Categorifying computability

Ruxandra Icleanu
Mentor: Julian Gould

University of Pennsylvania
DRP Spring 2023

Overview

- Computability
- A simpler model?
- Recursive & recursively enumerable sets

Computability

- How many functions are there?

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to \text{Bool}$, i.e. uncountably many

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to$ Bool, i.e. uncountably many
- Any programming language has some fixed, *finite* alphabet Σ

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to$ Bool, i.e. uncountably many
- Any programming language has some fixed, *finite* alphabet Σ
  $\Rightarrow$ countably many programs

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to \text{Bool}$, i.e. uncountably many
- Any programming language has some fixed, *finite* alphabet Σ
  $\Rightarrow$ countably many programs
- So not every function can be computed

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to \text{Bool}$, i.e. uncountably many
- Any programming language has some fixed, *finite* alphabet Σ
  $\Rightarrow$ countably many programs
- So not every function can be computed

How can we formalise this?

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions only of type $\mathbb{N} \to \text{Bool}$, i.e. uncountably many
- Any programming language has some fixed, *finite* alphabet Σ
  $\Rightarrow$ countably many programs
- So not every function can be computed

How can we formalise this?

- Turing machines

A Turing machine is formally defined as a 7-tuple $\langle Q, q_0, F, \Gamma, b, \Sigma, \delta \rangle$, where

- Q is a finite, non-empty set of *states*
- $q_0 \in Q$ is the *initial state*
- $F \subset Q$ is the set of *accepting states*
- Γ is a finite, non-empty set of *tape symbols*
- $b \in \Gamma$ is the *blank symbol*
- $\Sigma \subset \Gamma \setminus \{b\}$ is a set of *input symbols*
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*, which is a partial function

A Turing machine's operation is a sequence $(\langle q_s', h_s, (t_{s,p})_{p=-\infty}^{\infty} \rangle)_{s=0}^{\infty}$, where

- q_s is the *state* of the tape head in the s-th step
- h_s is the *position* of the tape head in the s-th step
- $t_{s,p}$ is the *content* of the tape in the s-th step and p-th position

such that

- $q_0' = q_0$
- $h_0 = 0$
- If the input string is $(x_1, \ldots, x_n) \in \Sigma^n$, then $t_{0,p} = x_{p+1}$ for $0 \leq p \leq n-1$ and $t_{0,p} = b$ otherwise
- If $\delta(q_s, t_{s,h_s}) = (q, t, d)$, then:
  - $q_{s+1} = q$
  - $h_{s+1} = h_s - 1$ if $d = L$, and $h_{s+1} = h_s + 1$ if $d = R$
  - $t_{s+1,p} = t$ if $p = h_s$, and $t_{s+1,p} = t_{s,p}$ otherwise
- Otherwise, if $q_s \in F$ or $\delta(q_s, t_{s,h_s})$ is not defined, then the machine *halts*; the rest of the sequence is not defined, and s is the *number of steps* taken by the machine.

Figure: Formal definition of a Turing machine (Source)

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions $f : \mathbb{N} \to \text{Bool}$ functions, so uncountably many
- countably many programs
- $\Rightarrow$ not every function can be computed

How can we formalise this?

- Turing machines?

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions $f : \mathbb{N} \to$ Bool functions, so uncountably many
- countably many programs
- $\Rightarrow$ not every function can be computed

How can we formalise this?

- Turing machines?
- register machines?

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions $f : \mathbb{N} \to$ Bool functions, so uncountably many
- countably many programs
- $\Rightarrow$ not every function can be computed

How can we formalise this?

- Turing machines?
- register machines?
- boolean circuits?

Computability

- How many functions are there?
  $2^{|\mathbb{N}|}$ functions $f : \mathbb{N} \to$ Bool functions, so uncountably many
- countably many programs
- $\Rightarrow$ not every function can be computed

How can we formalise this?

- Turing machines?
- register machines?
- boolean circuits?

Would want a simpler model.

Definition

We define a category **Func** with

- objects: types
- morphisms: functions with input/ output of the specified type t_1, t_2 types
  $f : t_1 \to t_2$ has input of type t_1 and output of type t_2
- identity, composition (just usual function composition)

e.g. $\mathrm{Derivative} : \mathrm{Real}^{\mathrm{Real}} \to \mathrm{Real}^{\mathrm{Real}} \in \mathrm{Arr}(\mathrm{Func})$

Definition

We define a category **Func** with

- objects: types
- morphisms: functions with input/ output of the specified type t_1, t_2 types
  $f : t_1 \to t_2$ has input of type t_1 and output of type t_2
- identity, composition (just usual function composition)

e.g. Derivative : $\mathrm{Real}^{\mathrm{Real}} \to \mathrm{Real}^{\mathrm{Real}} \in \mathrm{Arr}(\mathrm{Func})$

Definition

We define a category **CompFunc** to be a subcategory of **Func** in which we restrict the morphismsm to functions for which there exists a Python program that can compute them.

Definition

We define a category **Func** with

- objects: types
- morphisms: functions with input/ output of the specified type t_1, t_2 types
  $f : t_1 \to t_2$ has input of type t_1 and output of type t_2
- identity, composition (just usual function composition)

e.g. $\text{Derivative} : \text{Real}^{\text{Real}} \to \text{Real}^{\text{Real}} \in \text{Arr}(\text{Func})$

Definition

We define a category **CompFunc** to be a subcategory of **Func** in which we restrict the morphismsm to functions for which there exists a Python program that can compute them.

Definition

We define a category **TotCompFunc** to be a subcategory of **Func** in which we restrict the morphismsms to total functions for which there exists a Python program that can compute them.

Clearly,

$$\mathsf{TotCompFunc} \hookrightarrow \mathsf{CompFunc} \hookrightarrow \mathsf{Func}$$

Clearly,

$$\text{TotCompFunc} \hookrightarrow \text{CompFunc} \hookrightarrow \text{Func}$$

Aside

Func, CompFunc, and TotCompFunc are very structured categories – they are symmetric monoidal categories.

- monoidal category: can compose the morphisms 'in parallel'

  Given $x, x' \in ob(\mathcal{C})$, we can form $x \otimes x' \in ob(\mathcal{C})$
  Given $x, x' \in ob(\mathcal{C})$, $y, y' \in (\mathcal{D})$ s.t. there exists $f : x \to y$ and $f' : x' \to y'$, we can also form

  $$f \otimes f' : x \otimes x' \to y \otimes y'$$

- symmmetric: when doing so, the order is not important

$$f' \otimes f = f \otimes f'$$

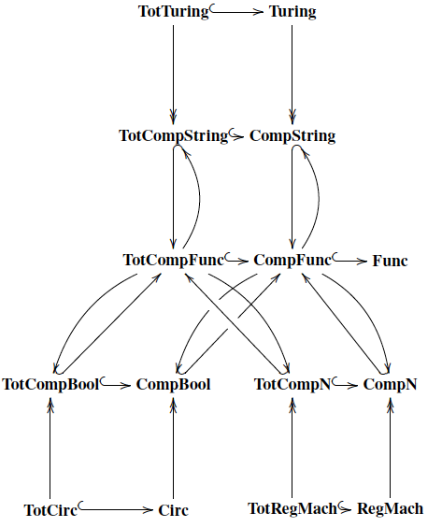

Figure: Big picture of models

(from Yanofsky's "Theoretical Computer Science for the Working Category Theorist)

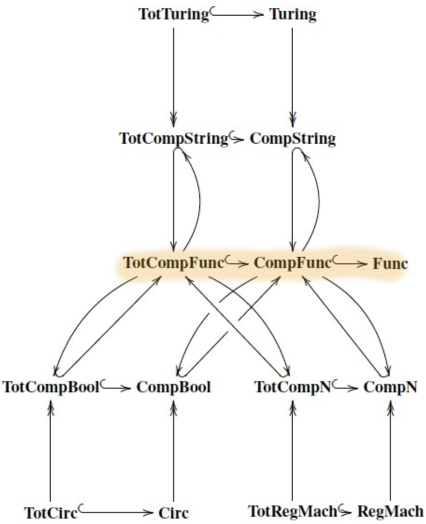

Figure: Big picture of models

(from Yanofsky's "Theoretical Computer Science for the Working Category Theorist)

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S?

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers?

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓
- the set of even numbers?

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓
- the set of even numbers? ✓

Recursive sets

Question: I give you some set S. You need to answer queries of the form $'x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓
- the set of even numbers? ✓
- encode every Python program P by a natural number p, and define $S = \{p \mid P \text{ stops in a finite number of steps}\}$?

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S?'$ with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓
- the set of even numbers? ✓
- encode every Python program P by a natural number p, and define $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✗

Recursive sets

Question: I give you some set S. You need to answer queries of the form '$x \in S$?' with 'Yes/No'. Can you always give an answer?

- finite S? ✓
- the set of prime numbers? ✓
- the set of even numbers? ✓
- encode every Python program P by a natural number p, and define $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✗

Intuition: The answer is 'yes' if S has some well-defined structure.

Recursive sets

Definition

Let $A \subseteq \mathbb{N}$. We define the characteristic function of A to be

$$\chi_A := \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

χ_A is a morphism $\mathbb{N} \to \text{Bool}$ in **Func**.

Definition

We call A recursive (computable) if its characteristic function $\chi_A : \mathbb{N} \to \text{Bool}$ is a morphism in **TotCompFunc**.

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set?

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep?

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$?

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✓

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✓
- $\overline{S}$?

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✓
- $\overline{S}$? ✗

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✓
- $\overline{S}$? ✗

Intuition: The answer is 'yes' if elements of T can be enumerated in order.

Recursively enumerable sets

Question: I give you some set T. Now, you need to answer queries of the form '$x \in T$' with 'Yes' when this is the case. Is this always possible?

- T recursive set? ✓
- set constructed by picking some element from a uniform distribution at every timestep? ✓
- $S = \{p \mid P \text{ stops in a finite number of steps}\}$? ✓
- $\overline{S}$? ✗

Intuition: The answer is 'yes' if elements of T can be enumerated in order.

Definition

Let $A \subseteq \mathbb{N}$. We define the function $\tilde{\chi}_A$ to be

$$\tilde{\chi}_A := \begin{cases} 1 & \text{if } x \in A \\ \uparrow & \text{otherwise.} \end{cases}$$

$\tilde{\chi}_A$ is a morphism $\mathbb{N} \to \text{Bool}$ in **Func**.

Recursively enumerable sets

Definition

Let $A \subseteq \mathbb{N}$. We call A recursively enumerable (r.e.) if the function $\tilde{\chi}_A : \mathbb{N} \to \text{Bool}$ is a morphism in **CompFunc**.

Recursively enumerable sets

Definition

Let $A \subseteq \mathbb{N}$. We call A recursively enumerable (r.e.) if the function $\tilde{\chi}_A : \mathbb{N} \to \text{Bool}$ is a morphism in **CompFunc**.

Theorem

$A \subseteq \mathbb{N}$ computable if and only if A and $\overline{A}$ are r.e.

Recursively enumerable sets

Definition

Let $A \subseteq \mathbb{N}$. We call A recursively enumerable (r.e.) if the function $\tilde{\chi}_A : \mathbb{N} \to \mathrm{Bool}$ is a morphism in **CompFunc**.

Theorem

$A \subseteq \mathbb{N}$ computable if and only if A and $\overline{A}$ are r.e.

Proof sketch.

Want to show: $\chi_A \in \mathrm{Arr}(\mathbf{TotCompFunc}) \Leftrightarrow \tilde{\chi}_A, \tilde{\chi}_{\overline{A}} \in \mathrm{Arr}(\mathbf{CompFunc})$.

Recursively enumerable sets

Definition

Let $A \subseteq \mathbb{N}$. We call A recursively enumerable (r.e.) if the function $\tilde{\chi}_A : \mathbb{N} \to \mathrm{Bool}$ is a morphism in **CompFunc**.

Theorem

$A \subseteq \mathbb{N}$ computable if and only if A and $\overline{A}$ are r.e.

Proof sketch.

Want to show: $\chi_A \in \mathrm{Arr}(\mathbf{TotCompFunc}) \Leftrightarrow \tilde{\chi}_A, \tilde{\chi}_{\overline{A}} \in \mathrm{Arr}(\mathbf{CompFunc})$.
"$\Rightarrow$" easy
"$\Leftarrow$" Want to construct χ_A as a composition of morphisms in **CompFunc**.

$$\mathbb{N} \xrightarrow{n \mapsto (n,n)} \mathbb{N} \times \mathbb{N} \xrightarrow{\tilde{\chi}_A \times \tilde{\chi}_{\overline{A}}} \mathrm{Bool} \times \mathrm{Bool} \xrightarrow{1 \times \mathrm{Not}} \mathrm{Bool} \times \mathrm{Bool} \xrightarrow{\mathrm{Parallel}} \mathrm{Bool}$$

$\square$

References

[1] Noson S. Yanofsky (2022)
"Theoretical Computer Science for the Working Category Theorist"

[2] Nigel Cutland (1980)
"Computability: An introduction to recursive function theory"