Enumerating Error Bounded Polytime Algorithms Through Arithmetical Theories

Melissa Antonelli **Ugo Dal Lago** **Davide Davoli**
Isabel Oitavem **Paolo Pistone**

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

informatiques mathématiques

European Research Council
Established by the European Commission

Radbound University, April 18th 2024

Part I

Context and Motivation

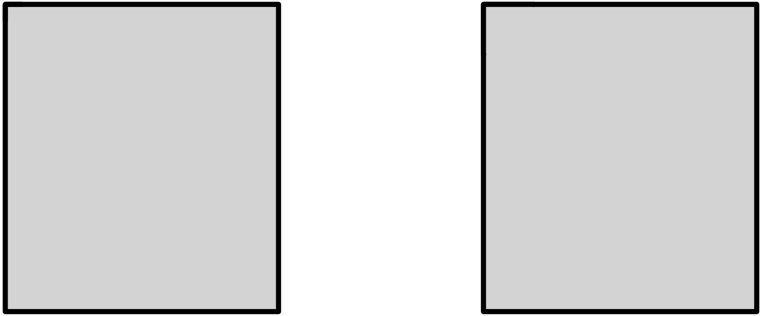
Implicit Computational Complexity

Programs

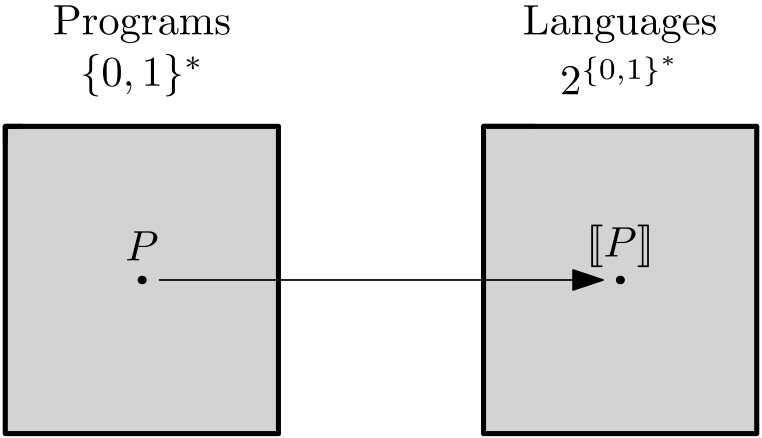
$\{0,1\}^*$

Languages

$2^{\{0,1\}^*}$



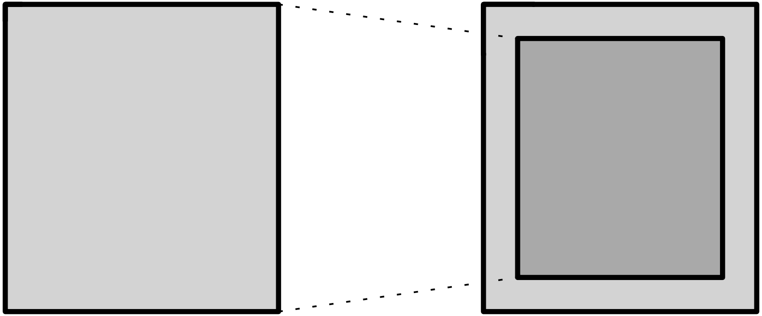
Implicit Computational Complexity



Implicit Computational Complexity

Programs
$\{0,1\}^*$

Languages
$2^{\{0,1\}^*}$



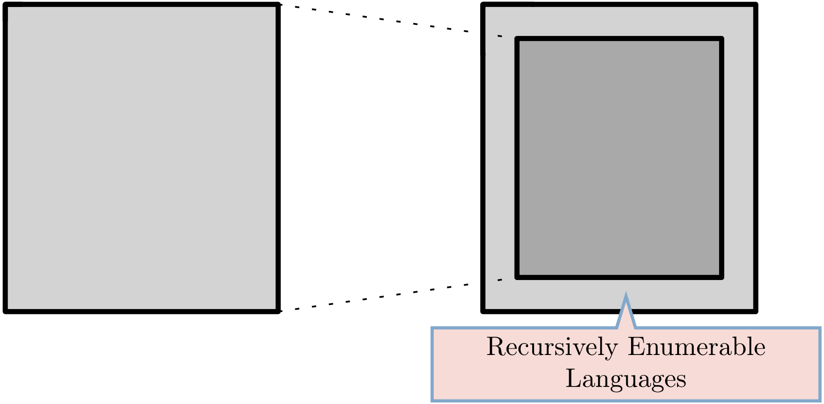
Implicit Computational Complexity

Programs

$\{0,1\}^*$

Languages

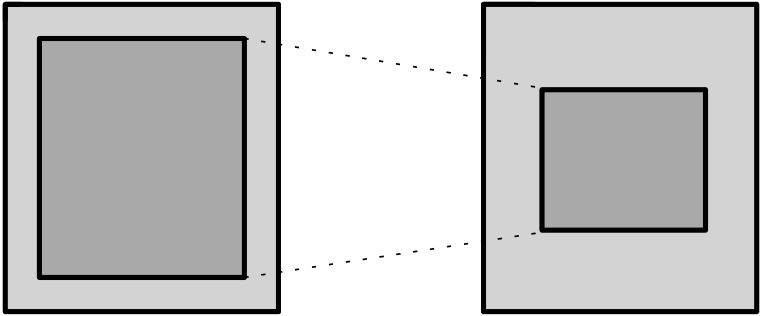
$2^{\{0,1\}^*}$



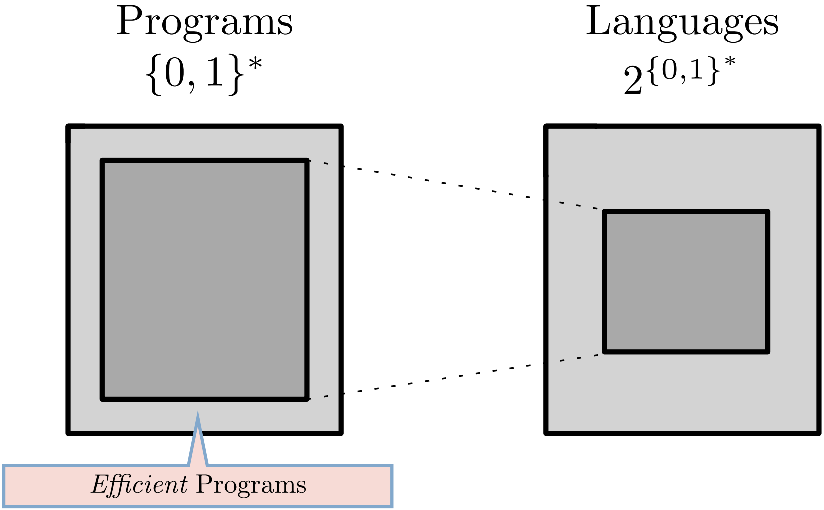
Implicit Computational Complexity

Programs
$\{0,1\}^*$

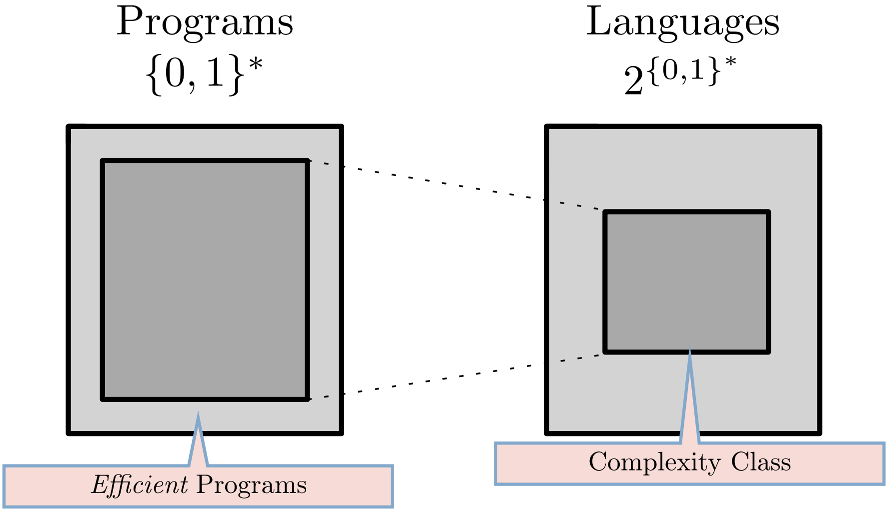
Languages
$2^{\{0,1\}^*}$



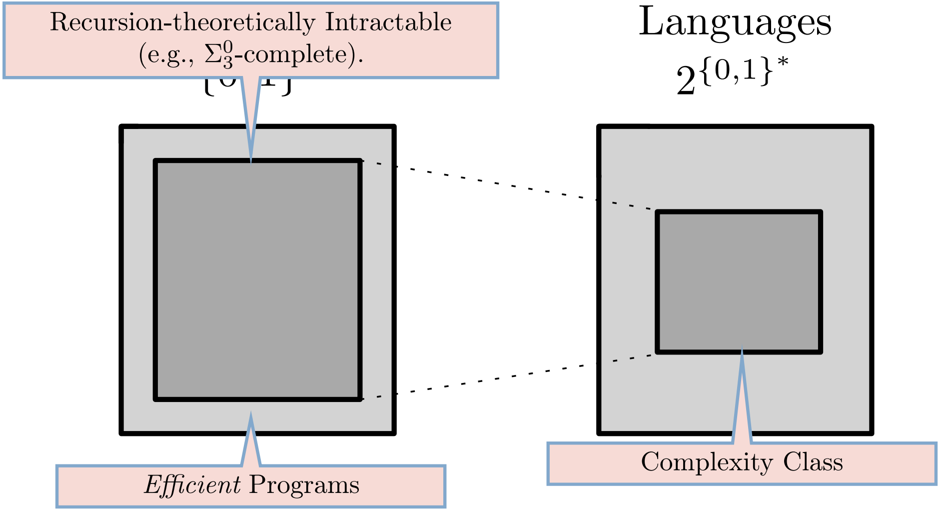
Implicit Computational Complexity



Implicit Computational Complexity



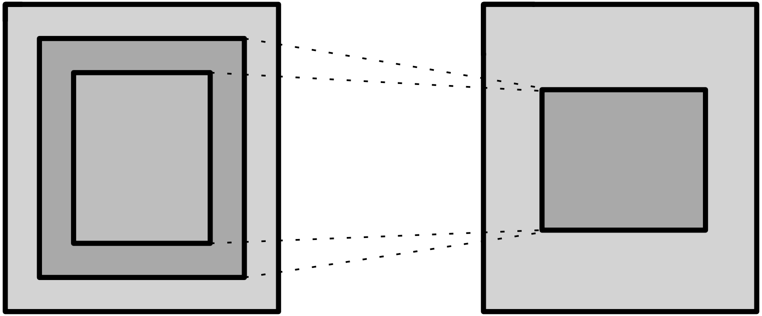
Implicit Computational Complexity



Implicit Computational Complexity

Programs
$\{0,1\}^*$

Languages
$2^{\{0,1\}^*}$

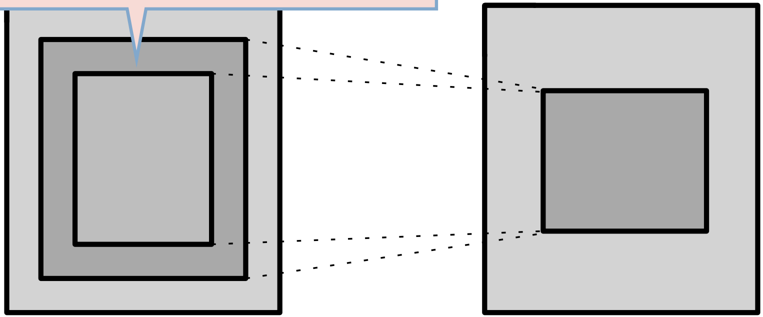


Implicit Computational Complexity

Recursion-theoretically Tractable

Examples: bounded arithmetic, safe recursion, light logics, path orders, etc.

Languages
$2^{\{0,1\}^*}$



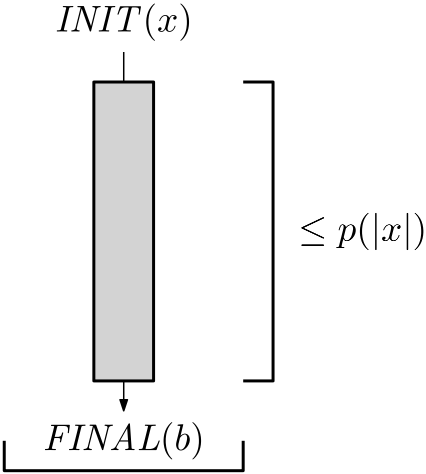
So What?

- **Pave the Way Towards Formal Methods**
  - Once a class is characterized by a collection of programs which is itself easy to decide, a sound (but not complete!) formal method for complexity analysis is available.
  - Since the class is *characterized*, every feasible language is captured by *a* program.
  - Of course, nothing prevents the caught algorithms to be highly *unnatural*.
  - This path has been followed many times in the past, e.g. in amortized analysis and polynomial interpretations.

So What?

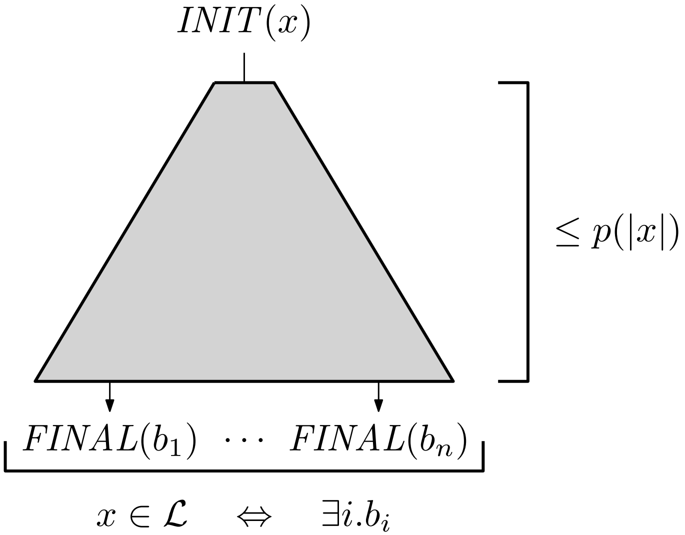
- **Pave the Way Towards Formal Methods**
  - Once a class is characterized by a collection of programs which is itself easy to decide, a sound (but not complete!) formal method for complexity analysis is available.
  - Since the class is *characterized*, every feasible language is captured by *a* program.
  - Of course, nothing prevents the caught algorithms to be highly *unnatural*.
  - This path has been followed many times in the past, e.g. in amortized analysis and polynomial interpretations.
- **Complexity Classes from an Unusual Viewpoint**
  - The obtained definition is *intrinsically different* from the usual one.
  - As such, it could shed light on the nature of the characterized classes, most of them still being mysterious objects.

P

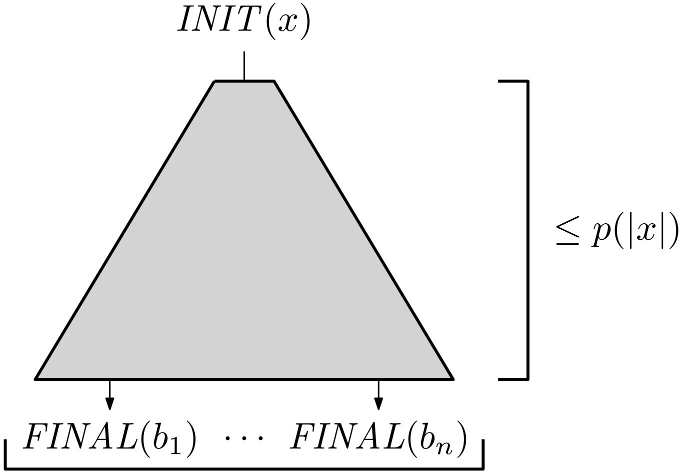


$$x \in \mathcal{L} \quad \Leftrightarrow \quad b$$

NP


$INIT(x)$
$\leq p(|x|)$
$FINAL(b_1) \cdots FINAL(b_n)$
$x \in \mathcal{L} \quad \Leftrightarrow \quad \exists i . b_i$

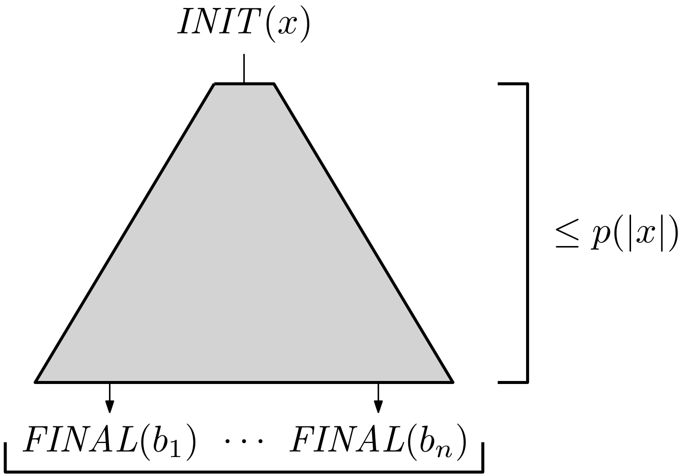
BPP



$$x \in \mathcal{L} \quad \Rightarrow \quad \Pr[b_i] \geq \tfrac{2}{3}$$

$$x \notin \mathcal{L} \quad \Rightarrow \quad \Pr[\neg b_i] \geq \tfrac{2}{3}$$

BPP



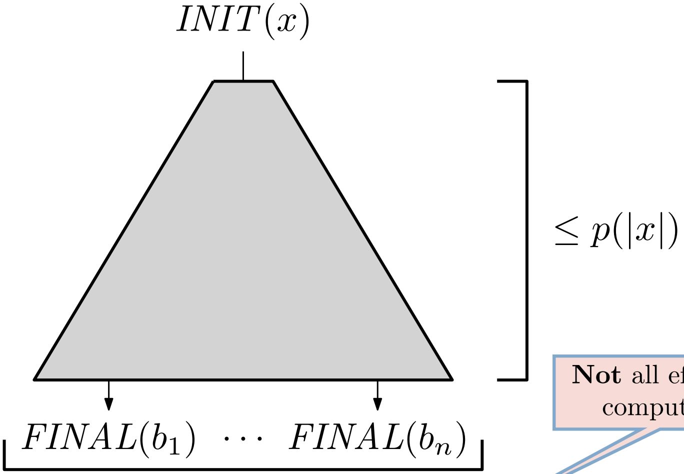
$$x \in \mathcal{L} \quad \Rightarrow \quad \Pr[b_i] \geq \frac{2}{3}$$

$$x \notin \mathcal{L} \quad \Rightarrow \quad \Pr[\neg b_i] \geq \frac{2}{3}$$

Not the same as

$$x \in \mathcal{L} \quad \Leftrightarrow \quad \Pr[b_i] \geq \frac{2}{3}$$

BPP



$$x \in \mathcal{L} \quad \Rightarrow \quad \Pr[b_i] \geq \frac{2}{3}$$

$$x \notin \mathcal{L} \quad \Rightarrow \quad \Pr[\neg b_i] \geq \frac{2}{3}$$

Not all efficient machines compute *a* language!

Not the same as

$$x \in \mathcal{L} \quad \Leftrightarrow \quad \Pr[b_i] \geq \frac{2}{3}$$

Two Kinds of Classes

Syntactic Classes

P **PSPACE** **L** . . .

coNP **NP** **EXP**

Semantic Classes

BPP . . .

ZPP **BQP**

Two Kinds of Classes

Syntactic Classes

P **PSPACE** **L** **coNP** **NP** **EXP** . . .

- Complete problems exist;
- Hierarchy theorems hold.

Semantic Classes

BPP **ZPP** **BQP** . . .

Two Kinds of Classes

Syntactic Classes

P **PSPACE** **L** **coNP** **NP** **EXP** ...

- Complete problems exist;
- Hierarchy theorems hold.

Semantic Classes

BPP **ZPP** **BQP** ...

Nothing is known about complete problems nor about hierarchy theorems.

The Strong Church-Turing Thesis

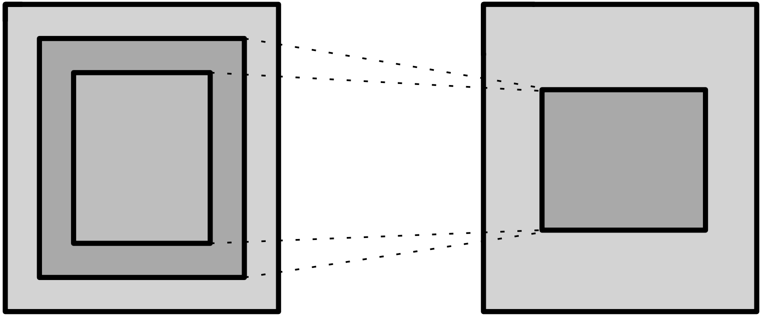
Just as the theory of computability has its foundations in the Church-Turing thesis, computational complexity theory rests upon a modern strengthening of this thesis, which asserts that any "reasonable" model of computation can be efficiently simulated on a probabilistic Turing machine (an efficient simulation is one whose running time is bounded by some polynomial in the running time of the simulated machine). Here, we take reasonable to mean in principle physically realizable.

Ethan Bernstein & Umesh Varirani

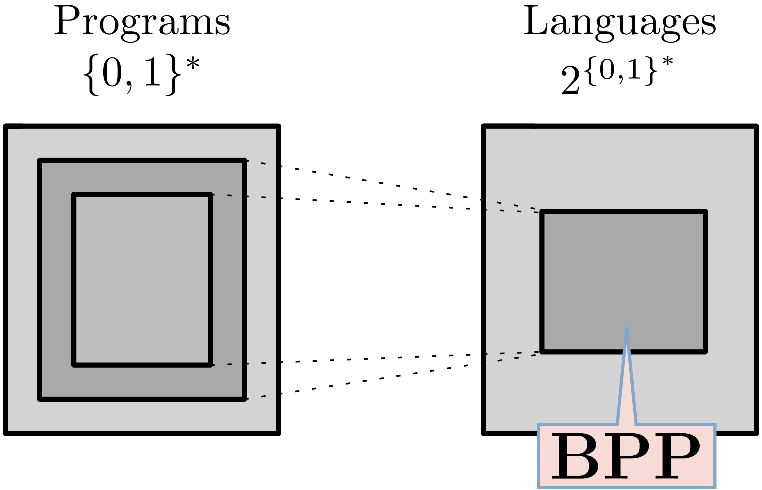
Back to ICC

Programs
$\{0,1\}^*$

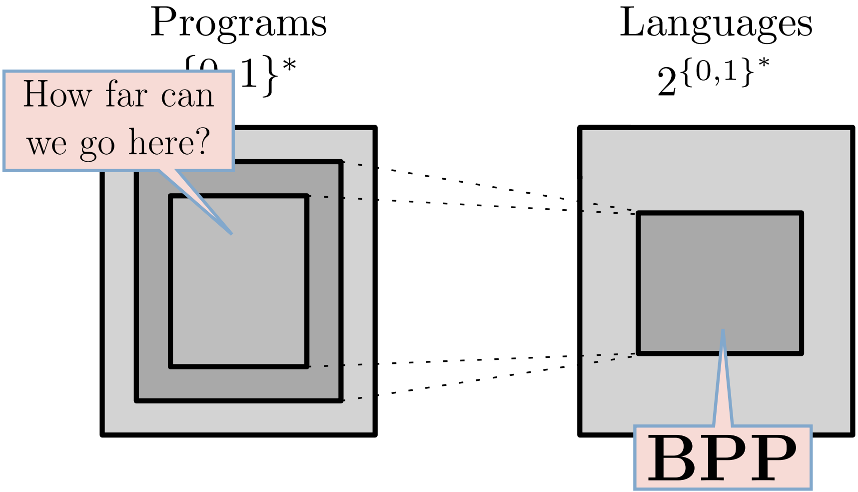
Languages
$2^{\{0,1\}^*}$



Back to ICC



Back to ICC



$$\mathbf{P} \stackrel{?}{=} \mathbf{BPP}$$

Complexity theorists believe this is true...

$$\mathbf{P} \stackrel{?}{=} \mathbf{BPP}$$

Complexity theorists believe this is true. . .

$$\mathbf{P} \stackrel{?}{=} \mathbf{BPP}$$

. . . but the classes' definitions look fundamentally different!

Complexity theorists believe this is true...

$$\mathbf{P} \stackrel{?}{=} \mathbf{BPP}$$

...but the classes' definitions look fundamentally different!

Can we somehow reconcile them?

Part II

Bounded Arithmetic

PA as a Way to Represent Functions

$$\mathsf{PA} \vdash \forall x. \exists ! y. A(x, y)$$

PA as a Way to Represent Functions

- Peano Axioms, reformulated on strings.
- Unrestricted induction.

$$\mathsf{PA} \vdash \forall x. \exists! y. A(x, y)$$

PA as a Way to Represent Functions

- Peano Axioms, reformulated on strings.
- Unrestricted induction.

- First order logic on strings.

$$\mathsf{PA} \vdash \forall x.\exists! y.A(x, y)$$

PA as a Way to Represent Functions

- Peano Axioms, reformulated on strings.
- Unrestricted induction.

- First order logic on strings.

$$\mathsf{PA} \vdash \forall x.\exists! y.A(x,y) \quad \Longrightarrow \quad \begin{array}{c} f : \mathbb{S} \to \mathbb{S} \\ \models A(s, f(s)) \text{ for every } s \in \mathbb{S} \end{array}$$

PA as a Way to Represent Functions

- Peano Axioms, reformulated on strings.
- Unrestricted induction.

- First order logic on strings.

$$\mathsf{PA} \vdash \forall x.\exists! y.A(x,y) \quad \Longrightarrow \quad \begin{array}{c} f : \mathbb{S} \to \mathbb{S} \\ \models A(s, f(s)) \text{ for every } s \in \mathbb{S} \end{array}$$

$$[\![\mathsf{PA}]\!] := \{f : \mathbb{S} \to \mathbb{S} \mid f \text{ is provably total in } \mathsf{PA}\}$$

PA as a Way to Represent Functions

- Peano Axioms, reformulated on strings.
- Unrestricted induction.

- First order logic on strings.

$$\mathsf{PA} \vdash \forall x.\exists ! y.A(x, y) \quad \Longrightarrow \quad \begin{array}{c} f : \mathbb{S} \to \mathbb{S} \\ \models A(s, f(s)) \text{ for every } s \in \mathbb{S} \end{array}$$

$$[\![\mathsf{PA}]\!] := \{f : \mathbb{S} \to \mathbb{S} \mid f \text{ is provably total in } \mathsf{PA}\}$$

Too large for our purposes!

Restricting PA

$$A(0) \to \forall x.(A(x) \to A(x+1)) \to \forall x.A(x)$$

Restricting PA

- The number of times the induction hypothesis is applied when proving $A(n)$ is $\Theta(n)$
- This, in turn, is exponential in $|n|$.
- Better switch to recursion *on notation*

$$A(\varepsilon) \to \forall x.(A(x) \to A(x0) \wedge A(x1)) \to \forall x.A(x)$$

$$A(0) \to \forall x.(A(x) \to A(x+1)) \to \forall x.A(x)$$

Restricting PA

- The number of times the induction hypothesis is applied when proving $A(n)$ is $\Theta(n)$
- This, in turn, is exponential in $|n|$.
- Better switch to recursion *on notation*

$$A(\varepsilon) \to \forall x.(A(x) \to A(x0) \wedge A(x1)) \to \forall x.A(x)$$

$$A(0) \to \forall x.(A(x) \to A(x+1)) \to \forall x.A(x)$$

- If quantifiers occurs in A, the logic becomes simply too powerful.
- Even a single layer of existential quantifiers is too much.
- Better to consider *bounded* quantification:

$$\forall x \leq p.A \qquad \exists x \leq q.A$$

Characterizing **FP**

$$\llbracket \mathsf{S}^1_2 \rrbracket$$

Characterizing **FP**

$$[\![\mathsf{S}^1_2]\!]$$

- Induction *on notation*.
- Induction formulas are Σ^b_1, namely bounded existential quantifications of sharply bounded formulas.

Characterizing **FP**

- Due to Buss [Buss86].
- Many variations exists.

$$[\![\mathsf{S}^1_2]\!] = \mathbf{FP}$$

Characterizing $\mathbf{FP}$

$$[\![\mathsf{S}^1_2]\!] = \mathbf{FP}$$

$$\begin{tikzcd}
\mathsf{S}^1_2 \arrow[r, bend left] & \mathcal{PR} \arrow[l, bend left] \arrow[r, bend left] & \mathbf{PDTM} \arrow[l, bend left]
\end{tikzcd}$$

Characterizing **FP**

$$[\![\mathsf{S}^1_2]\!] = \mathbf{FP}$$

$$\begin{tikzcd} \mathsf{S}^1_2 \arrow[r, bend left] & \mathcal{PR} \arrow[l, bend left] \arrow[r, bend left] & \mathbf{PDTM} \arrow[l, bend left] \end{tikzcd}$$

Cobham's *Bounded Recursion on Notation*

Characterizing **FP**

$$[\![\mathsf{S}^1_2]\!] = \mathbf{FP}$$

$$\begin{tikzcd}
\mathsf{S}^1_2 \arrow[r, bend left] & \mathcal{PR} \arrow[l, bend left] \arrow[r, bend left] & \mathbf{PDTM} \arrow[l, bend left]
\end{tikzcd}$$

Cobham's *Bounded Recursion on Notation*

Polytime Deterministic TMs.

Characterizing **FP**

- Arguably the most difficult step.
- Can be done in various way, e.g. through cut-elimination process, or by realizability.

$$[\![\mathsf{S}^1_2]\!] = \mathbf{FP}$$

$$\begin{tikzcd}
\mathsf{S}^1_2 \arrow[r, bend left] & \mathcal{PR} \arrow[l, bend left] \arrow[r, bend left] & \mathbf{PDTM} \arrow[l, bend left]
\end{tikzcd}$$

Cobham's *Bounded Recursion on Notation*

Polytime Deterministic TMs.

Part III

Incepting Randomness into Bounded Arithmetic

The Main Idea

$$\begin{array}{c} \mathsf{PA} \\ \big\downarrow \\ \mathsf{S}^1_2 \end{array}$$

The Main Idea

$$\begin{tikzcd} \mathsf{PA} \arrow[r] \arrow[d] & \mathsf{MQPA} \\ \mathsf{S}^1_2 & \end{tikzcd}$$

The Main Idea

$$\begin{tikzcd} \mathsf{PA} \arrow[r] \arrow[d] & \mathsf{MQPA} \\ \mathsf{S}^1_2 & \end{tikzcd}$$

- A conservative extension of PA capturing randomized computation.
- All computable random functions from $\mathbb{S}$ to distributions over $\mathbb{S}$ can be represented in MQPA.

The Main Idea

$$\begin{tikzcd} \mathsf{PA} \arrow[r] \arrow[d] & \mathsf{MQPA} \arrow[d] \\ \mathsf{S}^1_2 \arrow[r] & ? \end{tikzcd}$$

What Changes?

The Formulas

- We follow the ideas behind MQPA.
- We endow the grammar of formulas with a *unary predicate* Flip, which models the access to an oracle providing fair random bits, and is interpreted as a measurable subset of $2^{\mathbb{S}}$
- The semantics of a closed formula A is not a truth value but the (measurable!) set $[\![A]\!]$ of all the possible interpretations of `Flip` rendering A true. For example:

$$
\begin{aligned}
[\![\mathsf{Flip}(0) \vee \mathsf{Flip}(1)]\!] &= \{\omega \mid \omega(0) = 1 \text{ or } \omega(1) = 1\} \\
[\![\forall x.\mathsf{Flip}(x)]\!] &= \{\omega \mid \omega(s) = 1 \text{ for every } s \in \mathbb{S}\}
\end{aligned}
$$

- MQPA also has counting quantifiers, which we do not need here.

What Changes?

- ## The Formulas
  - We follow the ideas behind MQPA.
  - We endow the grammar of formulas with a *unary predicate* Flip, which models the access to an oracle providing fair random bits, and is interpreted as a measurable subset of $2^{\mathbb{S}}$
  - The semantics of a closed formula A is not a truth value but the (measurable!) set $[\![A]\!]$ of all the possible interpretations of `Flip` rendering A true. For example:

$$\begin{aligned}[\![\mathsf{Flip}(0) \vee \mathsf{Flip}(1)]\!] &= \{\omega \mid \omega(0) = 1 \text{ or } \omega(1) = 1\} \\ [\![\forall x.\mathsf{Flip}(x)]\!] &= \{\omega \mid \omega(s) = 1 \text{ for every } s \in \mathbb{S}\}\end{aligned}$$

  - MQPA also has counting quantifiers, which we do not need here.
- ## The Underlying Theory
  - It stays essentially the same as S^1_2, and we call it RS^1_2.
  - The only difference is the fact that `Flip` can occur in any formula in axiom schemas.

What Changes?

- **The Formulas**
  - We follow the ideas behind MQPA.
  - We endow the grammar of formulas with a *unary predicate* Flip, which models the access to an oracle providing fair random bits, and is interpreted as a measurable subset of $2^{\mathbb{S}}$
  - The semantics of a closed formula A is not a truth value but the (measurable!) set $[\![A]\!]$ of all the possible interpretations of `Flip` rendering A true. For example:

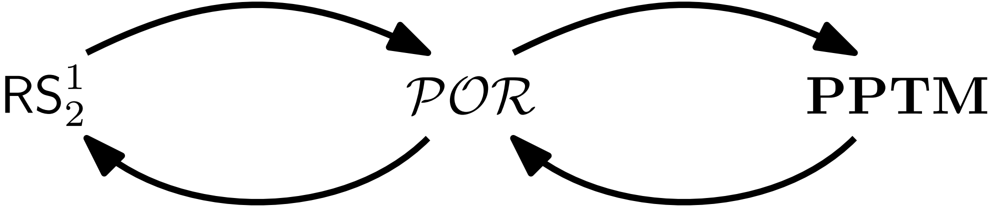
$$
\begin{aligned}
[\![\mathsf{Flip}(0) \vee \mathsf{Flip}(1)]\!] &= \{\omega \mid \omega(0) = 1 \text{ or } \omega(1) = 1\} \\
[\![\forall x.\mathsf{Flip}(x)]\!] &= \{\omega \mid \omega(s) = 1 \text{ for every } s \in \mathbb{S}\}
\end{aligned}
$$

  - MQPA also has counting quantifiers, which we do not need here.
- **The Underlying Theory**
  - It stays essentially the same as S^1_2, and we call it RS^1_2.
  - The only difference is the fact that `Flip` can occur in any formula in axiom schemas.
- **The Target Computational Model**
  - It is natural to just switch to *probabilistic* TMs.
  - We see them as computing functions from $\mathbb{S}$ to *distributions* over $\mathbb{S}$.
  - We ask that *indipendently* on randomness, the runtime of the machine at hand is polynomially bounded.

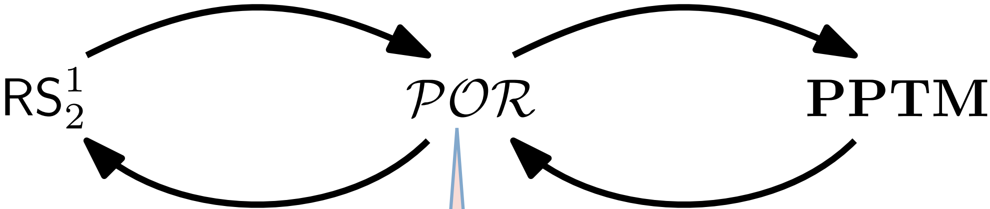
The Result

$$[\![\mathsf{RS}^1_2]\!] \;=\; \{f : \mathbb{S} \to \mathbb{D}(\mathbb{S}) \mid f \text{ can be computed by a } \mathbf{PPTM}\}$$

The Proof



The Proof



- Obtained by extending $\mathcal{PR}$ with a basic function accessing the random oracle.
- Generates functions from $\mathbb{S} \times 2^{\mathbb{S}}$ to $\mathbb{S}$.

The Proof

- Based on “randomized” realizability.
- Closely follows [CookUrquhart1990].

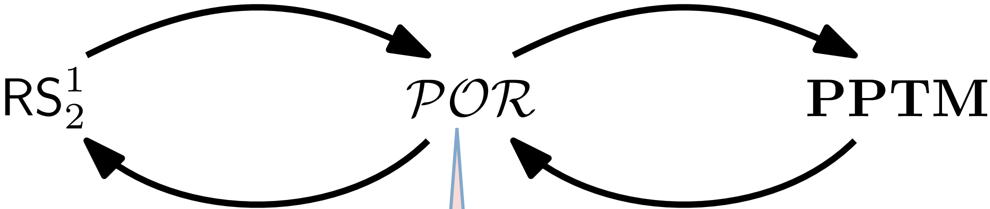
$$\begin{tikzcd} \mathsf{RS}^1_2 \arrow[rr, bend left] & & \mathcal{POR} \arrow[ll, bend left] \arrow[rr, bend left] & & \mathbf{PPTM} \arrow[ll, bend left] \end{tikzcd}$$

- Obtained by extending $\mathcal{PR}$ with a basic function accessing the random oracle.
- Generates functions from $\mathbb{S} \times 2^{\mathbb{S}}$ to $\mathbb{S}$.

The Proof

- Based on "randomized" realizability.
- Closely follows [CookUrquhart1990].

- $\mathcal{POR}$ captures functions in $\mathbb{S}^{\mathbb{S}\times 2^{\mathbb{S}}}$;
- **PPTM** rather captures functions in $\mathbb{S}^{\mathbb{S}\times 2^{\mathbb{N}}}$.



- Obtained by extending $\mathcal{PR}$ with a basic function accessing the random oracle.
- Generates functions from $\mathbb{S} \times 2^{\mathbb{S}}$ to $\mathbb{S}$.

A Particularly Challenging Step

$$\begin{tikzcd} \mathcal{POR} \arrow[rr, bend left] & & \mathbf{PPTM} \end{tikzcd}$$

A Particularly Challenging Step

$$\begin{tikzcd} \mathcal{POR} \arrow[rr, bend left] & & \mathbf{PPTM} \end{tikzcd}$$

- How can we even define the *correctness* of an encoding of a function $f \in \mathcal{POR}$ as a **PPTM** machine $\langle f \rangle$?
  - Their types are different!

A Particularly Challenging Step

$$\mathcal{POR} \longrightarrow \mathbf{PPTM}$$

- How can we even define the *correctness* of an encoding of a function $f \in \mathcal{POR}$ as a **PPTM** machine $\langle f \rangle$?
  - Their types are different!
- The machine $\langle f \rangle$ simulates f only *modulo* measures:

$$\forall x, y \in \mathbb{S}.\ \mu(\{\omega \mid f(x, \omega) = y\}) = \mu(\{\eta \mid \langle f \rangle(x, \eta) = y\})$$

A Particularly Challenging Step

$$\mathcal{POR} \longrightarrow \textbf{PPTM}$$

- How can we even define the *correctness* of an encoding of a function $f \in \mathcal{POR}$ as a **PPTM** machine $\langle f \rangle$?
  - Their types are different!
- The machine $\langle f \rangle$ simulates f only *modulo* measures:

$$\forall x, y \in \mathbb{S}.\ \mu(\{\omega \mid f(x, \omega) = y\}) = \mu(\{\eta \mid \langle f \rangle(x, \eta) = y\})$$

- The way f accesses ω is…random, while $\langle f \rangle$ makes use of η linearly, in an on demand fashion.

A Particularly Challenging Step

$$\begin{tikzcd} \mathcal{POR} \arrow[r, bend left] & \mathbf{PPTM} \end{tikzcd}$$

- How can we even define the *correctness* of an encoding of a function $f \in \mathcal{POR}$ as a **PPTM** machine $\langle f \rangle$?
  - Their types are different!
- The machine $\langle f \rangle$ simulates f only *modulo* measures:

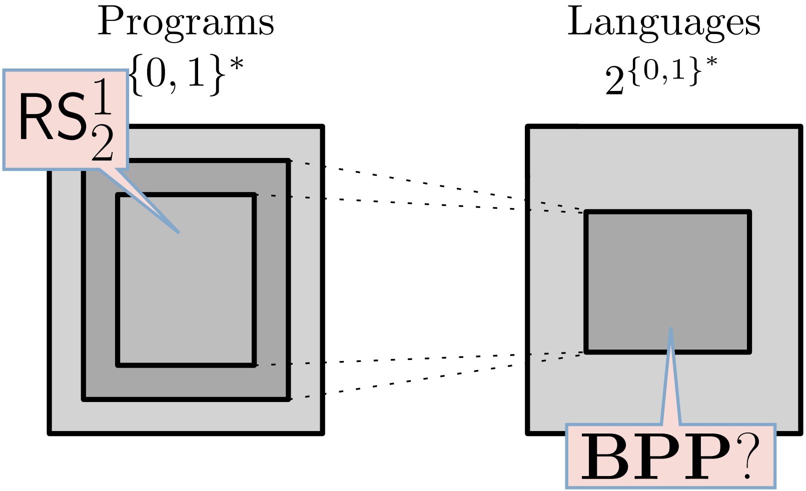
$$\forall x, y \in \mathbb{S}.\ \mu(\{\omega \mid f(x, \omega) = y\}) = \mu(\{\eta \mid \langle f \rangle(x, \eta) = y\})$$

- The way f accesses ω is...random, while $\langle f \rangle$ makes use of η linearly, in an on demand fashion.
- It is convenient to see the encoding as going through *several intermediate steps*, each of them being conceptually very simple.

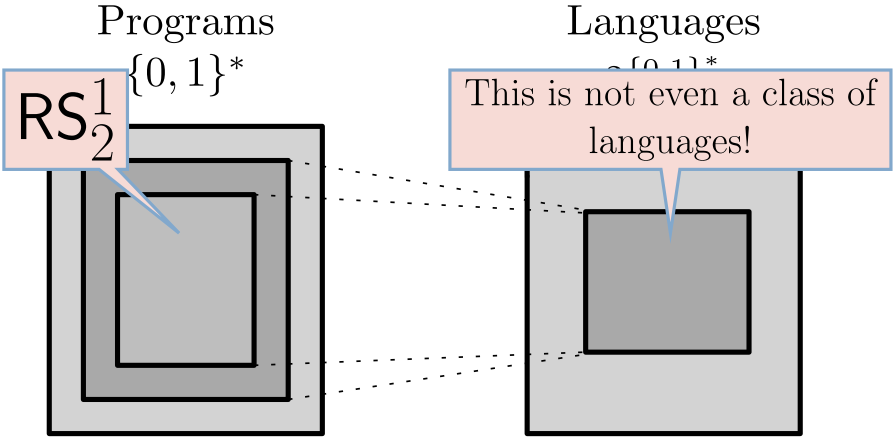
Part IV

Towards **BPP**

Are We There, Yet?



Are We There, Yet? Actually, No!



BPP Through Counting Quantifiers

From...

$$f : \mathbb{S} \to \mathbb{D}(\mathbb{S}) \in [\![\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ f = RandomFunction(A) \end{array}$$

BPP Through Counting Quantifiers

From. . .

$$f : \mathbb{S} \to \mathbb{D}(\mathbb{S}) \in [\![\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ f = RandomFunction(A) \end{array}$$

. . . To

$$(L \subseteq \mathbb{S}) \in [\![\mathbf{C}\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = Language(A) \end{array}$$

BPP Through Counting Quantifiers

From...

$$f : \mathbb{S} \to \mathbb{D}(\mathbb{S}) \in [\![\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ f = \mathit{RandomFunction}(A) \end{array}$$

Counting Quantifier

$$[\![\mathbf{C}^{\frac{t}{s}} B]\!] = \begin{cases} 2^{\mathbb{S}} & \text{if } \mu[\![B]\!] \geq \frac{[\![t]\!]}{[\![s]\!]} \\ \emptyset & \text{otherwise} \end{cases}$$

$$(L \subseteq \mathbb{S}) \in [\![\mathsf{CRS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}_2 \vdash \ldots \exists! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = \mathit{Language}(A) \end{array}$$

$\mathbf{BPP}$ Through Counting Quantifiers

From. . .

$$f : \mathbb{S} \to \mathbb{D}(\mathbb{S}) \in [\![\mathsf{RS}_2^1]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}_2^1 \vdash \forall x.\exists! y.A(x,y) \\ f = \mathit{RandomFunction}(A) \end{array}$$

. . . To

$$(L \subseteq \mathbb{S}) \in [\![\mathbf{C}\mathsf{RS}_2^1]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}_2^1 \vdash \forall x.\exists! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = \mathit{Language}(A) \end{array}$$

Theorem

$$[\![\mathbf{C}\mathsf{RS}_2^1]\!] = \mathbf{BPP}$$

Getting Rid of Counting Quantification

From. . .

$$(L \subseteq \mathbb{S}) \in [\![\mathbf{C}\mathsf{RS}_2^1]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}_2^1 \vdash \forall x.\exists! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = Language(A) \end{array}$$

Getting Rid of Counting Quantification

From...

$$(L \subseteq \mathbb{S}) \in [\![\mathsf{CRS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = \mathit{Language}(A) \end{array}$$

... To

$$(L \subseteq \mathbb{S}) \in [\![\mathsf{T} \oplus \mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y) \\ \mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[A](x,y) \\ L = \mathit{Language}(A) \end{array}$$

Getting Rid of Counting Quantification

From...

$$\mathsf{RS}^1_2 \vdash \forall x.\exists! y.A(x,y)$$
$$(L \subseteq \mathbb{S}) \in [\![\mathbf{C}\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y)$$

- The formula A, once the value of its parameters are fixed, only depends on finitely many coordinates of `Flip`
- We can then internalize the Error Bound inside the language of plain arithmetic, making `Flip` to disappear.
- This go via threshold quantifiers.

$$(L \subseteq \mathbb{S}) \in [\![\mathsf{T} \oplus \mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[A](x,y)$$
$$L = Language(A)$$

Getting Rid of Counting Quantification

From. . .

$$(L \subseteq \mathbb{S}) \in [\![\mathbf{C}\mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists ! y.A(x,y) \\ \models \forall x.\exists y.\mathbf{C}^{\frac{2}{3}} A(x,y) \\ L = Language(A) \end{array}$$

. . . To

$$(L \subseteq \mathbb{S}) \in [\![\mathsf{T} \oplus \mathsf{RS}^1_2]\!] \quad \Leftrightarrow \quad \begin{array}{c} \mathsf{RS}^1_2 \vdash \forall x.\exists ! y.A(x,y) \\ \mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[A](x,y) \\ L = Language(A) \end{array}$$

Theorem

$$[\![\mathsf{T} \oplus \mathsf{RS}^1_2]\!] \subseteq \mathbf{BPP}$$

$\texttt{PIT} \in [\![\mathsf{PA} \oplus \mathsf{RS}^1_2]\!]$

- **Polynomial Identity Testing**, `PIT`.
  - Given two arithmetical circuits, test whether they compute the same polynomial.
  - It admits a naïve decision procedure working in *deterministic exponential time*.
  - Randomization allows the time complexity to be trimmed down to *polynomial*, i.e., `PIT` is well known to be in **BPP**.
  - It is not known whether `PIT` is in **P**.

$\texttt{PIT} \in [\![\mathsf{PA} \oplus \mathsf{RS}^1_2]\!]$

- **Polynomial Identity Testing**, `PIT`.
  - Given two arithmetical circuits, test whether they compute the same polynomial.
  - It admits a naïve decision procedure working in *deterministic exponential time*.
  - Randomization allows the time complexity to be trimmed down to *polynomial*, i.e., `PIT` is well known to be in **BPP**.
  - It is not known whether `PIT` is in **P**.
- **Capturing `PIT` from within Arithmetic**

$$\text{Randomized Algorithm for } \texttt{PIT} \quad \Longrightarrow \quad PIT(x, y)$$

$\mathtt{PIT} \in [\![\mathsf{PA} \oplus \mathsf{RS}^1_2]\!]$

- **Polynomial Identity Testing**, `PIT`.
  - Given two arithmetical circuits, test whether they compute the same polynomial.
  - It admits a naïve decision procedure working in *deterministic exponential time*.
  - Randomization allows the time complexity to be trimmed down to *polynomial*, i.e., `PIT` is well known to be in **BPP**.
  - It is not known whether `PIT` is in **P**.
- **Capturing `PIT` from within Arithmetic**

$$\text{Randomized Algorithm for } \mathtt{PIT} \quad \Longrightarrow \quad PIT(x, y)$$

$$\mathsf{RS}^1_2 \vdash \forall x.\exists! y.PIT(x, y) \qquad\qquad \mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[PIT](x, y)$$

Wrapping Up

- ICC and bounded arithmetic can be seen as ways to *enumerate* complexity classes.
- Semantic classes like **BPP** are not known to be enumerable, due to the *error bound* intrinsic in their definitions.
- We can however enumerate *subclasses* of **BPP** by *internalizing* the error bound check into arithmetic itself!
- Is it that $[\![\mathsf{PA} \oplus \mathsf{RS}^1_2]\!]$ equals **BPP**?

Wrapping Up

- ICC and bounded arithmetic can be seen as ways to *enumerate* complexity classes.
- Semantic classes like **BPP** are not known to be enumerable, due to the *error bound* intrinsic in their definitions.
- We can however enumerate *subclasses* of **BPP** by *internalizing* the error bound check into arithmetic itself!
- Is it that $[\![\mathsf{PA} \oplus \mathsf{RS}^1_2]\!]$ equals **BPP**?

Thank you! Questions?