

# Post's Problem and The Priority Method in Synthetic Computability

Haoyi Zeng

**Advisors:** Yannick Forster and Dominik Kirst

**Supervisor:** Prof. Gert Smolka

**Programming Systems Lab**



# Background

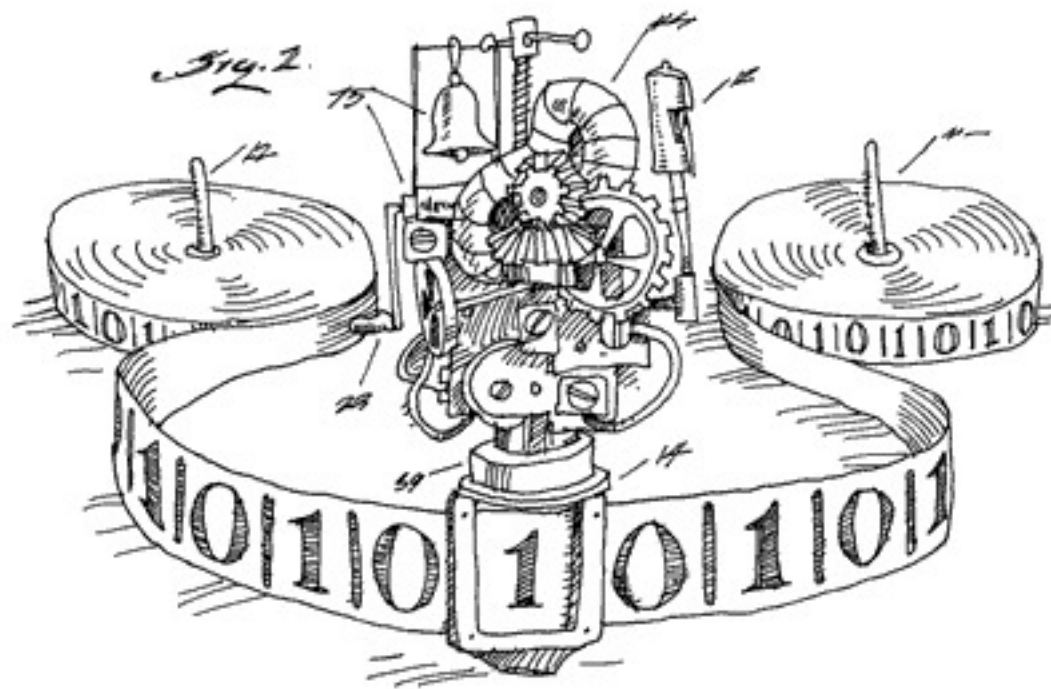
# Synthetic Computability

Recap

$P$  is Decidable:  $\exists f : X \rightarrow \mathbb{B} . P x \leftrightarrow f x = \text{tt}$   ~~$\wedge f$  is computable~~

What is computable ?

Turing machine



$\lambda$ -Calculus

$$\text{fix } F := \lambda x . \text{fix}' \text{ fix}' F x$$
$$\text{fix}' := \lambda f, F . F (\lambda x . f f F x)$$

Synthetic Computability

# Synthetic Computability

A predicate  $P : X \rightarrow \mathbb{P}$  is

**Decidable**  $\exists f : X \rightarrow \mathbb{B} . P x \leftrightarrow f x = \text{tt}$

**Semi-decidable**  $\exists f : X \rightarrow \mathbb{N} \rightarrow \mathbb{B} . P x \leftrightarrow \exists n . f x n = \text{tt}$

“Does a Turing machine halt  
on a given input?”

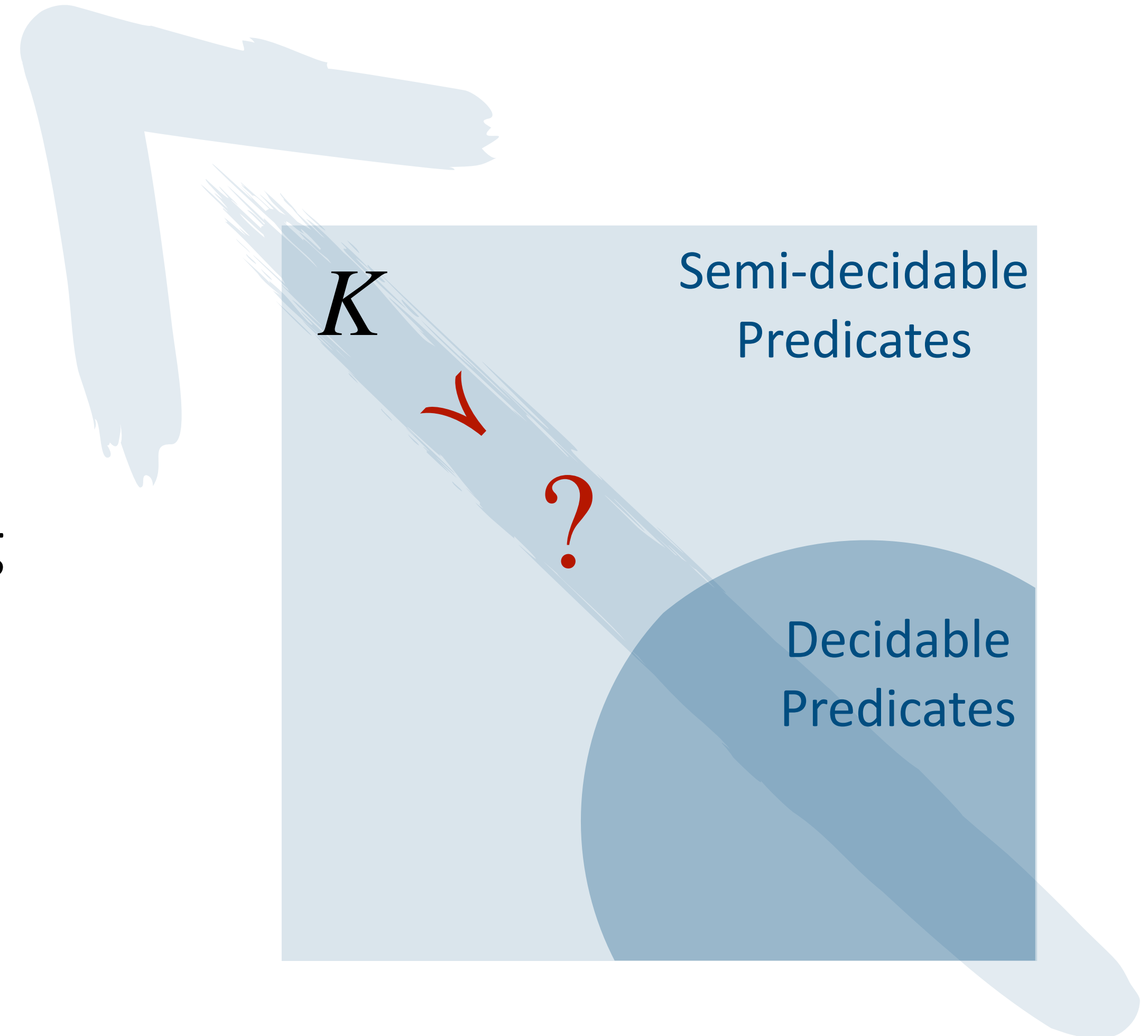
**Halting Problem**  $K$

$K x \leftrightarrow x$ -th partial function halts on  $x$

# Post's Problem

“Is there an undecidable, semi-decidable predicate that is strictly **easier than** the Halting problem?”

- Post, 1944

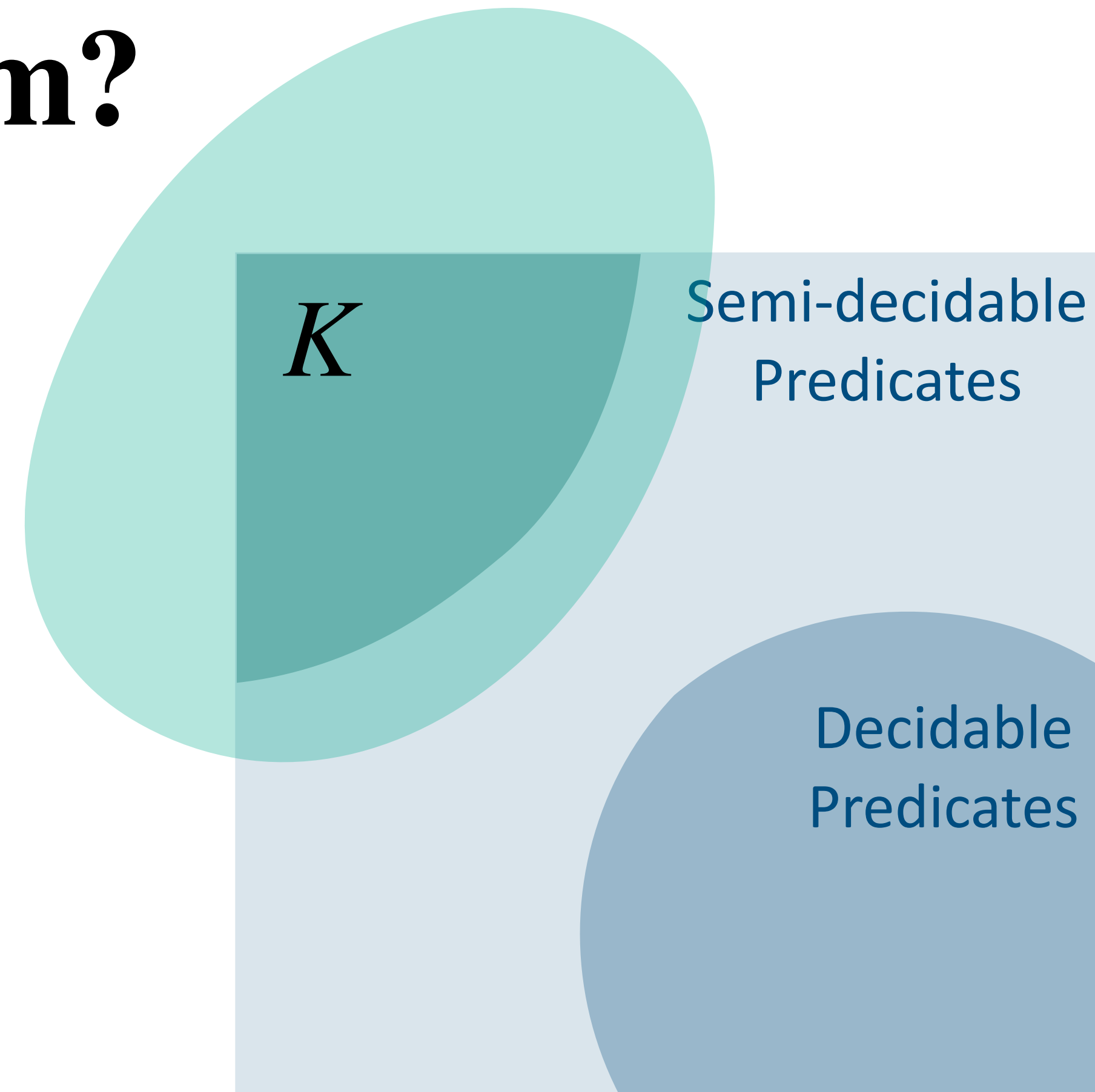


# Easier than Halting Problem?

$K$  is reducible to  $P$

Many-one reduction:  $K \leq_m P$

Truth-table reduction:  $K \leq_{tt} P$

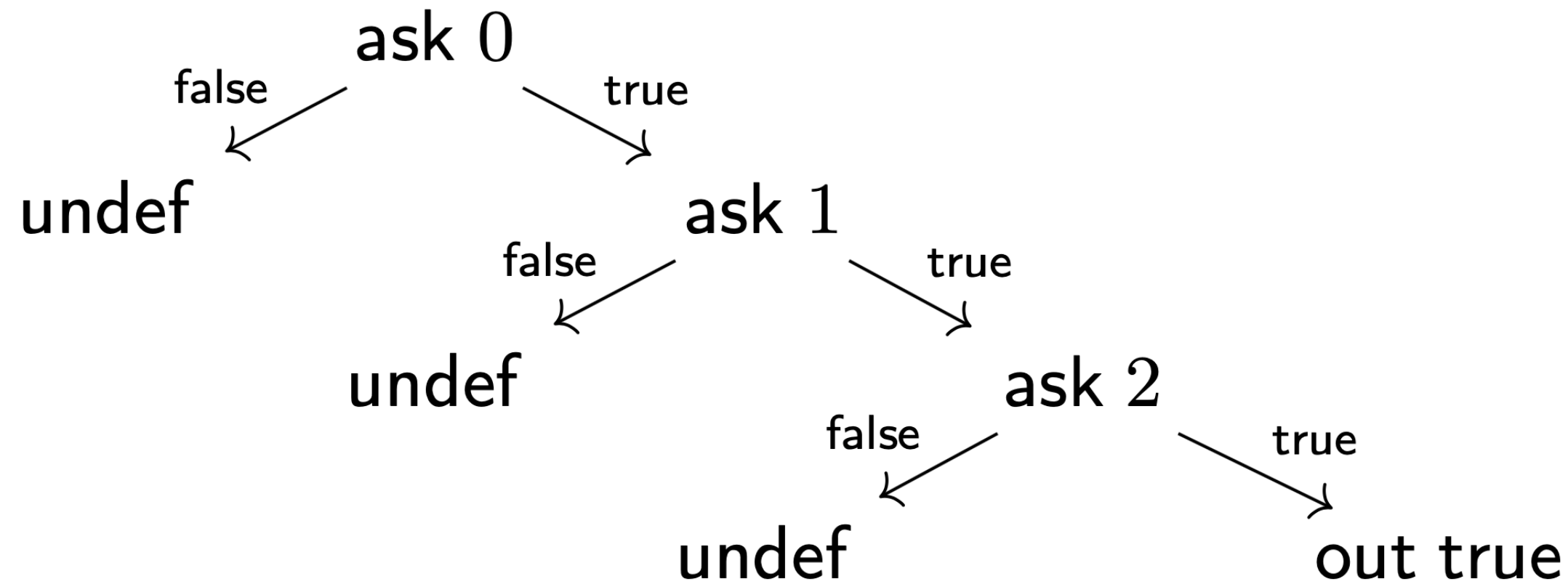


Consider reductions in the most general sense, i.e., **Turing reduction**, which is also the problem Post left open in his paper.

# Turing reducible in synthetic computability

Modelling Oracle Computable (O. C.):

$F$  is O.C. if  $F$  described by such computable tree [Forster, Kirst & Mück 2023]



**Oracle Computability and Turing Reducibility in the Calculus of Inductive Constructions\***

Yannick Forster<sup>1</sup>, Dominik Kirst<sup>2,3</sup>, and Niklas Mück<sup>3</sup>

<sup>1</sup> Inria, LS2N, Université Nantes, France  
yannick.forster@inria.fr  
<sup>2</sup> Ben-Gurion University of the Negev, Beer-Sheva, Israel  
kirst@cs.bgu.ac.il  
<sup>3</sup> Saarland University and MPI-SWS, Saarland Informatics Campus, Saarbrücken, Germany  
smu@informatik.uni-saarland.de

**Abstract.** We develop synthetic notions of oracle computability and Turing reducibility in the Calculus of Inductive Constructions (CIC), the constructive type theory underlying the Coq proof assistant. As usual in synthetic approaches, we employ a definition of oracle computations based on meta-level functions rather than object-level models of computation, relying on the fact that in constructive systems such as CIC all definable functions are computable by construction. Such an approach lends itself well to machine-checked proofs, which we carry out in Coq. There is a tension in finding a good synthetic rendering of the higher-order notion of oracle computability. On the one hand, it has to be informative enough to prove central results, ensuring that all notions are faithfully captured. On the other hand, it has to be restricted enough to benefit from axioms for synthetic computability, which usually concern first-order objects. Drawing inspiration from a definition by Andrej Bauer based on continuous functions in the effective topos, we use a notion of sequential continuity to characterise valid oracle computations. As main technical results, we show that Turing reducibility forms an upper semilattice, transports decidability, and is strictly more expressive than truth-table reducibility, and prove that whenever both a predicate  $p$  and its complement are semi-decidable relative to an oracle  $q$ , then  $p$  Turing-reduces to  $q$ .

**Keywords:** Type theory · Logical foundations · Synthetic computability theory · Coq proof assistant

\* Yannick Forster received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101024453. Dominik Kirst is supported by a Minerva Fellowship of the Minerva Stiftung Gesellschaft für die Forschung mbH.

**7 Properties of Turing Reducibility**

We continue with similarly standard properties of Turing reducibility. Again, all proofs are concise but precise. As a preparation, we first note that Turing reducibility can be characterised without the relational layer.

**Lemma 20.**  $p \leq q$  if and only if there is  $r$  such that for all  $x$  and  $b$  we have  $\text{feb} \leftrightarrow \exists p a s . r x ; q \cdot q s ; a s \wedge r x \text{ asout } b$ .

Now to begin, we show that Turing reducibility is a preorder.

**Theorem 21.** Turing reducibility is reflexive and transitive.

*Proof.* Reflexivity follows directly by the identity functional being computable via Lemma 4. Transitivity follows with Lemma 8.  $\square$

In fact, Turing reducibility is an upper semilattice:

**Theorem 22.** Let  $p : X \rightarrow \mathbb{P}$  and  $q : Y \rightarrow \mathbb{P}$ . Then there is a lowest upper bound  $p + q : X + Y \rightarrow \mathbb{P}$  w.r.t.  $\leq$ . Let  $(p + q) p + q$  be the join of  $p$  and  $q$  w.r.t.  $\leq$ .  $p \leq r$  and  $q \leq r$  then  $p + q \leq r$ .

*Proof.* The first two claims follow by let  $F_i$  reduce  $p$  to  $r$  and be computable  $r_i$ . Define

$$FRx o := \begin{cases} F_i R x o & \text{if } z = \text{inl } x \\ F_j R x o & \text{if } z = \text{inr } y \end{cases}$$

$r$  computes  $F$ , and  $F$  reduces  $p + q$ .

We continue by establishing proper oracle semi-decidability discussed in the non-relativised notion of decidability.

We define oracle computability by observing that a terminating computation with oracles has a sequential form: in any step of the sequence, the oracle computation can ask a question to the oracle, return an output, or diverge. Informally, we can enforce such sequential behaviour by requiring that every terminating computation  $FRi o$  can be described by (finite, possibly empty) lists  $q s : Q^*$  and  $a s : A^*$  such that from the input  $i$  the output  $o$  is eventually obtained after a finite sequence of steps, during which the questions in  $q s$  are asked to the oracle one-by-one, yielding corresponding answers in  $a s$ . This computational data can be captured by a partial<sup>3</sup> function of type  $I \rightarrow A^* \rightarrow Q + O$ , called the (computation) tree of  $F$ , that on some input and list of previous answers either returns the next question to the oracle, returns the final output, or diverges.

So more formally, we call  $F : (Q \rightarrow A \rightarrow \mathbb{P}) \rightarrow (I \rightarrow O \rightarrow \mathbb{P})$  an (oracle-)computable functional if there is a tree  $r : I \rightarrow A^* \rightarrow Q + O$  such that

$$\forall R i o . FRi o \leftrightarrow \exists q s a s . r i ; R \cdot q s ; a s \wedge r i \text{ asout } o$$

with the interrogation relation  $\sigma ; R \vdash q s ; a s$  as being defined inductively by

$$\frac{\sigma ; R \vdash [] ; []}{\sigma ; R \vdash [] ; []} \quad \frac{\sigma ; R \vdash q s ; a s \quad \sigma \text{ asask } q \quad R q a}{\sigma ; R \vdash q s ++ [q] ; a s ++ [a]}$$

where  $A^*$  is the type of lists over  $a$ ,  $l ++ l'$  is list concatenation, where we use the suggestive shorthands  $\text{ask } q$  and  $\text{out } o$  for the respective injections into the sum type  $Q + O$ , and where  $\sigma : A^* \rightarrow Q + O$  denotes a tree at a fixed input  $i$ .

To provide some further intuition and visualise the usage of the word “tree”, we discuss the following example functional in more detail:

$$F : (N \rightarrow \mathbb{B} \rightarrow \mathbb{P}) \rightarrow (N \rightarrow \mathbb{B} \rightarrow \mathbb{P})$$

$$FRi o := o = \text{true} \wedge \forall q < i . R q \text{ true}$$

**Turing reduction**  $P \leq Q := \exists F . F \text{ is O.C.} \wedge \forall x . P x \leftrightarrow F \hat{Q} x \text{ tt}$   
 $\neg P x \leftrightarrow F \hat{Q} x \text{ ff}$

# Solutions to Post's Problem

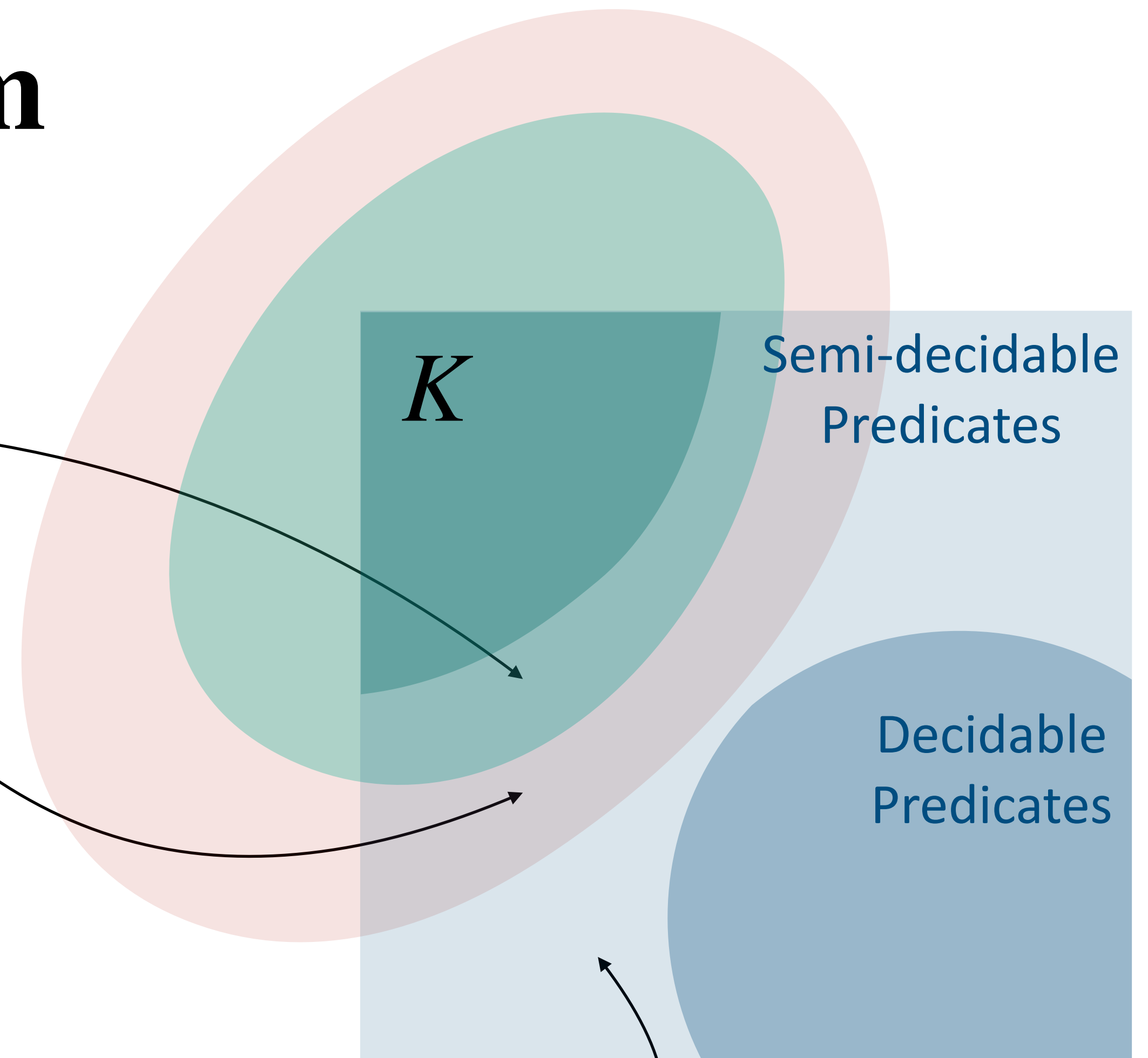
Finite extension  
method [Post 1944]

- Simple Set
- Hyper Simple Set

in synthetic computability  
[Forster & Jahn 2023]

Priority  
Method

- Friedberg–Muchnik Theorem [Mučnik 1956] [Friedberg 1957]
- LOW** Simple Set [Lerman & Soare 1980] [Soare 1999]





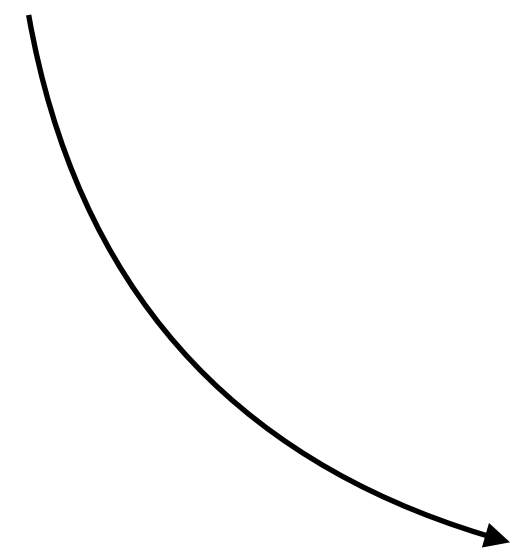
# Lowness

“Showing  $P$  is **limit computable** is difficult!”

**Lowness** Turing jump of  $P$  is reducible to halting problem:  $P' \leq K$



$P' x \leftrightarrow x$ -th oracle machine with oracle  $P$  halts on  $x$



**Low Simple Set** [Lerman & Soare 1980] [Soare 1999]

# **Current State**

# Limit Computable in synthetic computability

[Shoenfield 1959] [Gold 1965]

A **uniform** sequence  $f : \mathbb{N} \rightarrow Y$  is **convergent** to some value  $b$  if :

$$\lim_{n \rightarrow \infty} f(n) = b \quad \text{iff} \quad \exists n : \mathbb{N} . \forall m \geq N . f(m) = b$$

A given predicate  $P$  is termed **limit computable** when there is a decider  $f : X \rightarrow \mathbb{N} \rightarrow \mathbb{B}$  s.t.

$$\forall x . \quad P x \leftrightarrow \lim_{n \rightarrow \infty} f(x, n) = \text{tt}$$

$$\neg P x \leftrightarrow \lim_{n \rightarrow \infty} f(x, n) = \text{ff}$$

# Example

Fix an input  $x$ , test whether  $x$  is in a limit computable predicate  $P$  by executing the function  $f$ :

$$f(x,0) \ f(x,1) \ f(x,2) \ f(x,3) \ \cdots \ \begin{array}{c} n \\ \vdots \\ \vdots \\ \vdots \end{array} \ f(x,n) \ f(x,n+1) \ f(x,n+2) \ f(x,n+3) \ \cdots$$

It doesn't matter what any of the runs turn out to be, we need to observe the **limits**

# Limit Lemma 1

**Lemma 1:** If a predicate  $P$  is limit computable, then both  $P$  and  $\bar{P}$  are  $\Sigma_2$  predicates.

*Proof.* Rewrite the definition:

$$P x \iff \exists n . \forall m \geq n . f(x, m) = \text{tt}$$

$$\bar{P} x \iff \neg P x \iff \exists n . \forall m \geq n . f(x, m) = \text{ff}$$

**Lemma 2:** If both  $P$  and  $\bar{P}$  are  $\Sigma_2$  predicates, then  $P$  is reducible to  $K$ .

*Proof.* By Post's theorem. [Forster, Kirst & Mück 2024]

# Limit Lemma 2

**Lemma 3:** A predicate  $P$  is limit computable, if  $P$  is reducible to  $K$ .

*Proof.* Define a step-indexing function:  $\Phi_e^K(x)[n] = \Phi_{e, n}^{K_n}(x)$ ,  
since  $K := \bigcup_{n \in \mathbb{N}} K_n$  can be approximated by an accumulative sequence.

$$P x \iff \lim_{n \rightarrow \infty} \Phi_e^K(x)[n] = \text{tt} \qquad \neg P x \iff \lim_{n \rightarrow \infty} \Phi_e^K(x)[n] = \text{ff}$$

**Corollary:** A predicate  $P$  is limit computable iff  $P$  is reducible to  $K$ .



# Outlook

# Priority Method

Positive Requirements  $P_e := W_e$  is infinite  $\rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 \prec N_1 \prec P_2 \prec N_2 \prec P_3 \prec N_3 \cdot \cdot \cdot$$



# Priority Method

Positive Requirements  $P_e := W_e \text{ is infinite} \rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 \prec N_1 \prec P_2 \prec N_2 \prec P_3 \prec N_3 \cdot \cdot \cdot$$

# Priority Method

Positive Requirements  $P_e := W_e$  is infinite  $\rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 \prec N_1 \prec P_2 \prec N_2 \prec P_3 \prec N_3 \cdot \cdot \cdot$$

# Priority Method

Positive Requirements  $P_e := W_e \text{ is infinite} \rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 \prec N_1 \prec P_2 \prec N_2 \prec P_3 \prec N_3 \cdot \cdot \cdot$$

# Priority Method

Positive Requirements  $P_e := W_e \text{ is infinite} \rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 < N_1 < P_2 < N_2 < P_3 < N_3 \cdot \cdot \cdot$$

# Priority Method

Positive Requirements  $P_e := W_e$  is infinite  $\rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 \prec N_1 \prec P_2 \prec N_2 \prec P_3 \prec N_3 \cdot \cdot \cdot$$

# Priority Method

Positive Requirements  $P_e := W_e$  is infinite  $\rightarrow W_e \cap A \neq \emptyset$

Negative Requirements  $N_e := (\exists^\infty s . \Phi_\xi^A(e)[s] \downarrow) \rightarrow \Phi_\xi^A(e) \downarrow$

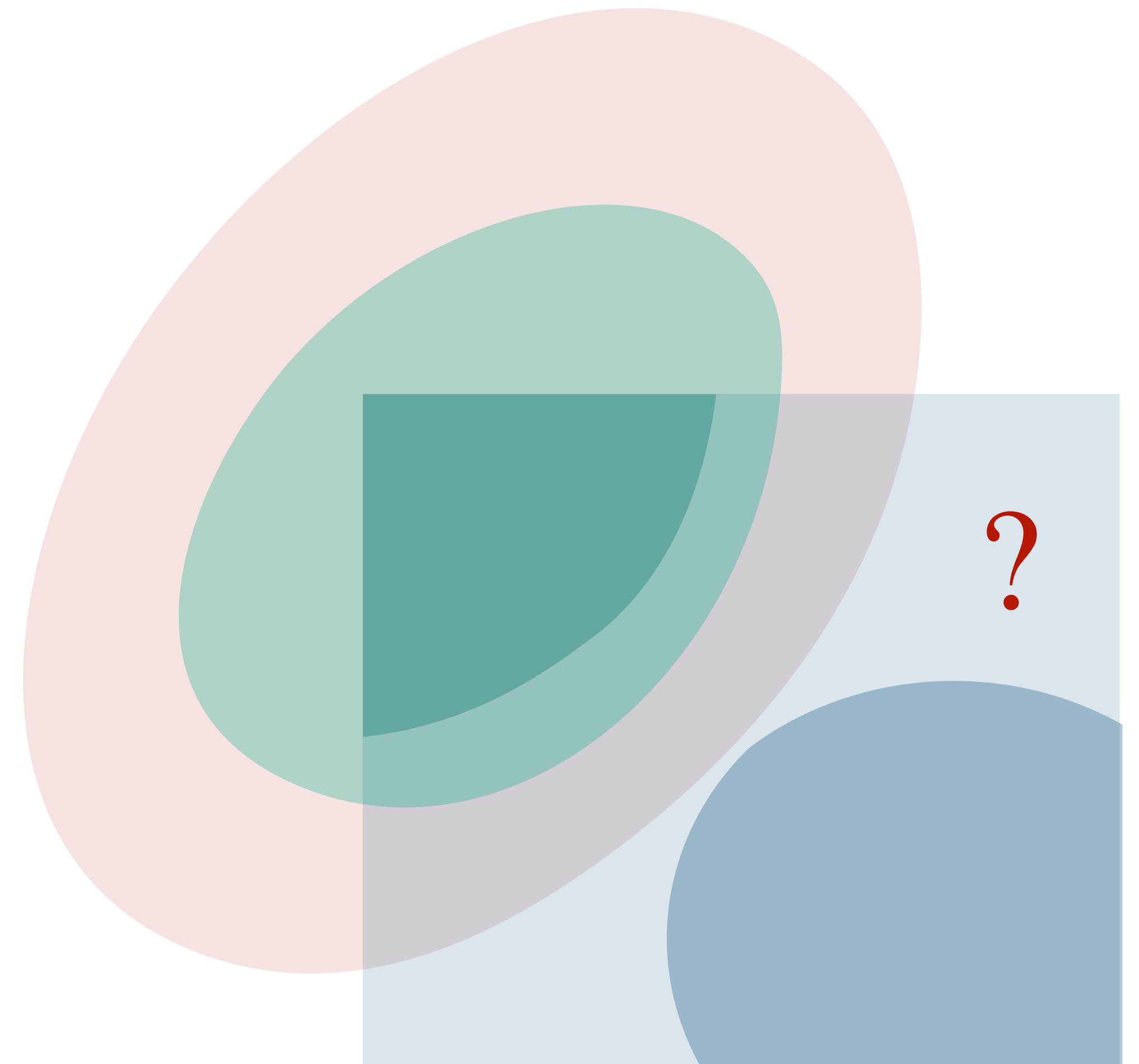
Construct a predicate  $A := \bigcup_{n \in \mathbb{N}} A_n$  stage by stage such that:

$$P_1 < N_1 < P_2 < N_2 < P_3 < N_3 \cdot \cdot \cdot$$

# Goals

We aim to show the following theorem and construction in synthetic computability:

- Definition of limit computable
- Limit lemma
- **The priority method**
- Definition of low simple predicate
- Existence of low simple predicate
- Friedberg-Muchnik Theorem
- Constructive analysis



# References

[Forster, Kirst & Mück 2023] Yannick Forster, Dominik Kirst and Niklas Mück. Oracle computability and Turing reducibility in the calculus of inductive constructions.  
*APLAS 2023*

[Post 1944] Post, E. L. Recursively enumerable sets of positive integers and their decision problems.

[Forster & Jahn 2023] Yannick Forster and Felix Jahn. Constructive and Synthetic Reducibility Degrees: Post's Problem for Many-one and Truth-table Reducibility in Coq.  
*CSL 2023*



# References

[Friedberg 1957] Richard M. Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944). [Proceedings of the National Academy of Sciences of the United States of America](#). Vol. 43, no. 2. pp. 236–238.

[Lerman & Soare 1980] LERMAN, M., AND SOARE, R. d-simple sets, small sets, and degree classes. [Pacific Journal of Mathematics](#) 87, 1 (1980), 135–155.

[Soare 1999] SOARE, R. Recursively enumerable sets and degrees: A study of computable functions and computably generated sets. [Springer Science & Business Media](#), 1999.

# References

[Mučnik 1956] Mučnik Albert Abramovich On the unsolvability of the problem of reducibility in the theory of algorithms. [Doklady Akademii Nauk SSSR. 108: 194–197.](#)

[Forster, Kirst & Mück 2024] Yannick Forster, Dominik Kirst and Niklas Mück. The kleene-post and post's theorem in the calculus of inductive constructions [CSL](#)

[Shoenfield 1959] Shoenfield, J. R. On degrees of unsolvability. [Annals of mathematics 69, 3 \(1959\), 644–653.](#)

[Gold 1965] Gold, E. M. Limiting recursion. [The Journal of Symbolic Logic 30, 1 \(1965\), 28–48.](#)

# Appendix A

## Oracle Computable

Based on a notion of computability of functionals  $F : (Q \rightarrow A \rightarrow \mathbb{P}) \rightarrow (I \rightarrow Q \rightarrow \mathbb{P})$ , The argument  $R : Q \rightarrow A \rightarrow \mathbb{P}$  is to be read as the oracle relating questions  $q : Q$  to answers  $a : A$ ,  $i : I$  is the input to the computation, and  $o : O$  is the output, such an  $F$  is considered (oracle)-computable if there is an underlying computation tree  $\tau : I \rightarrow A^* \rightarrow (Q + O)$ :

$$\forall R x b . F R x b \iff \exists qs as . \tau x; R \vdash qs; as \wedge \tau x as \triangleright \text{out } b$$

where the interrogation relation  $\sigma; R \vdash qs; as$  is inductively defined for  $\sigma : A^* \rightarrow Q + O$  as:

$$\frac{}{\sigma ; R \vdash []; []} \qquad \frac{\sigma ; R \vdash qs; as \quad \sigma as \triangleright \text{ask } q \quad R(q, a)}{\sigma ; R \vdash qs@[q]; as@[a]}$$

# Appendix B

## Step index function

We insert this oracle  $O$  into our Turing machine by fixing a  $n$ , and subsequently run  $\tau$ . Based on this effectively computable oracle, we can define a total function  $\Phi$  as follows:

$$\Phi_{\tau}^{O(n)}(x, i, j) := \begin{cases} \ulcorner \text{out } o \urcorner & \text{if } (\tau, x, []) \rightsquigarrow_j \text{out } o \\ \ulcorner \text{ask } q \urcorner & \text{if } (\tau, x, []) \rightsquigarrow_j \text{ask } q \text{ and } i = 0 \\ \Phi_{\tau @ [\chi_O, n, q]}^{O(n)}(x, i', j) & \text{if } (\tau, x, []) \rightsquigarrow_j \text{ask } q \text{ and } i = S i' \\ \text{none} & \text{otherwise} \end{cases}$$

Given that  $P$  is Turing reducible to  $\emptyset$ , we obtain the computable tree  $\tau$ . Building upon the step-index function described above, we define the following function:

$$\chi_P(s, x) := \begin{cases} b & \text{if } \Phi_{\tau}^K(x)[s] = \ulcorner b \urcorner \\ \text{tt} & \text{otherwise} \end{cases}$$

# Low Simple Predicate 1

undecidable predicate

**Simple predicate:**

If a predicate  $P$  is simple, then  $P$  is semi-decidable and  $\neg(P \preceq \emptyset)$

**Turing jump of  $P$ :**

$P' x \leftrightarrow x$ -th oracle machine with oracle  $P$  halts on  $x$

# Low Simple Predicate 2

**Low predicate:** A predicate  $P$  is low, if the Turing jump of  $P$  is reducible to  $K$

$$P' \leq K \Rightarrow \neg(K \leq P)$$

**Low Simple predicate:**  $\emptyset < P < K$ , where  $P < Q := P \leq Q \wedge \neg(P \leq Q)$

A positive solution to Post's Problem

Showing a predicate is reducible to  $K$  is **difficult!**