Towards a fine structure of computabilities

Vers une structure fine des calculabilités

Fabien Givors

joint work with Grégory Lafitte

Université Montpellier II - CNRS, LIRMM

CMF13' - Caen

December 18th

Motivations

Computational models for fragments of computability

A computability framework for subrecursion

A hierarchy of computabilities, from primitive recursion to admissible recursion, and above

1. classical computability

Church, Kleene, Rosser, Turing

Computable functions

Basic operations

$\mathbf{o} : x \mapsto 0$

$\mathbf{s} : x \mapsto x + 1$

$\mathbf{cond} : x, y, a, b \mapsto \begin{cases} a & \text{if } x = y \\ b & \text{otherwise} \end{cases}$

Operations can be composed

$\mathbf{s} \circ \mathbf{o} = x \mapsto 1$

Complex operations

μ: minimisation

$\mathbf{rec}_{\mathrm{p}}$: primitive recursion

Computation tree

Not necessarily finite

Computable functions

Basic operations

$\mathbf{s} : x \mapsto x + 1$

$\mathbf{cond} : x, y, a, b \mapsto \begin{cases} a & \text{if } x = y \\ b & \text{otherwise} \end{cases}$

Operations can be composed

$\mathbf{s} \circ \mathbf{0} = x \mapsto 1$

Complex operations

μ: minimisation

primitive recursion

Computation tree

Not necessarily finite

Computation Tree / Derivation Tree

$(x \mapsto \mathbf{cond}(\mathbf{s}(x), 3, 9, 7))\ (3)$

$\mathbf{cond}(\mathbf{s}(3), 3, 9, 7)$

$\mathbf{cond}(4, 3, 9, 7)$

7

The more complex the machine, the more complex the tree.

Notations

- Enumeration of all the recursive functions, indexed by the natural integers:

$$(\varphi_e)_{e \in \omega}$$

- Convergence of a function:

$$\varphi_{26}(13) \downarrow = 7$$

- Divergence of a function:

$$\varphi_{34}(4) \uparrow$$

- Equivalence of functions:

$$\varphi_{e_1}(x_1) \simeq \varphi_{e_2}(x_2)$$

if both computations diverge or converge to the same value.

Universality and total functions

- Recursive universal function

$$\exists u, \varphi_u : (\langle e, x \rangle) \mapsto \varphi_e(x)$$

- Functions cannot all be total

$$f : x \mapsto \mathbf{s} \circ \varphi_u \left(\langle x, x \rangle \right)$$

$$\exists e, f = \varphi_e$$

$$\begin{aligned} f(e) &\cong \mathbf{s}(\varphi_u(\langle e, e \rangle)) \\ &\cong \varphi_e(e) + 1 \\ &\cong f(e) + 1 \end{aligned}$$

Canonical form and interpretation

Kleene's Normal Form

There exist an elementary function F and a recursive primitive predicate T such that:

$$\forall e, x,\ \varphi_u(\langle e, x\rangle) \cong \varphi_e(x) \cong F(\mu y.T(e, x, y))$$

- Only μ operator
- Function index
- Computation tree
- Input
- Checker

Halt

Computation tree

Finite?

Bounded?

s_n^m and Fixed Point

- s_n^m:
  There exists a recursive function s_n^m such that $\forall m, n, e$,

$$\varphi_e(\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle) \cong \varphi_{s_n^m(e,x_1,\ldots,x_n)}(\langle y_1, \ldots, y_m \rangle)$$

- Fixed point:
  For each total recursive function f we can recursively compute an n such that:

$$\forall x, \varphi_n(x) \cong \varphi_{f(n)}$$

s_n^m and Fixed Point

- s_n^m:
  There exists a recursive function s_n^m such that $\forall m, n, e,$

$$\varphi_e(\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle) \cong \varphi_{s_n^m(e, x_1, \ldots, x_n)}(\langle y_1, \ldots, y_m \rangle)$$

- Fixed point:
  For each total recursive function f we can recursively compute an n such that:

$$\forall x, \varphi_n(x) \cong \varphi_{f(n)}$$

Example

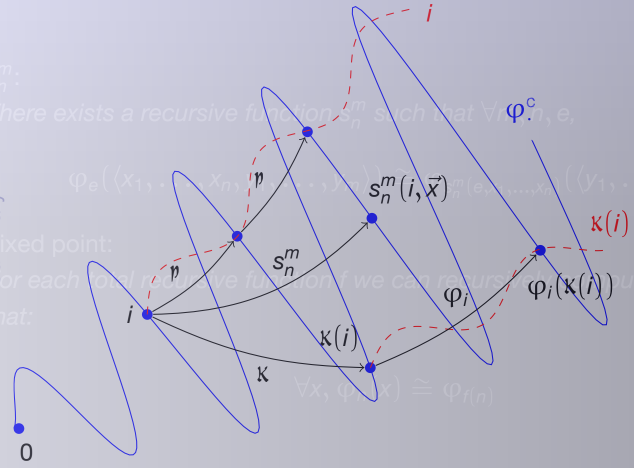
Let f be a function such that $\varphi_{f(n)} : x \mapsto \varphi_n(x) + 1$. Then there is an n verifying:

$$\forall x, \varphi_n(x) \cong \varphi_{f(n)}(x) \cong \varphi_n(x) + 1$$

which is an index for the nowhere defined function.

s_n^m and Fixed Point

- s_n^m:
  There exists a recursive function s_n^m such that $\forall m, n, e$,
  $$\varphi_e(\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle) \simeq \varphi_{s_n^m(e, x_1, \ldots, x_n)}(\langle y_1, \ldots, y_m \rangle)$$
- Fixed point:
  For each total recursive function f we can recursively compute an n such that:
  $$\forall x, \varphi_n(x) \simeq \varphi_{f(n)}$$



2. from total functions to partial computabilities

An interesting trade-off

Classes of total functions

For a class $\mathcal{C}$ of total recursive functions, closed under composition:

- Universal function not in $\mathcal{C}$
- No unbounded search
- Limited function growth (e.g. Ackermann function not in $\mathcal{P}$)
- Implies limited power

Classes of total functions

For a class $\mathbb{C}$ of total recursive functions, closed under composition:

- Universal function not in $\mathbb{C}$
- No unbounded search
- Limited function growth (e.g. Ackermann function not in $\mathbb{P}$)
- Implies limited power

How to deal with these limitations?

Primitive recursive coding schema

For a class C *with an enumeration* $\varphi^{\mathrm{C}}_{\cdot}$*:*

Indices

Toolbox for indices:

Compute new indices for a function

Compute index of composition

Primitive recursive coding schema

For a class $\mathcal{C}$ *with an enumeration* $\phi^{\mathcal{C}}_{\cdot}$*:*

Toolbox for indices:

Indices

Compute new indices for a function

Compute index of composition

Requirements:

padding function: $\mathfrak{p}$

$\forall e, (\mathfrak{p}(e) > e) \wedge (\phi^{\mathcal{C}}_{\mathfrak{p}(e)} = \phi^{\mathcal{C}}_{e})$

composition function: $\mathfrak{c}$

$\forall e_1, e_2, \phi^{\mathcal{C}}_{\mathfrak{c}(e_1,e_2)} = \phi^{\mathcal{C}}_{e_1} \circ \phi^{\mathcal{C}}_{e_2}$

Def: Coding schema

Primitive recursive coding schema

For a class $\mathbb{C}$ *with an enumeration* $\phi^{\mathbb{C}}_{\cdot}$:

Indices

Toolbox for indices:

Compute new indices for a function

Compute index of composition

Simulation

Toolbox for simulation:

Check the validity of a tree

Bound the tree of a given function

Def: Coding schema

Requirements:

padding function: $\mathfrak{p}$

$\forall e, (\mathfrak{p}(e) > e) \wedge (\phi^{\mathbb{C}}_{\mathfrak{p}(e)} = \phi^{\mathbb{C}}_{e})$

composition function: $\mathfrak{c}$

$\forall e_1, e_2, \phi^{\mathbb{C}}_{\mathfrak{c}(e_1,e_2)} = \phi^{\mathbb{C}}_{e_1} \circ \phi^{\mathbb{C}}_{e_2}$

Primitive recursive coding schema

For a class c *with an enumeration* $\phi^{\mathsf{c}}_{\cdot}$*:*

Indices

Toolbox for indices:

Compute new indices for a function

Compute index of composition

Simulation

Toolbox for simulation:

Check the validity of a tree

Bound the tree of a given function

Def: Coding schema

Requirements:

padding function: $\mathfrak{p}$

$\forall e, (\mathfrak{p}(e) > e) \wedge (\phi^{\mathsf{c}}_{\mathfrak{p}(e)} = \phi^{\mathsf{c}}_e)$

composition function: $\mathfrak{c}$

$\forall e_1, e_2, \phi^{\mathsf{c}}_{\mathfrak{c}(e_1,e_2)} = \phi^{\mathsf{c}}_{e_1} \circ \phi^{\mathsf{c}}_{e_2}$

step-by-step simulation: $\mathrm{sim}_{\mathsf{c}}$

$\forall x, e, \exists n, \phi^{\mathsf{c}}_e(x) = \mathrm{sim}_{\mathsf{c}}(e, x, n)$

cost function: $\mathrm{use}_{\mathsf{c}}$

$\forall x, e, \phi^{\mathsf{c}}_e(x) = \mathrm{sim}_{\mathsf{c}}(e, x, \phi^{\mathsf{c}}_{\mathrm{use}_{\mathsf{c}}(e)}(x))$

Fundamental classes

Def: Fundamental class

Basis functions

Contains all the primitive recursive functions

Coding functions

Primitive recursive coding schema: $\mathbf{rec}_{\mathrm{p}}$

Pairing functions: $\langle\cdot,\cdot\rangle, \pi^2_1, \pi^2_2 \in \mathrm{c}$,

Projection functions: $\forall n_1, n_2, \pi^2_i(\langle n_1, n_2\rangle) = n_i$

Closure

Stable by composition: $\forall f, g \in \mathrm{c}, f \circ g \in \mathrm{c}$.

Stable by primitive recursion: $\forall g, h \in \mathrm{c}, \mathbf{rec}_{\mathrm{p}}(g, h) \in \mathrm{c}$

Enumeration

Tied to an enumeration $\phi^{\mathrm{c}}_{\cdot}$ (recursive) which is not in the class

Classes defined using recursion schemata

Definition

Primitive recursive class: p

Smallest fundamental class (stable by primitive recursion: $\mathbf{rec}_{\mathrm{p}}$)

Classes defined using recursion schemata

Definition

Primitive recursive class: p

Smallest fundamental class (stable by primitive recursion: $\mathbf{rec}_{\mathrm{p}}$)

Primitive recursion

For $g, h \in \mathrm{p}$,

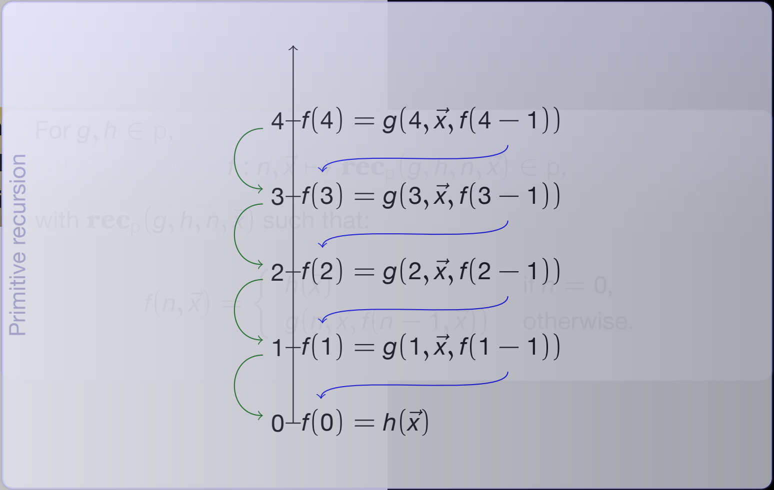
$$f : n, \vec{x} \mapsto \mathbf{rec}_{\mathrm{p}}(g, h, n, \vec{x}) \in \mathrm{p},$$

with $\mathbf{rec}_{\mathrm{p}}(g, h, n, \vec{x})$ such that:

$$f(n, \vec{x}) = \begin{cases} h(\vec{x}) & \text{if } n = 0, \\ g(n, \vec{x}, f(n-1, \vec{x})) & \text{otherwise.} \end{cases}$$

Classes defined using recursion schemata

Definition **Prim**…
Smal…
primi…



Classes defined using recursion schemata

Definition

Primitive recursive class: p
Smallest fundamental class (stable by primitive recursion: $\mathbf{rec}_{\mathrm{p}}$)

Definition

α-recursive classes: c_{α}
Smallest fundamental class stable by α recursion: $\mathbf{rec}_{\alpha}$

Classes defined using recursion schemata

Definition

Primitive recursive class: p
Smallest fundamental class stable by primitive recursion: $\mathbf{rec}_p$)

Definition

α-recursive classes: c_α
Smallest fundamental class stable by α recursion: $\mathbf{rec}_\alpha$

α-recursion

For $g, h \in c_\alpha$,

$$f : n, \vec{x} \mapsto \mathbf{rec}_{\alpha,\lhd}(g, h, n, \vec{x}) \in c_\alpha$$

with $\mathbf{rec}_{\alpha,\lhd}(g, h, n, \vec{x})$ such that:

$$f(n, \vec{x}) = \begin{cases} g(n, \vec{x}, f(\theta(n, \vec{x}), \vec{x})) & \text{if } \overline{0} \lhd n \text{ and } \theta(n, \vec{x}) \lhd n, \\ h(n, \vec{x}) & \text{otherwise,} \end{cases}$$

where $\overline{\beta}$ stands for the ordinal notation (in $\lhd$) for β.

Classes defined using recursion schemata

Definition

Prim

Smal

primi

Definition

α-re

Smal

α rec

α-recursion

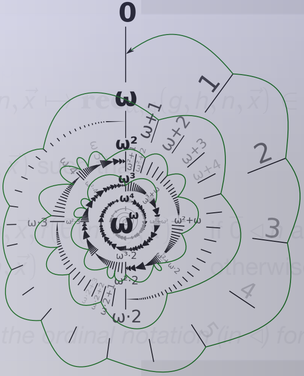
For $g, h \in c_\alpha$,

$$f : n, \vec{x} \mapsto \mathbf{rec}_\alpha(g, h, n, \vec{x}) \in c_\alpha$$

with $\mathbf{rec}_{\alpha,\lhd}(g, h, n, \vec{x})$ such that

$$f(n, \vec{x}) = \begin{cases} g(n, \vec{x}, f(\ldots)) & \text{if } 0 \lhd n \text{ and } \theta(n, \vec{x}) \lhd n, \\ h(n, \vec{x}) & \text{otherwise,} \end{cases}$$

where $\overline{\beta}$ stands for the ordinal notation (in $\lhd$) for β.



Classes defined using recursion schemata

Definition

Primitive recursive class: p

Smallest fundamental class (stable by primitive recursion: $\mathbf{rec}_p$)

Rathjen: Functions provably total by a theory of ordinal analysis α are exactly α-recursive functions.

Definition

α-recursive classes: c_α

Smallest fundamental class stable by α recursion: $\mathbf{rec}_\alpha$

How far did we get?

Thm s_n^m **for fundamental classes**

Note: Primitive recursion is needed in order to obtain an homogeneous s.

How far did we get?

Thm

s_n^m for fundamental classes

Note: Primitive recursion is needed in order to obtain an homogeneous s.

Thm

Still no recursion theorem

It is actually impossible.

How far did we get?

Thm s_n^m ... fundamental classes

Note: Primitive recursion is needed in order to obtain an homogeneous s.

Thm **Still no recursion theorem**

It is actually impossible.

Kleene and total functions

Recall our previous example:

Let f be a function such that $\varphi_{f(n)} : x \mapsto \varphi_n(x) + 1$. Then there is an n verifying:

$$\forall x, \varphi_n(x) \simeq \varphi_{f(n)}(x) \simeq \varphi_n(x) + 1$$

which is an index of the nowhere defined function.

Such a function cannot be total.

How far did we get?

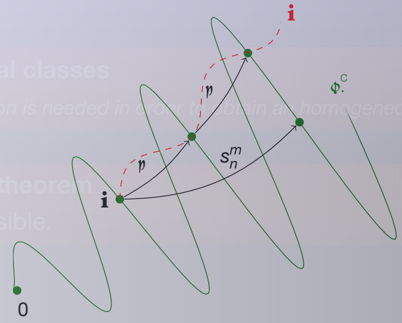
Thm **s^m_n for fundamental classes**

Note: Primitive recursion is needed in order to obtain an homogeneous s.

Thm **Still no recursion theorem**

It is actually impossible.

Summary



Rogers' Isomorphism Theorem

Ensures that our results do not depend on our choice of an enumeration

Rogers' Isomorphism Theorem

Ensures that our results do not depend on our choice of an enumeration

Go from an acceptable enumeration to another

Rogers' Isomorphism Theorem

Ensures that our results do not depend on our choice of an enumeration

Go from an acceptable enumeration to another

Thm

Myhill's isomorphism for fundamental classes

For A and B two sets of integers, f 1-1 from A to B and g 1-1 from B to A, we can build h an isomorphism between A and B.

Rogers' Isomorphism Theorem

Ensures that our results do not depend on our choice of an enumeration

Go from an acceptable enumeration to another

Thm **Myhill's isomorphism for fundamental classes**
For A and B two sets of integers, f 1-1 from A to B and g 1-1 from B to A, we can build h an isomorphism between A and B.

Thm **Rogers' isomorphism for fundamental classes**
Any acceptable enumeration is isomorphic to the canonical one.

Simulation, halt and domination

Primitive recursive case

Simulation, halt and domination

Primitive recursive case

Ackermann function

Definition:

$$A : m, n \mapsto \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{otherwise} \end{cases}$$

Unary version:

$$\text{Ack} : n \mapsto A(n, n)$$

Simulation, halt and domination

Primitive recursive case

Ackermann function properties

Grows faster than any primitive recursive function

Simulation, halt and domination

Primitive recursive case

Ackermann function properties

Grows faster than any primitive recursive function

Enables us to bound the size of their tree

Simulation, halt and domination

Primitive recursive case

Ackermann function properties

Grows faster than any primitive recursive function

Enables us to bound the size of their tree

Universal simulation with $\mathrm{sim_p}$ and $\mathrm{use_p}$:

$$x \mapsto \mathrm{sim_p}(e, x, \mathrm{Ack}(f(\mathrm{use_p}(e), x)))$$

for some primitive recursive f

Simulation, halt and domination

General case for fundamental classes

Primitive recursive case

Ackermann function properties

Grows faster than any primitive recursive function

Enables us to bound the size of their tree

Universal simulation with sim_p and use_p:

$$x \longmapsto \mathrm{sim}_p(e, x, \mathrm{Ack}(f(\mathrm{use}_p(e), x)))$$

for some primitive recursive f

Simulation, halt and domination

General case for fundamental classes

Primitive recursive case

Busy Beaver function

Definition:

For an enumerable class $\mathcal{C}$ with $\text{sim}_{\mathcal{C}}$ and $\text{use}_{\mathcal{C}}$ functions:

$$\mathbb{BB}^{\Phi}_{\mathcal{C}} = x \mapsto max \left\{ \phi^{\mathcal{C}}_{\text{use}_{\mathcal{C}}(i)}(0) : i \leqslant x \right\}$$

Ackermann function properties

Grows faster than any primitive recursive

Enables us to bound the size of their tree

Universal simulation with sim_p and use_p:

$$x \mapsto \text{sim}_p(e, x, \text{Ack}(f(\text{use}_p(e), x)))$$

for some primitive recursive f

Simulation, halt and domination

General case for fundamental classes

Primitive recursive case

Thm

Busy Beaver properties

Grows faster than any c-fundamental function

Enables us to bound the size of their tree

Universal simulation with sim_c and use_c:

$$x \longmapsto \mathrm{sim}_c(e, x, \mathbb{BB}_c^{\phi}(s_1^1(e, x)))$$

Ackermann function properties

Grows faster than any primitive recursive function

Enables us to bound the size of their tree

Universal simulation with sim_p and use_p:

$$x \longmapsto \mathrm{sim}_p(e, x, \mathrm{Ack}(f(\mathrm{use}_p(e), x)))$$

for some primitive recursive f

A recursive jump for fundamental classes

$\mathbb{BB}_{\mathbb{C}}^{\Phi}$ allows us to totally compute any function in $\mathbb{C}$.

Similar to the classical halting problem.

A recursive jump for fundamental classes

$\mathbb{BB}^{\phi}_{\mathrm{C}}$ allows us to totally compute any function in C.

Similar to the classical halting problem.

Def: Jump

Jump of C: $© = \mathrm{C}[\mathbb{BB}^{\phi}_{\mathrm{C}}]$

Still a fundamental class

$\phi^{©}_{.}$ has a universal function for C.

A recursive jump for fundamental classes

$\mathbb{BB}_{\mathbb{C}}^{\phi}$ allows us to totally compute any function in $\mathbb{C}$.

Similar to the classical halting problem.

Def: Jump

Jump of $\mathbb{C}$: $© = \mathbb{C}[\mathbb{BB}_{\mathbb{C}}^{\phi}]$

Still a fundamental class

$\phi_{\cdot}^{©}$ has a universal function for $\mathbb{C}$.

Relativisations of Kleene's $\mathcal{O}$ and Hyperarithmetic sets

A notion of $\mathbb{C}$-recursive orders (ordinals)

A fine hierarchy of $\mathbb{C}$-degrees

3. c-enumerability and c-recursivities

Complexity of sets

Enumerability and repetitions

Aim: Capture a class complexity through its enumerable sets

Enumerability and repetitions

Aim

Capture a class complexity through its enumerable sets

Enumerable sets and p

Every enumerable set is enumerable by a primitive recursive function.

Enumerate $\mathbb{W}_e$

Simulate φ_e using bounded μ in Kleene's Normal Form:

$$\varphi_{e,s}(x) \simeq F(\mu y \leqslant s . T(e,x,y)) \simeq \mathrm{sim}_T(e,x,s)$$

Enumeration with repetitions:

$$\varphi_e(0), \varphi_e(0), \ldots, \varphi_e(0), \varphi_e(1), \varphi_e(2), \varphi_e(2), \ldots$$

Enumerability and repetitions

Aim

Capture a class complexity through its enumerable sets

Anoyance

Classical definition not interesting

Enumerable sets are c-enumerable.

Enumerability and repetitions

Aim: Capture a class complexity through its enumerable sets

Anoyance: Classical definition not interesting

Enumerable sets are c-enumerable.

Classical characterisations

Domain of a partial function

Range of a partial function

Enumerability and repetitions

Aim

Capture a class complexity through its enumerable sets

Anoyance

Classical definition not interesting

Enumerable sets are c-enumerable.

Classical characterisations

Domain of a partial function

Range of a partial function

Range of a total function

Enumerability and repetitions

Aim

Capture a class complexity through its enumerable sets

Anoyance

Classical definition not interesting

Enumerable sets are c-enumerable.

Classical characterisations

Domain of a partial function

Range of a partial function

Range of a total function

Range of a 1-1 function
(partial for finite sets)

Enumerability and repetitions

Aim
Capture a class complexity through its enumerable sets

Anoyance
Classical definition not interesting

Enumerable sets are c-enumerable.

Classical characterisations

Domain of a partial function

Range of a partial function

Range of a total function

Range of a 1-1 function
(partial for finite sets)

Produce a new element on each iteration

Enumerability and repetitions

Aim: Capture a class complexity through its enumerable sets

Anoyance: Classical definition not interesting

Enumerable sets are c-enumerable.

Solution: A set is c-enumerable if:

it is finite

or it is the range of a 1-1 $f \in \mathrm{c}$.

Enumeration: $\left(\mathrm{w}_e^{\mathrm{c}}\right)_{e\in\omega}$

Classical characterisations

Domain of a partial function

Range of a partial function

Range of a total function

Range of a 1-1 function
(partial for finite sets)

Produce a new element on each iteration

Enumerability and repetitions

Aim
Capture a class complexity through its enumerable sets

Anoyance
Classical definition not interesting

Enumerable sets are c-enumerable.

Solution
A set is c-enumerable if:

it is finite

or it is the range of a 1-1 $f \in \mathrm{c}$.

Enumeration: $(\mathrm{w}_e^{\mathrm{c}})_{e \in \omega}$

Classical characterisations

Domain of a partial function

Range of a partial function

Range of a total function

Range of a 1-1 function
(partial for finite sets)

Produce a new element on each iteration

Tips & Tricks

How do we know if a function is 1-1?

We do not.

Check 1-1-ness for each new value

If not, the 1-1 prefix defines a finite set

c-recursivities

Extend the notion of c-enumerability to a notion of c-recursivity.

$\mathbb{C}$-recursivities

Extend the notion of $\mathbb{C}$-enumerability to a notion of $\mathbb{C}$-recursivity.

Classical characterisations

A set E is recursive if:

Its characteristic function χ_E is recursive

E and $\overline{E}$ are enumerable

E can be enumerated increasingly

c-recursivities

Extend the notion of c-enumerability to a notion of c-recursivity.

Classical characterisations

A set E is recursive if:

Its characteristic function χ_E is recursive

E and $\overline{E}$ are enumerable

E can be enumerated increasingly

χ-c-recursivity Def

c-fundamental characteristic function

weak-c-recursivity Def

c-enumerable and co-c-enumerable

strong-c-enumerability Def

c-enumerable increasingly

strong-c-recursivity Def

Strongly c-enumerable and co-strongly c-enumerable

c-recursivities

Extend the notion of c-enumerability to a notion of c-recursivity.

Classical characterisations

A set E is recursive if:

Its characteristic function χ_E is recursive

E and $\overline{E}$ are enumerable

E can be enumerated increasingly

χ-c-recursivity (Def)

c-fundamental characteristic function

weak-c-recursivity (Def)

c-enumerable and co-c-enumerable

strong-c-enumerability (Def)

c-enumerable increasingly

strong-c-recursivity (Def)

Strongly c-enumerable and co-strongly c-enumerable

These notions are all different, and all compatible with the classical one (Thm)

Noticeable sets and recursive properties

Def

Diagonal set

$\kappa_{c}^{\phi} = \{e : \phi_{e}^{c}(e) > 0\}$

Noticeable sets and recursive properties

Def — Diagonal set

$\kappa_{\mathrm{c}}^{\phi} = \{e : \phi_e^{\mathrm{c}}(e) > 0\}$

Def — Unary Ackermann range

$A = \mathrm{range}(\mathrm{Ack})$

Def — Busy Beaver range

$B_{\mathrm{p}} = \mathrm{range}(\mathrm{BB}_{\mathrm{p}}^{\phi})$

$B_{\mathrm{c}} = \mathrm{range}(\mathrm{BB}_{\mathrm{c}}^{\phi})$

Noticeable sets and recursive properties

Def — Diagonal set

$\kappa_{\mathrm{c}}^{\phi} = \{e : \phi_e^{\mathrm{c}}(e) > 0\}$

Def — Unary Ackermann range

$A = \mathrm{range}(\mathrm{Ack})$

Def — Busy Beaver range

$B_{\mathrm{p}} = \mathrm{range}(\mathrm{BB}_{\mathrm{p}}^{\phi})$

$B_{\mathrm{c}} = \mathrm{range}(\mathrm{BB}_{\mathrm{c}}^{\phi})$

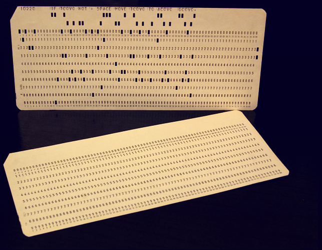
Thm

	r.e.	co-*r.e.*	p-e	s-p-e	co-p-e	co-s-p-e	w-p-rec	s-p-rec	χ-p-rec
κ	✓	✗	✓	✗	✗	✗	✗	✗	✗
$\kappa_{\mathrm{p}}^{\phi}$	✓	✓	✓	✗	✓	✗	✓	✗	✗
A	✓	✓	✗	✗	✓	✓	✗	✗	✓
B_{p}	✓	✓	✗	✗	✓	✓	✗	✗	✓

	r.e.	co-*r.e.*	c-e	s-c-e	co-c-e	co-s-c-e	w-c-rec	s-c-rec	χ-c-rec
κ	✓	✗	✓	✗	✗	✗	✗	✗	✗
$\kappa_{\mathrm{c}}^{\phi}$	✓	✓	✓	✗	✓	✗	✓	✗	✗
B_{c}	✓	✓	✗	✗	✓	✓	✗	✗	✓

4. subcomputabilities

Computability, with holes



c-partial functions

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

c-partial functions

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def

$\forall e, \mathcal{G}_e$ is the graph induced by $\mathtt{w}_e^{\mathrm{c}}$

Enumeration: $(\varphi_e^{\mathrm{c}})_{e \in \omega}$

c-partial functions

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def

$\forall e, \mathcal{G}_e$ is the graph induced by $\mathtt{w}_e^{\mathrm{c}}$

Enumeration: $(\varphi_e^{\mathrm{c}})_{e\in\omega}$

Thm

Fundamental functions are c-partials

c-partial functions

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def

$\forall e, \mathcal{G}_e$ is the graph induced by $\mathtt{w}_e^{\mathrm{c}}$

Enumeration: $(\varphi_e^{\mathrm{c}})_{e \in \omega}$

Thm

Fundamental functions are c-partials

Thm

Growth speed is dominated by fundamental functions

c-partial functions

Unusual closure

Thm

Not stable by composition

No s_n^m theorem

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

$\forall e, \mathcal{G}_e$ is the graph induced by $\mathtt{w}_e^{\mathrm{c}}$

Enumeration: $(\varphi_e^{\mathrm{c}})_{e\in\omega}$

Def

Fundamental functions are c-partials

Thm

Growth speed is dominated by fundamental functions

Thm

C-partial functions

Thm — Unusual closure

Not stable by composition

No s_n^m theorem

Counter examples — No composition

C contains

$$f : n \mapsto \begin{cases} BB_C^{\Phi}(p) & \text{if } n = 2p \\ p & \text{if } n = 2p+1 \end{cases}$$

and $g : n \mapsto 2n$, but not $f \circ g = BB_C^{\Phi}$.

No s_n^m theorem

C contains $f : \langle e, x \rangle \mapsto \varphi_e(x)$ but not all the recursive functions.

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def — $\forall e, \mathcal{G}_e$ is the graph induced by φ_e^C

Enumeration: $(\varphi_e^C)_{e \in \omega}$

Thm — ... partials

Thm — Growth speed is dominated by fundamental functions

c-partial functions

Thm
Unusual closure
Not stable by composition
No s_n^m theorem

Thm
Non-trivial c-creativity/productivity notion

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def
$\forall e, \mathcal{G}_e$ is the graph induced by $\mathbf{w}_e^{\mathrm{c}}$

Enumeration: $(\varphi_e^{\mathrm{c}})_{e\in\omega}$

Thm
Fundamental functions are c-partials

Thm
Growth speed is dominated by fundamental functions

c-partial functions

Thm

Unusual closure

Not stable by composition

No s_n^m theorem

Thm

Non-trivial c-creativity/productivity notion

Thm

Partial Kleene's second recursion theorem for a fundamental class c

For $f \in \mathsf{c}$ and h c-partial of domain A co-enumerable,

$\exists n$ s.t. $\left(\varphi_n^{\mathsf{c}}\right)\restriction_{\overline{A}} \cong \left(\varphi_{f(n)}^{\mathsf{c}}\right)\restriction_{\overline{A}}$ and $\left(\varphi_n^{\mathsf{c}}\right)\restriction_A \cong h$

A partial function is a function whose graph is enumerable.

A graph of a function is a well-formed set of integers.

Def

$\forall e$, $\mathcal{G}_e$ is the graph induced by $\mathbf{w}_e^{\mathsf{c}}$

Enumeration: $\left(\varphi_e^{\mathsf{c}}\right)_{e \in \omega}$

Thm

Fundamental functions are c-partials

Thm

Growth speed is dominated by fundamental functions

c-partial functions

Thm **Unusual closure**
Not stable by composition
No s^m_n theorem

Thm **Non-trivial c-creativity/productivity notion**

Thm **Partial Kleene's second recursion theorem for a fundamental class c**

For $f \in$ c and h c-partial of domain A co-enumerable,

$\exists n$ s.t. $(\varphi^c_n)_{|\overline{A}} \simeq (\varphi^c_{f(n)})_{|\overline{A}}$ and $(\varphi^c_n)_{|A} \simeq h$

A partial function is a function whose graph is enumerable.

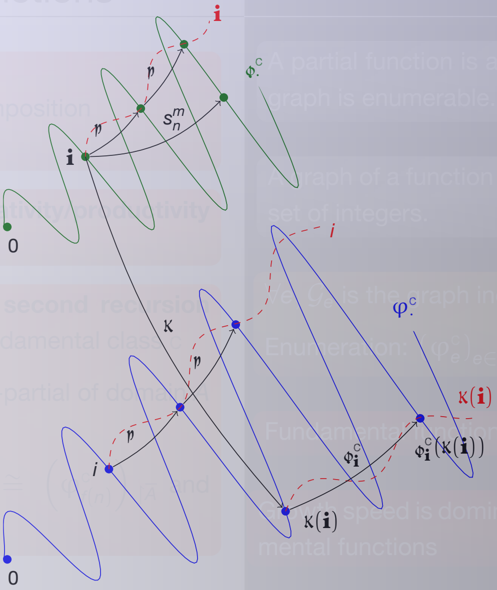
A graph of a function is a well-formed set of integers.

Def $\mathcal{G}_e$ is the graph induced by w^c_e
Enumeration: $(\varphi^c_e)_{e\in\omega}$

Thm Fundamental functions are c-partials

Thm Growth speed is dominated by fundamental functions

Summary



5. fragments of admissible recursion

Rising above

Fragments above computability

The case of Σ-recursion
Functions over sets in admissible levels of Gödel's L hierarchy

An enumeration $\left(\phi_e^{\mathbb{A}}\right)_{e\in\alpha}$ of Δ_0 (α-finite) sets
Plays the role of fundamental functions

An enumeration $\left(\varphi_e^{\mathbb{A}}\right)_{e\in\alpha}$ of Σ_1 (α-enumerable) sets
Plays the role of partial functions

Preliminary results
s_n^m-like theorem
Fixed-point theorem

Perspectives and conclusion

A general computability framework
for studying subrecursion and beyond

Perspectives and conclusion

A general computability framework

for studying subrecursion and beyond

Applications

Relativised notion of Kolmogorov complexity

General fine structure

Study of the $\mathbb{c}$-recursive ordinals

Ordinal iterations of the jump

Proof theory

Links between $\mathbb{c}$-degrees and honest degrees

Yielding results about minimal independent statements using $\mathbb{c}$ classes?

↓

Thank you for your attention.