Computability in Timed Sets in Opetaa, Estonia

Robin Cockett
Joaquín Díaz-Boïls
Jonathan Gallagher
Pavel Hrubeš

University ofCalgary

February 4, 2013

Explicit versus implicit

EXPLICIT
(model)

↔

IMPLICIT
(axiomatic theory)

PROBLEM:

Need to know *what* one is modelling ...

Need to know *how* to axiomatize the phenomenon

Explicit versus implicit

EXPLICIT COMPUTABILITY
(particular model)

↕

IMPLICIT COMPUTABILITY
(axiomatic theory)

Explicit computability

- Turing machine computing (partial) functions
- Kleene's first model (natural numbers are codes for machines which act on numbers)
- Oracle computability (jump operators)
- Combinatory and λ-algebras
- Domain theory models.

Implicit computability

- Axiomatic/logic approaches to computability ...
- Combinatory logic and λ-calculus ...
- Turing categories

Turing categories = abstract computability

MANY non-standard models!!

... all models are Turing categories.

Explicit versus implicit

EXPLICIT COMPLEXITY
(particular model)

↕

IMPLICIT COMPLEXITY
(axiomatic theory)

Explicit complexity

- Time complexity: counting the ticks of a Turing/computing machine
- Space complexity: counting the storage required by a Turing/computing machine

Want these notions to be independent of the machine model ...

Are they?
Well not really!
e.g. Turing machine versus pointer models at low complexity

Implicit complexity

Why do it?

- Theoretical understanding of complexity ...
  - wide variety of different models
  - relationship between different models
  - correspondence between axiomatic features and complexity
- Type checking for complexity
  - real-time applications ...

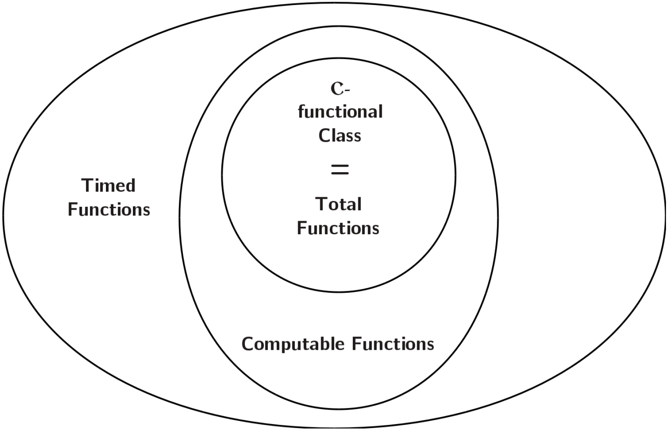
This talk looks at the explicit models complexity theorists themselves use!!!

but with categorical eyes!

Part of the program of abstract computability:

unifies complexity and computability.

Functional Complexity in a Timed Maps "Universe"



Functional Complexity in a Timed Maps "Universe"

A surprise connection between partiality and complexity

A categorical model/semantics of *basic* complexity theory

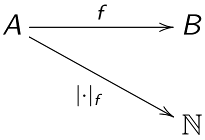
A construction that builds models of computability whose total maps are *precisely* the maps of a given functional complexity class:

$\mathcal{P}$-time, Log-space, and above ...

I.e. mimic what complexity theorists do ... BUT with categorical eyes.

The timing of a partial map as a primitive

Start with a notion of timing/costing a partial map:



$$f(x)\downarrow \quad \Leftrightarrow \quad |x|_f \downarrow$$

- A partial function f may have different timings
- Think of each timing as the cost (time/space/resource) of computing f by an algorithm

ASIDE: What is cost?

We shall assume cost is a natural number

BUT *the theory works more generally!*

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

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

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

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

Note: size monoids are orthogonal to commutative groups.

The Category of Timed Sets

TSet:

- Objects: Sets
- Maps: Timed partial functions
- Identity: The identity function with 0 cost
- Composition:

$$\left(A \xrightarrow{f} B,\ A \xrightarrow{|\cdot|_f} \mathbb{N}\right) \ \left(B \xrightarrow{g} C,\ B \xrightarrow{|\cdot|_g} \mathbb{N}\right) = \left(A \xrightarrow{fg} C,\ A \xrightarrow{|\cdot|_f + |f(\cdot)|_g} \mathbb{N}\right)$$

The Category of Timed Sets

Too restrictive ...

Two maps are equal only if their timing are *exactly* the same ...

Need to capture $\mathcal{O}(\_)$ the order of complexity ...

Complexity Orders

An **additive complexity order** $\mathcal{C}$ is a class of monotone functions $P : \mathbb{N} \longrightarrow \mathbb{N}$ such that $\mathcal{C}$ is:

- down-closed: $P \in \mathcal{C}$ and $Q \leq P$ then $Q \in \mathcal{C}$;
- closed to composition: if P, Q in $\mathcal{C}$ then $PQ \in \mathcal{C}$;
- additive: $0 \in \mathcal{C}$ and if $P, Q \in \mathcal{C}$ then $P + Q \in \mathcal{C}$.

Examples of complexity orders

Linear:

$$\mathcal{L} = \langle \lambda x.nx | n \in \mathbb{N} \rangle$$

Polynomial:

$$\mathcal{P} = \left\langle \lambda x.\sum_{i=1}^{n} a_i x^i | n \in \mathbb{N} \right\rangle$$

Where $\langle \_ \rangle$ denotes down-closure.

$\mathcal{C}$-ordering

Every complexity order $\mathcal{C}$ induces a preorder enrichment on the maps of TSet, $f \leq_{\mathcal{C}} g$:

- $g(x)\downarrow$ implies $f(x)\downarrow$ and $g(x) = f(x)$;
- there is a $P \in \mathcal{C}$ such that for all x, $|x|_f \leq P(|x|_g)$.

$\mathcal{C}$-equivalence

$\mathcal{C}$-equivalence is the congruence $f =_{\mathcal{C}} g$ if:

$$f \leq_{\mathcal{C}} g \qquad \text{and} \qquad g \leq_{\mathcal{C}} f$$

E.g. $f =_{\mathcal{L}} g$ if $|x|_f \leq m|x|_g$ and $|x|_g \leq n|x|_f$

Partiality: Restriction Categories

For each map $f : A \longrightarrow B$: a **restriction idempotent** $\overline{f} : A \longrightarrow A$ such that

[R.1]	$\overline{f}\, f = f$	**[R.3]**	$\overline{f}\, \overline{g} = \overline{\overline{f}\, g}$
[R.2]	$\overline{f}\, \overline{g} = \overline{g}\, \overline{f}$	**[R.4]**	$f\, \overline{h} = \overline{fh}\, f$

- A general framework for partiality [Cockett and Lack 2002]
  E.g. Sets and partial functions: $\overline{f}$ is domain of definition
- P-categories [Robinson and Rosolini 1988]
- Influential paper by Robert Di Paola and Alex Heller on "dominical categories" (1986) initiates abstract computability.

Totality in Restriction Category

Recall that a map in a restriction category is **total** in case

$$\overline{f} = 1.$$

Timed Sets and Restriction Structure

For TSet, the desired restriction is $\overline{(f, |\cdot|_f)} = (\overline{f}, |\cdot|_f)$.

However, this is not a restriction structure since **[R.1]** fails:

$$\begin{aligned}\overline{(f, |\cdot|_f)}\,(f, |\cdot|_f) &= (f, |\cdot|_f + |\cdot|_f) \\ &\neq (f, |\cdot|_f)\end{aligned}$$

The Restriction Category of Timed Sets

TSet may be quotiented by the congruence $=_{\mathcal{C}}$.

Proposition.

For any complexity order $\mathcal{C}$, $\mathsf{TSet}/\mathcal{C}$ is a restriction category where

$$\overline{(f, |\cdot|_f)} = (\bar{f}\,, |\cdot|_f).$$

Linking Complexity Order and Partiality

Every restriction category is partial order enriched by $f \leq g$:

$$\overline{f}\, g = f.$$

Lemma.

In TSet$/\mathcal{C}$,

$$f \leq g$$

if and only if

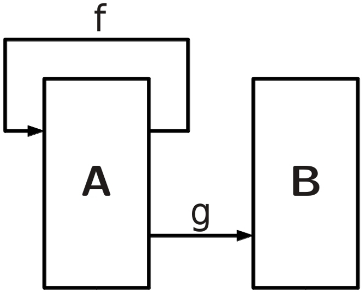
$$g \leq_{\mathcal{C}} f.$$

Iteration in a restriction category

$f \star g$:

$f^n g$ (for at most one n)

Intuitively
$g \sqcup fg \sqcup ffg \sqcup \cdots$



Iteration in a restriction category

```
Iterate(f,g)(x) =
    while (x in dom(f))
        x := f (x)
    g(x)
```

Iteration: one way to obtain computability ...

Disjoint joins

Proposition.

For every $\mathcal{C}$, TSet$/\mathcal{C}$ has disjoint joins.

What does that mean?

Disjoint joins

Disjointness, means that "domains" do not overlap

$$\overline{f}\,\overline{g} = \emptyset$$

The join of disjoint maps f, g is the join, $\sqcup$, with respect to $\leq$. Must also be "stable" with respect to composition:

$$h(f \sqcup g) = hf \sqcup hg$$

Disjoint joins and iteration

Need disjoint joins for iteration ...

$$f \star g = g \sqcup fg \sqcup ffg \sqcup \cdots = \bigsqcup_n f^n g$$

Also need

$$\bigsqcup_n f^n g =_{\mathcal{C}} \bigsqcup_n f'^n g'$$

whenever $f =_{\mathcal{C}} f'$ and $g =_{\mathcal{C}} g'$.

This requires the complexity order satisfy an extra **laxness** condition ...

Distributive Restriction Categories

Proposition. [Cockett and Lack 2007]

For a restriction category

$$\text{Distributivity} \Rightarrow \text{Extensiveness} \Rightarrow \text{Disjoint Joins}$$

The Distributive Restriction Category of Timed Sets

Proposition.

For every complexity order $\mathcal{C}$, TSet$/\mathcal{C}$ is a distributive restriction category.

TSet/$\mathcal{C}$ has a restriction terminal object

$\mathbf{1} = \{\star\}$ is the restriction terminal object.

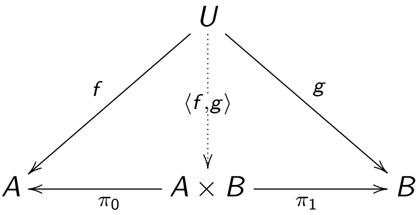
$$!_A : A \longrightarrow \mathbf{1}$$

is always defined and has zero cost. Thus, for any $f : A \longrightarrow \mathbf{1}$,

$$f = \overline{f}\,!_A$$

Restriction products

The **binary restriction product** of A, B is $A \times B$ with total projections π_0, π_1 and a unique pairing such that in



$\langle f, g\rangle\, \pi_0 = \overline{g}\, f$ and $\langle f, g\rangle\, \pi_1 = \overline{f}\, g$.

TSet/$\mathcal{C}$ has restriction products

$A \times B$ is as in Sets.

Projections, π, are always defined and have zero cost.

$$\langle (f, |\cdot|_f), (g, |\cdot|_g) \rangle := (\langle f, g \rangle, |\cdot|_{\langle f, g \rangle})$$

where

$$|x|_{\langle f, g \rangle} := |x|_f + |x|_g$$

TSet/$\mathcal{C}$ has an initial object

$\mathbf{0} = \emptyset$ is the initial object.

Note also that TSet/$\mathcal{C}$ has nowhere defined maps:

$$\emptyset := (\emptyset, \emptyset) : A \longrightarrow B$$

TSet/$\mathcal{C}$ has coproducts

$A + B$ is as in Sets.

Coprojections σ are always defined and have zero cost.

$$[(f, |\cdot|_f), (g, |\cdot|_g)] = ([f, g], |\cdot|_{[f,g]})$$

where

$$|\cdot|_{[f,g]} = [|\cdot|_f, |\cdot|_g]$$

TSet/$\mathcal{C}$ is distributive

The map

$$[A + \sigma_B, A + \sigma_C] : (A \times B) + (A \times C) \longrightarrow A \times (B + C)$$

is an isomorphism in Sets, and is zero cost.

Iteration

$$\frac{f : A \longrightarrow A \quad g : A \longrightarrow B \quad f, g \text{ disjoint}}{f \star g : A \longrightarrow B}$$

where [Conway 1971]:

W.1 $(fg) \star h = h \sqcup f((gf) \star (gh))$

W.2 $(f \sqcup g) \star h = (f \star g) \star (f \star h)$

W.3 $(f \star g)h = f \star (gh)$

W.4 $1 \times (f \star g) = (1 \times f) \star (1 \times g)$

W.5 $f \leq f'$, $g \leq g'$ then $f \star g \leq f' \star g'$

For example **W.1**:

$$\begin{aligned} & f \star h \\ & = h \sqcup f(f \star h) \\ & = h \sqcup f(h \sqcup f(f \star h)) \\ & = h \sqcup fh \sqcup f^2(f \star h) \\ & = \cdots \end{aligned}$$

Iteration in TSet/$\mathcal{C}$

Definition.

A complexity order $\mathcal{C}$ is **lax** if it is generated by functions P,

$$P(m) + P(n) \leq P(m + n)$$

Proposition.

If $\mathcal{C}$ is lax, then TSet/$\mathcal{C}$ has iteration

Both $\mathcal{L}$ and $\mathcal{P}$ are lax ..

Iteration in $\mathsf{TSet}/\mathcal{C}$

Given disjoint timed maps $f : A \longrightarrow A$, $g : A \longrightarrow B$,

$$f \star g(x) := \begin{cases} g(f^n(x)) & \exists n. f^n \in \overline{g} \\ \uparrow & \text{else} \end{cases}$$

where the cost is

$$|x|_{f \star g} := \begin{cases} \sum_{i=0}^{n-1} |f^i(x)|_f + |f^n(x)|_g & \exists n. f^n \in \overline{g} \\ \uparrow & \text{else} \end{cases}$$

Structural Recap

The basic structural ingredients for building a simple model of complexity:

- Timed functions
- $\mathcal{C}$-equivalence
- Distributivity
- Iteration

Additional Structure

Discreteness: the map $\Delta : A \longrightarrow A \times A$ has a partial inverse:

$$\Delta^{-1}(x, y) = \begin{cases} x & x = y \\ \uparrow & \text{else} \end{cases}$$

Ranges: restriction idempotents that act on the codomain; provides the image.

Finite Joins: If $\overline{f}\, g = \overline{g}\, f$, then the stable join with respect to $\leq$ of f, g exists.

Total maps

Problems:

The total maps are zero cost maps

$\overline{(f, |\cdot|_f)} = 1$ if, in particular, there is a P such that

$$|\cdot|_f \leq P(0) = 0.$$

However, "running time" should be a function of input size.

Restriction Idempotents in TSet/$\mathcal{C}$

A restriction idempotent is a timed partial identity

Restriction idempotents can be thought of as measuring the size of the input.

Restriction Idempotents Splitting of $\mathsf{TSet}_{/\mathcal{C}}$

An object in Split($\mathsf{TSet}_{\mathcal{C}}$) is a sized set

$$e = (A, |\cdot|_e)$$

A map $f : e \longrightarrow e'$ is a timed map such that $efe' =_{\mathcal{C}} f$:

$$|x|_e + |x|_f + |f(x)|_{e'} \leq P(|x|_f)$$

Intuitively a function cannot be "faster" than the time required to read its input and produce its output!!

Linking Complexity and Totality

Recall, in a restriction category, f is **total** if $\overline{f} = 1$.

In the restriction idempotent splitting:

$$f : e \longrightarrow e' \text{ is total iff } e = \overline{f}$$

In $\mathrm{Split}(\mathrm{TSet}_{\mathcal{C}})$, what doe this mean?

$$P(|x|_e) \geq |x|_f$$

f is $\mathcal{C}$-bounded by the size of its input.
i.e. total maps are exactly the "$\mathcal{C}$-timed" maps!!!

The structure in $\mathsf{Split}(\mathsf{TSet}_{\mathcal{C}})$

All the structure lifts to the idempotent splitting.

Theorem.
$\mathsf{Split}(\mathsf{TSet}_{\mathcal{C}})$ is a distributive restriction category with iteration where the total maps are precisely those with $\mathcal{C}$-cost.

Sizes are non-zero ..

Elements with zero size have no impact on complexity!

How do we ensure all sizes are non-zero?

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

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

Lemma

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

Computability

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

- Not computable
- Their $\mathcal{P}$-timing are arbitrarily assigned.

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

E.g. by a Turing machine with the standard timing.

Shall show how this gives a Turing category whose total maps are precisely PTIME maps.

Powerful and program objects

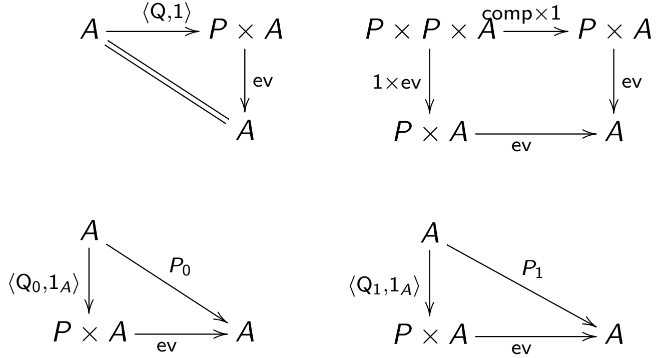
A is a **powerful object** in case there are total maps $s_\times : A \times A \longrightarrow A$ and partial maps $P_0, P_1 : A \longrightarrow A$ such that $s_x \langle P_0, P_1 \rangle = 1_{A \times A}$.

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:

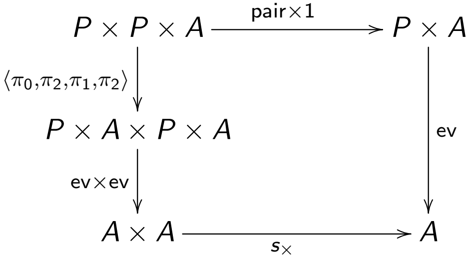
$$
\begin{array}{ll}
s_\times(b : bs, b' : bs') = 1 : b : 1 : b' : s_\times(bs, bs') & P_0(1 : b : \_ : \_ : rs) = b : P_0(rs) \\
s_\times([], b' : bs') = 0 : 0 : 1 : b' : s_\times([], bs') & P_0(0 : 0 : \_ : \_ : rs) = [] \\
s_\times(b : bs, []) = 1 : b : 0 : 0 : s_\times(bs, []) & P_1(\_ : \_ : 1 : b' : \_ : rs') = b' : P_1(rs') \\
 & P_1(\_ : \_ : 0 : 0 : rs') = []
\end{array}
$$

Powerful and program objects

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



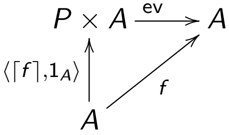
Powerful and program objects



We shall say that P is a **machine** program object in case $\mathsf{ev} = \mathsf{step} \star \mathsf{halt}$ where $\overline{\mathsf{step}} \vee \overline{\mathsf{halt}} = 1_{P \times A}$. In other words ev is a trace of a machine transition which is *total*.

Powerful and program objects

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



If X and Y are (particular) retracts of A then a map $h : X \longrightarrow Y$ is P-**programmable** if the map $A \xrightarrow{r_X} X \xrightarrow{h} Y \xrightarrow{s_Y} A$ is programmable.

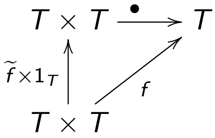
Theorem

If A is an inhabited powerful object in $\mathbb{X}$ and P is an A-program object, then the subcategory of P-programmable maps, $\text{Prog}_P(\mathbb{X})$*, on powers of A forms a cartesian restriction subcategory.*

Turing structure

An object T, in a cartesian restriction category, is a **Turing object** in case:

- Every object in the category is a retract of T.
- There is an **application map**, also called a **Turing morphism**, $\bullet : T \times T \longrightarrow T$ such that for every (partial) map $f : T \times T \longrightarrow T$ there is a *total map* $\widetilde{f} : A \longrightarrow T$ such that:



A cartesian category with a Turing object is a **Turing category**: these provide a unifying formulation of abstract computability.

When does an A-program object P make A a Turing object?

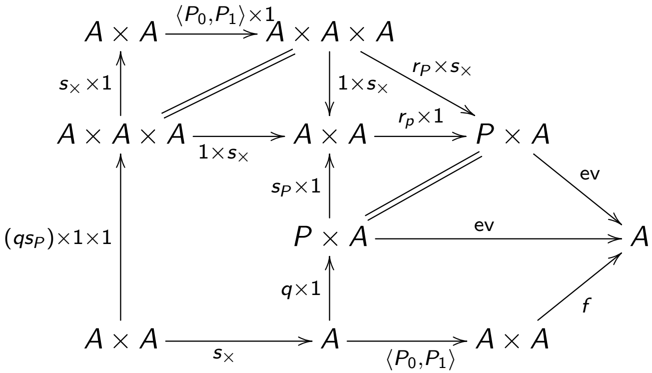
Turing structure

Theorem

If $\mathbb{X}$ *is a cartesian restriction category with an inhabited powerful object* A *and an* A*-programming object* P *such that* P *is a retract of* A *and* comp, pair, ev, Q_0, Q_1, *and* Q *are all* P*-programmable then* $\mathsf{Prog}_P(\mathbb{X})$ *is a Turing category.*

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

$\bullet := (\langle P_0, p_1 \rangle \times 1)(r_P \times s_\times)\mathrm{ev}.$

Turing Categories and Total Maps

Theorem.

The maps that are computable by a Turing machine within $\mathcal{P}$-time in Split($\mathsf{TSet}_{\mathcal{P}}$) form a Turing category, $\mathbb{T}_{\mathcal{P}}$, whose Total maps are the $\mathcal{P}$-time maps.

Theorem.

The maps that are computable on a Transducer within Log-space in Split($\mathsf{TSet}_{\mathcal{L}}$) form a Turing category, $\mathbb{T}_{\mathsf{Lg}}$, whose Total maps are the Log-space maps.

Turing Categories and Total Maps

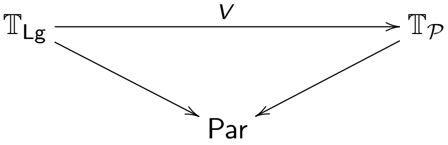
Proof.

Turing machines can be composed and paired in $\mathcal{P}$-time in the size of their inputs.

For evaluation use the fact that a universal Turing machine can simulate any Turing machine with just a polynomial overhead. $\square$

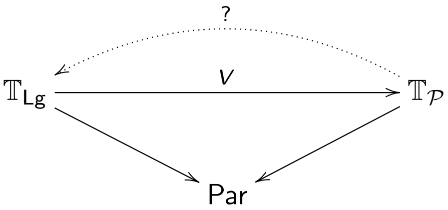
From Log-space to $\mathcal{P}$-time

There is a restriction preserving functor over Par between the above Turing categories:



Recall that if T runs in Space(S) then it runs in at most Time(2^S).

And back again?



V is an isomorphism if and only if $\mathcal{P}$-time = Log-space.

If $\mathcal{P}$-time and Log-space are equal, then for all T: Time(T) is Space(Log(T)).

Open complexity problem ....

In Conclusion

Ideas in complexity can now be translated into categorical notions.

Open complexity problems have been re-expressed into categorical questions.

There are Turing categories whose total maps are precisely those of functional complexity classes.

Abstract computability unifies complexity and computability ....

In Conclusion

For more:

Robin Cockett, Joaquin Diaz-Bols, Jonathan Gallagher, Pavel Hrubes

"Timed Sets, Functional Complexity, and Computability"

Electronic Notes in Theoretical Computer Science
Volume 286, 24 September 2012, Pages 117–137.