + |f(x)|_g \leqslant P(|x|_g) + Q(|f(x)|_k) \leqslant R(|x|_g + |f(x)|_k)$$

last step uses lax additivity.
The quotient by this congruence gives $\mathrm{TSet}_{\mathcal{C}}(M)$ ...

E.g. $\mathrm{TSet}_{\mathcal{K}}(M)$ has maps equivalence classes of functions which perform within a constant bound of each other.

$\mathrm{TSet}_{\mathcal{C}}(M)$ as a restriction category

Theorem

When $\mathcal{C}$ is an additive complexity order, $\mathrm{TSet}_{\mathcal{C}}(M)$ *is a restriction category.*

The restriction is given by:

$$\frac{(f, |\_|_f) : X \to Y}{\overline{(f, |\_|_f)} = (\bar{f}, |\_|_f) : X \to X}$$

Note: for $\bar{f}\, f = f$ one needs $\mathcal{C}$ is additive:

$$|x|_{\bar{f}\, f} = |x|_{\bar{f}} + |x|_f = |x|_f + |x|_f = 2|x|_f.$$

$\mathrm{TSet}_{\mathcal{C}}(M)$ as a restriction category

Where are we? ... not there!

The total maps in $\mathrm{TSet}(M)_{\mathcal{C}}$ have (up to $\mathcal{C}$) zero cost. For a pointed complexity order they are *exactly* the zero time cost total functions.

This does not look right from a complexity standpoint!

... but the problem will solve itself!

... as a Cartesian restriction category

In $\mathrm{TSet}_{\mathcal{C}}(M)$ the restriction terminal object:
$1 := \{()\}$ and, for each A, $!_A(a) := ()$ with $|a|_{!_A} := 0$.

Note: Given an $f : A \to 1$, that whenever f is defined on a that $|a|_{\overline{f}\,!_A} = |a|_{\overline{f}} + |a|_{!_A} = |a|_f$.

$A \times B$ is the cartesian product of the sets, and $|(x, y)|_{\pi_i} = 0$.

$\langle f, g \rangle$ is the pairing map defined when both f and g are, with $|x|_{\langle f,g \rangle} = |x|_f + |x|_g$.

Note: $\Delta = \langle 1_A, 1_A \rangle : A \to A \times A$ is a zero cost map and so has a zero cost partial inverse. Thus, $\mathrm{TSet}_{\mathcal{C}}(M)$ is a *discrete* Cartesian category.

... as a join restriction category

When does $\mathrm{TSet}_{\mathscr{C}}(M)$ have joins?

Answer: When M has meets (infima)!

When $f \smile g$ that is $\overline{f}\, g = \overline{g}\, f$ define

$$(f, |\_|_f) \vee (g, |\_|_g) = (f \vee g, |\_|_f \wedge |\_|_g)$$

where

$$|x|_{f \vee g} = \begin{cases} |x|_f & f(x) \downarrow \text{ and } g(x) \uparrow \\ |x|_g & f(x) \uparrow \text{ and } g(x) \downarrow \\ |x|_f \wedge |x|_g & f(x) \downarrow \text{ and } g(x) \downarrow \end{cases}$$

... as a distributive category

TSet$(M)_{\mathcal{C}}$, for any size monoid M, has coproducts and is a *distributive* restriction category.

Every distributive restriction category is an *extensive* restriction category. Extensive restriction categories always have *disjoint* joins of partial maps.

.... as an itegory

When can we iterate maps?

A category has iteration when it is *traced* on the coproduct:

$$\frac{h : A \to A + B}{h^\dagger : A \to B}\ \text{Trace}$$

In an extensive restriction category $h : A \to A + B$ may be expressed by two disjoint maps:

$$\frac{f : A \to A \quad g : A \to B \quad f \perp g}{f \sqcup g = h : A \to A + B}$$

the trace can then be expressed as a “Kleene wand” ...

... as an itegory

A **Kleene wand** is a combinator:

$$\frac{f : A \to A \quad g : A \to B \quad f \perp g}{f \dagger g : A \to B}\ \text{Kleene Wand}$$

which satisfies:

[W.1] When $f \perp h$ then $(fg) \dagger h = h \sqcup f((gf) \dagger (gh))$;

[W.2] When $f \perp g$, $g \perp h$, and $f \perp h$ then $(f \sqcup g) \dagger h = (f \dagger g) \dagger (f \dagger h)$;

[W.3] When $f \perp g$ then $(f \dagger g)h = f \dagger (gh)$;

[W.4] When $f \perp g$ then $1_A \times (f \dagger g) = (1_A \times f) \dagger (1_A \times g)$;

[W.5] When $f \leqslant f'$, $g \leqslant g'$, and $f' \perp g'$ then $f \dagger g \leqslant f' \dagger g'$.

... as an itegory

The first identity allows finite unwinding of the iteration:

$$f \dagger g = g \sqcup (g \dagger fg) = g \sqcup fg \sqcup (g \dagger ffg) = ...$$

The third identity tells us that $f \dagger g = (f \dagger \overline{g}\,)g$. Here $f \dagger \overline{g}$ is primitive form in which $\overline{g}$ may be regarded as a guard. Intuitively f is iterated until the guard g is encountered.

In sets and partial functions the canonical Kleene wand may be expressed as $f \dagger g = \bigsqcup_{i=0}^{\infty} f^i g$.

... as an itegory

Theorem

$\mathrm{TSet}_{\mathcal{C}}(M)$ *has iteration whenever* $\mathcal{C}$ *is laxly generated.*

We define $f \mathbin{*\!\!\dagger} g(x) := g(f^n(x))$ when this is defined for some n (which must be unique) and

$$|x|_{f \mathbin{*\!\!\dagger} g} := \left(\sum_{i=0}^{n-1} |f^i(x)|_f\right) + |f^n(x)|_g.$$

The only difficulty is to show that iteration is well-defined with respect to $\mathcal{C}$-equivalence of maps.

... as an itegory

Recall $\mathcal{C}$ is generated by a class of functions which are lax in the sense that $P(m) + P(n) \leqslant P(m+n)$ (we always have $0 \leqslant P(0)$).

$$
\begin{aligned}
|x|_{f \dagger g} &= |x|_{f^n g} \\
&= |x|_f + |f(x)|_f + \ldots + |f^n(x)|_g \\
&\leqslant |x|_{f'} + P(|x|_{f'}) + |f'(x)|_{f'} + P(|f'(x)|_{f'}) + \ldots + |f'^n(x)|_{g'} + Q(|f'^n(x)|_{g'} \\
&= |x|_{f'} + + |f'(x)|_{f'} + \ldots + |f'^n(x)|_{g'} + P(|x|_{f'}) + P(|f'(x)|_{f'}) + \ldots + Q(|f'^n(x)|_{g'}) \\
&= |x|_{f' \dagger g'} + P(|x|_{f'}) + P(|f'(x)|_{f'}) + \ldots + Q(|f'^n(x)|_{g'}) \\
&\leqslant |x|_{f' \dagger g'} + (P+Q)(|x|_{f'}) + (P+Q)(|f'(x)|_{f'}) + \ldots + (P+Q)(|f'^n(x)|_{g'}) \\
&\leqslant |x|_{f' \dagger g'} + (P+Q)(|x|_{f'} + |f'(x)|_{f'} + \ldots + |f'^n(x)|_{g'}) \\
&\leqslant |x|_{f' \dagger g'} + (P+Q)(|x|_{f' \dagger g'}).
\end{aligned}
$$

$\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$

Spitting idempotents gives size ..

Objects: Restriction idempotents in $\mathrm{TSet}_{\mathcal{C}}(M)$ – timed partial identity;

Maps: $f : e_1 \to e_2$ with $e_1 f e_2 = f$.

As idempotents already split in sets and partial maps it suffices to consider objects whose underlying partial identity is actually the identity. This makes the timing of the idempotent which is split key ... and provides a notion of "size".

Intuitively: the size of an element is the cost of "reading" the element but doing nothing with it.

$\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$

An object in $\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$ is a **sized set**: an A with a map which assigns to each element $a \in A$ a "size" $\|a\|$ ....

A timed map $f : e \to e'$ between two sized sets is a timed partial map such that $efe' =_{\mathcal{C}} f$ that is:

$$\|x\| + |x|_f + \|f(x)\| \leqslant P(|x|_f)$$

This requires that the timing of a map:

Cannot be "faster" than the time required to read its input and produce its output!!

$\mathrm{Total}(\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M)))$

What does it mean for a map to be total?

A map, $f : e \to e'$, in $\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$, is total if and only if

$$\bar{f} =_{\mathcal{C}} e$$

or

$$|x|_f \leqslant P(\|x\|)$$

which means:

- The timing of f is $\mathcal{C}$-bounded by the size of its input.
- *OR* f is total if and only if it is .....

..... in the complexity class determined by $\mathcal{C}$*!*

WOW!

$\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$

Theorem

For $\mathcal{C}$ a laxly generated complexity order, $\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}}(M))$ is a discrete distributive restriction category with iteration (i.e. an itegory) whose total maps are precisely those maps whose time complexity lies in $\mathcal{C}$.

i.e. the total maps are $\mathcal{C}$**-timed** maps.

Getting non-zero size ..

How do we ensure all sizes are non-zero?

Answer: Move to the slice $\mathsf{Split}(\mathsf{TSet}_{\mathcal{C}}(M))/\star$.

$\star$ is the subobject $1 = \{()\}$ determined by the idempotent $\star : 1 \to 1$ where $|()|_\star = 1$.

Lemma

If $\mathcal{C}$ is a pointed complexity order an object $Y \in \mathsf{Split}(\mathsf{TSet}_{\mathcal{C}}(M))$ has a total map to $\star$ if and only if each element of Y has a non-zero size.

... not yet PTIME

The total maps in $\mathrm{Split}(\mathrm{TSet}_{\mathcal{P}^*}(\mathbb{N}))/\star$ are by no means the standard PTIME maps of complexity theory!

- Not computable
- Their "timing" is arbitrary.

To obtain a standard notion of PTIME maps we must demand that the maps are *realized* by a machine.

.... for example by a Turing machine with the standard timing.

We now develop an abstract theory of program machines
... and show how to carve out a Turing category whose total maps are PTIME.

Powerful objects

In a Cartesian restriction category ...

A is a **powerful object** in case A is inhabited and there are total maps $s_\times : A \times A \to A$ and partial maps $P_0, P_1 : A \to A$ such that $s_x\langle P_0, P_1\rangle = 1_{A\times A}$.

The powerful object will be the **data** for our a programs ...

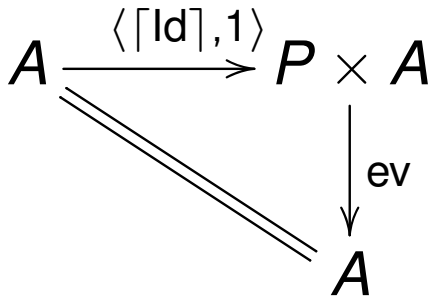
A non-trivial powerful object is List(Bool) with size given by $\|x\| = 1 + 2 \cdot \mathrm{len}(x)$. There are then *linear time* maps $s_\times$, P_0, and P_1 which code and decode pairs:

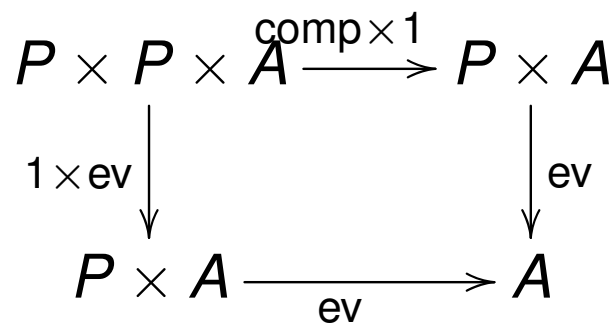
$$s_\times(b{:}bs, b'{:}bs') = 0{:}b{:}1{:}b'{:}s_\times(bs, bs')$$
$$s_\times([], b'{:}bs') = 1{:}b'{:}s_\times([], bs')$$
$$s_\times(b{:}bs, []) = 0{:}b{:}s_\times(bs, [])$$

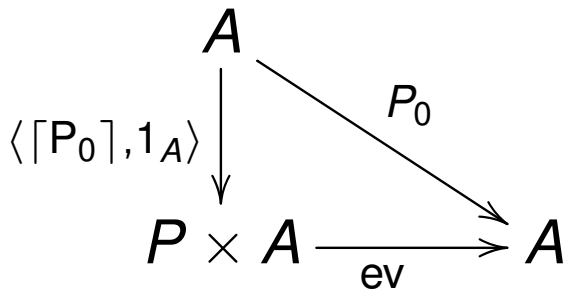
$$P_0(0{:}b{:}rs) = b{:}P_0(rs)$$
$$P_0([]) = []$$
$$P_1(1{:}b'{:}rs') = b'{:}P_1(rs')$$
$$P_1([]) = []$$

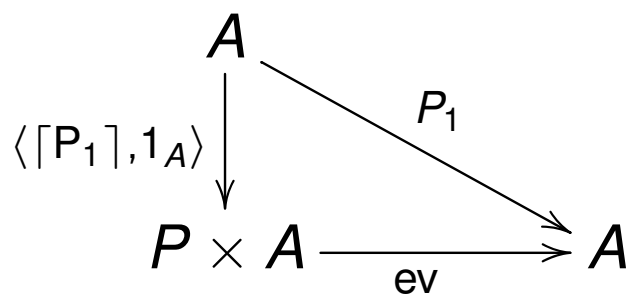
A-program objects

Given a powerful object, A, an A-**program object**, P, is an object which has total operations comp, pair : $P \times P \to P$ together with total points $\lceil \mathrm{P}_0 \rceil, \lceil \mathrm{P}_1 \rceil, \lceil \mathrm{Id} \rceil : 1 \to P$ and a partial **evaluation** map ev : $P \times A \to A$ such that:

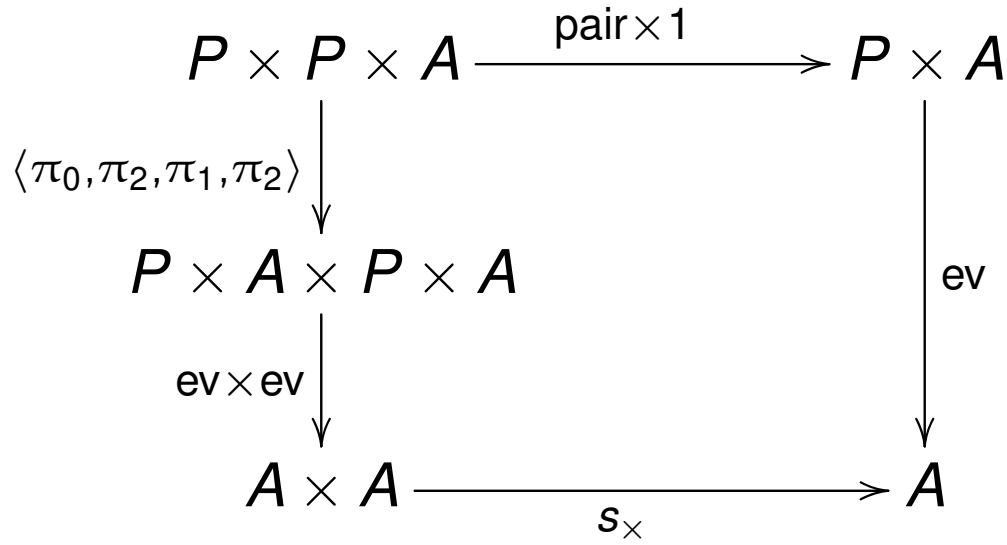








A-program objects



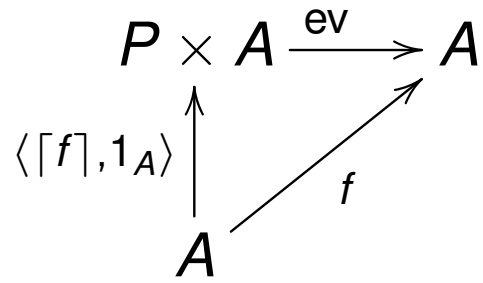
An A-program object P is intuitively a PCA in which the "code" and "data" have been separated ...

P is a **machine** program object in case ev $=$ step $\dagger$ halt where $\overline{\text{step}} \vee \overline{\text{halt}} = 1_{P \times A}$ and step and halt have low complexity. In other words ev is a trace of a machine transition which is *total* and of low complexity.

Of course, this does not mean eval is total!

P-programmable maps

A map $f : A \to A$ is said to be P-**programmable** in case there is an element $\lceil f \rceil : 1 \to P$ such that



If X and Y are retracts of A then a map $h : X \to Y$ is P-**programmable** if the map

$$A \xrightarrow{r_X} X \xrightarrow{\quad h \quad} Y \xrightarrow{s_Y} A$$

is P-programmable. An object is P-programmable when it's identity map is.

Theorem

The subcategory of P-programmable maps, $\mathrm{Prog}_P(\mathbb{X})$*, of any Cartesian restriction category* $\mathbb{X}$*, is a Cartesian restriction subcategory.*

Program objects and Turing structure

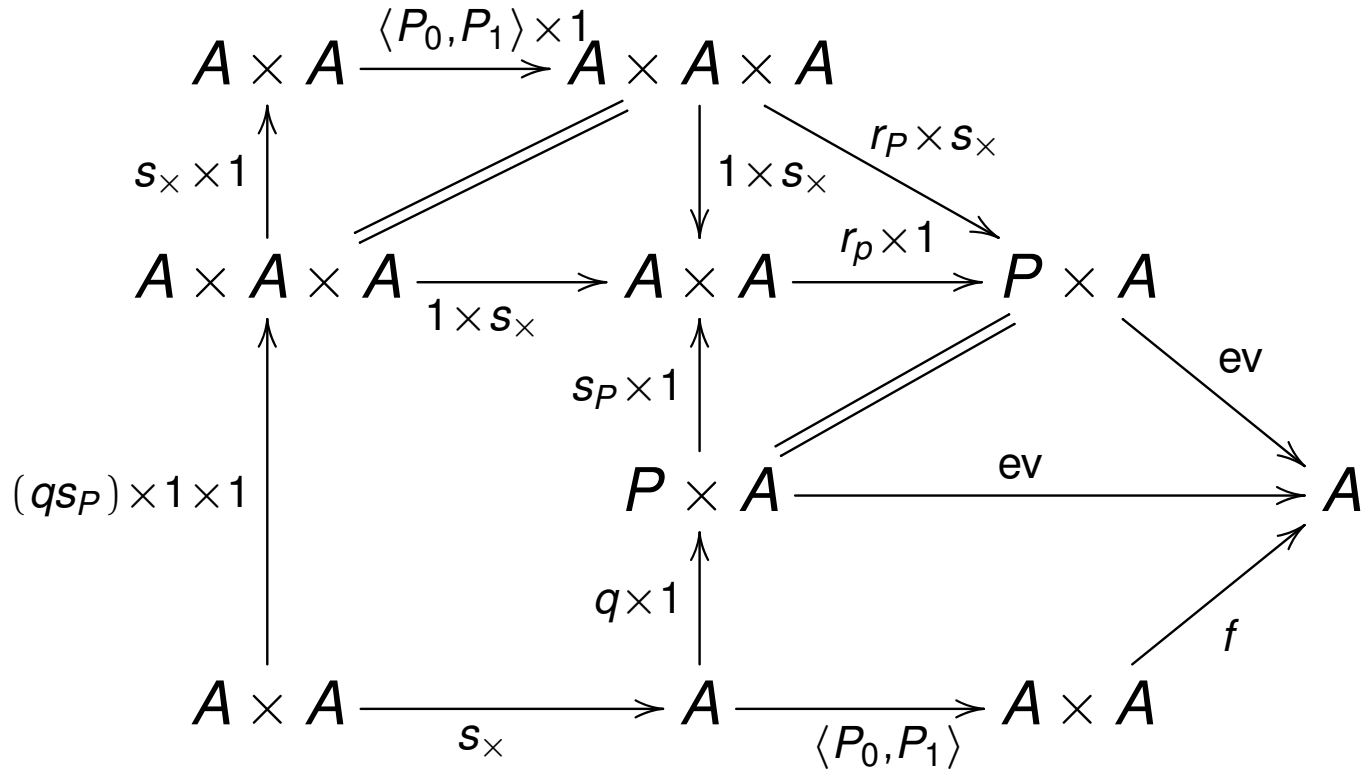
Now turn a program object back into a PCA by insisting it can simulate itself!

Theorem

If $\mathbb{X}$ *is a cartesian restriction category with an inhabited powerful object A and an A-program object P such that P can be simulated – that is P itself is a programmable object and* comp, pair, ev, $\lceil \mathsf{P}_0 \rceil$, $\lceil \mathsf{P}_1 \rceil$, *and* $\lceil \mathsf{Id} \rceil$ *are all programmable – then* $\mathsf{Prog}_P(\mathbb{X})$ *is a Turing category with Turing object A.*

Program objects and Turing structure

Define the program $q := \lceil \langle P_0, P_1 \rangle f \rceil$ then



where $(q \times 1)s_P s_\times$ is the required total map and

- $\bullet := (\langle P_0, P_1 \rangle \times 1)(r_P \times s_\times)\mathrm{ev}$.

Program objects and Turing structure

This is a fundamentally familiar theorem!!!

An abstract version of the constuction of a universal Turing machine!!

- $s_P : P \to A$ is the encoding of the description of a machine into data ...
- $\bullet := (\langle P_0, P_1 \rangle \times 1)(r_P \times s_\times)\mathrm{ev}$ is the running of the universal machine on a description of a machine and its data ...

Program objects and Turing structure

There is a standard theorem:

Theorem (Cost of universal simulation)

There is a universal Turing machine whose evaluation ev *for all Turing machines M, and inputs x, has*

$$|M, x|_{\mathsf{ev}} \leqslant K \cdot \|M\| \cdot |x|_M \cdot log(|x|_M)))$$

This means we can simulate the evaluation of a Turing machine within PTIME.

Corollary

Turing machines with their standard costing are a program object in $\mathsf{TSet}_{\mathcal{P}^*(\mathbb{N})/\star}$ *which can be simulated and thus, the programmable maps form a Turing category whose total maps are exactly the PTIME maps.*

Program objects and Turing structure

HOWEVER ...

NOTE: the overhead for simulating a Turing machine means we cannot use this simulation theorem for generating a Turing category whose total maps are linear time!!

.... BUT we know such a category exists!!
so is there a program object for LTIME?

Program objects and Turing structure

Sketch of an imperfect solution:

Consider the powerful object consisting of λ-terms in β-normal form, N:

$$s_\times : N \times N \to N; (n, m) \mapsto \langle n, m \rangle := \lambda p.pnm$$

$$P_0 : N \to N; t \mapsto t(\lambda xy.x) \qquad P_1 : t \mapsto t(\lambda xy.y).$$

This is the usual pairing ...

Use the usual size $\|\_\| : N \to \mathbb{N}$

Program objects and Turing structure

Use as the program object, L, the set of all λ-terms with the usual size. For evaluation:

$$\mathrm{ev} = \mathrm{app}(\mathrm{step} \dagger \mathrm{normal}) : L \times N \to N$$

is λ-application costed by counting rewrite steps (after the application) and adding in the sizes of the input and output term:

$$\mathrm{step} \sqcup \mathrm{normal} : L \to L + N$$

where step is a (leftmost-outermost) one-step β-reduction: normal extracts the normal form term.

Note: This is not a machine in $\mathrm{TSet}_{\mathcal{L}}/\star$ but ev is present ...

Program objects and Turing structure

We must show that there is a section

$$[\![\_]\!]_z : L \to N; t \mapsto \lambda z.[\![t]\!]_z$$

where z is a new variable:

$$\begin{aligned} [\![ts]\!]_z &= (z[\![t]\!]_z)(z[\![s]\!]_z) \\ [\![\lambda x.t]\!]_z &= \lambda x.[\![t]\!]_z \\ [\![x]\!]_z &= x \end{aligned}$$

Note: that $[\![t]\!]_z(\lambda x.x) = t$ in linear time so is programmable. Evaluation is programmable by $\lambda xy.(x(\lambda z.z))y$!

Program objects and Turing structure

MORAL: to capture low complexity settings need to choose your program object/machine model carefully ...

Turing machines often don't work ...

Question: can this be done using a linear time machine? (i.e. defined by iteration)?

CONJECTURE: pointer machines also allows LINEAR to be seen as a Turing category.

(Another well-known example:
.... to capture LOGSPACE have to use transducers.)

What about natural numbers?

Kronecker said:

> "God gave us the natural numbers:
> ....everything else was made by man."

An alternative and powerful way of describing computation is to use initial (and final) datatypes ...

Theorem

TSet$(\mathbb{N})_{\mathsf{Prim}}/\star$ *has a natural number object*
.... BUT for no complexity class smaller than Prim *is there one.*

Very limiting .....

What about natural numbers?

Solution: reintroduce datatypes via “safe recursion”!

Need a more complex fibrational setting with total category:

TSet(2) (think simple fibration):

Object: Pairs of sets (X, A), X is context, A is local object, with two sizes $\|x\|$, context size, and $\|(x, a)\|$ local size
such that for each x there is a λ_x such that $\|x, a\|_f \leqslant \lambda_x$ (bounded).

Maps: Pairs of partial functions $(f, g) : (X, A) \to (Y, B)$ where $f : X \to Y$ (global map), $g : X \times A \to B$ (local map) $\overline{(f \times 1)}\, g = g$
with context cost $|x|_f$ and local cost $|x, a|_g$ such that for each x there is a λ_x with $|x, a|_g \leqslant \lambda_x$ (locally bounded).

What about natural numbers?

Identity: $1_{(X,A)} : (X, A) \to (X, A); (x, a) \mapsto (x, \pi_1(x, a))$
where $|x|_1 = \|x\|$ and $|a, x|_1 = \|x, a\|$

Composition: $(f, g)(f', g') := (ff', \langle \pi_0 f, g \rangle g')$ with
$|x|_{(f,g)(f',g')} := |x|_f + |f(x)|_{f'}$ and
with $|x|_{ff'} = |x|_f + |f(x)|_{f'}$ and
$|(x, a)|_{(f,g)(f',g')} = |x|_f + |x, a|_g + |f(x), g(x, a)|_{g'}$
(this is locally bounded!)

Think simple fibration ...

What about natural numbers?

There is an obvious restriction functor

$$\partial : \mathrm{TSet}(2) \to \mathrm{TSet}_{\mathcal{P}^*}/\star$$

this is a (latent) fibration. Which has a *comprehension*:

$$C : \mathrm{TSet}(2) \to \mathrm{TSet}_{\mathcal{P}^*}/\star;\ (f, g) \mapsto \langle \pi_0 f, g \rangle$$

with $|(x, a)|_{C(f,g)} = |x|_f + |x, a|_g$.

This makes it a Polar setting ...

What about natural numbers?

So we can define the **fold** function for "safe" phrases:

$$\frac{(1, a) : (X, 1) \to (X, C) \quad (1, b) : (X, C) \to (X, C)}{\mathrm{fold}(a, b) : (X \times \mathbb{N}, 1) \to C}$$

$\mathbb{N}$ has size $\|n\| = n + 1$, zero and succ have cost 1 ...

$$|(n, x)|_{\mathrm{fold}(a,b)} = |x, ()|_a + \sum_{i=1}^{n} |b^n(a(x))_x, x|_b \leqslant |x, ()|_a + n \cdot \lambda_x(b)$$

so it is polynomial.

Note: Showing this is a semantics for safe recursion (with safe data – in particular binary numbers) proves the consistency of safe recursion for PTIME.

To show the converse: program a Turing machine in this setting!

Conclusions ...

1. Computability is not just about Turing machines!!
2. Costing maps, as in complexity theory, is closely related to partiality ...
   ... *and* this has an elegant categorical expression.
3. Turing categories provide a formal unification of complexity and computability ... widening our view of computability.
4. The technique of costing maps using complexity classes can be extended to provide a semantics for safe initial data.
5. Abstract computabiliy – and Turing categories – indicate that the END OF STORY is still a long way off!