TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 22: Probabilistic Complexity Classes (1)

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 14 Jan 2025

More recent versions of this slide deck might be available.
For the most current version of this course, see
https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en

Review: PP and BPP

Definition 21.4: A language **L** is in Polynomial Probabilistic Time (PP) if there is a PTM $\mathcal{M}$ such that:

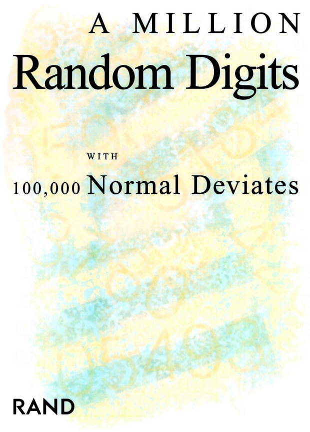
- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{2}$.

Definition 21.11: A language **L** is in Bounded-Error Polynomial Probabilistic Time (BPP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \geq \frac{2}{3}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{3}$.

Random numbers as witness strings

We can replace the built-in true random number generator by a sufficiently long string of random numbers provided in addition to the input.



Rand Corporation, `https://www.rand.org/pubs/monograph_reports/MR1418.html`

A word of warning on BPP

We gave two equivalent definitions for BPP:

(1) Polynomial-time bounded PTMs with probability $\leq \frac{1}{3}$ of returning a wrong result

(2) Polynomial-time bounded DTMs that receive an additional input of random numbers $r \in \{0, 1\}^m$ of uniform length m, polynomial in $|w|$; and producing the correct result for at least $\frac{2}{3}$ such random strings

A word of warning on BPP

We gave two equivalent definitions for BPP:

(1) Polynomial-time bounded PTMs with probability $\leq \frac{1}{3}$ of returning a wrong result

(2) Polynomial-time bounded DTMs that receive an additional input of random numbers $r \in \{0, 1\}^m$ of uniform length m, polynomial in $|w|$; and producing the correct result for at least $\frac{2}{3}$ such random strings

Note that we are not just counting the correct computation paths in either case:

- In Case (1), we sum up probabilities of correct runs; a short path has a higher probability than a single long path
- In Case (2), we count witness strings, but several witness strings might belong to the same path (if it is shorter and does not use all random numbers); thus paths have different numbers of witnesses to account for their different weight

A word of warning on BPP

We gave two equivalent definitions for BPP:

(1) Polynomial-time bounded PTMs with probability $\leq \frac{1}{3}$ of returning a wrong result

(2) Polynomial-time bounded DTMs that receive an additional input of random numbers $r \in \{0, 1\}^m$ of uniform length m, polynomial in $|w|$; and producing the correct result for at least $\frac{2}{3}$ such random strings

Note that we are not just counting the correct computation paths in either case:

- In Case (1), we sum up probabilities of correct runs; a short path has a higher probability than a single long path
- In Case (2), we count witness strings, but several witness strings might belong to the same path (if it is shorter and does not use all random numbers); thus paths have different numbers of witnesses to account for their different weight

Warning: If paths can have different lengths, then requiring that $\geq \frac{2}{3}$ of all computation paths are correct is not the same as requiring that a correct path occurs with probability $\geq \frac{2}{3}$. The former defines a class called $\mathsf{BPP}_{\mathsf{path}}$ instead of BPP. Han, Hemaspaandra, and Thierauf showed that $\mathsf{BPP}_{\mathsf{path}}$ is most likely strictly more powerful than BPP! For example, $\mathsf{NP}^{\mathsf{BPP}} \subseteq \mathsf{BPP}_{\mathsf{path}}$.

PP is hard

We found that:

- $\mathsf{NP} \subseteq \mathsf{PP}$,
- $\mathsf{coNP} \subseteq \mathsf{PP}$, and
- $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$ (without proof)

As an upper bound, we got $\mathsf{PP} \subseteq \mathsf{PSpace}$.

Indeed, the word problem for PP languages seems to require exponential effort to check, since the probability of accepting words in the language may be exponentially close to the probability of accepting words that are not in the language.

$\leadsto$ not a practical class of probabilistic algorithms

Understanding BPP

BPP is practical

We found (Theorem 21.12):

- If a polytime PTM produces the correct output with probability $\geq \frac{1}{2} + |w|^{-c}$,
- then some polytime PTM produces the correct output with probability $\geq 1 - 2^{-|w|^d}$.

In words: even a weak bound on the error is enough to obtain almost arbitrary certainty in polynomial time!

BPP is practical

We found (Theorem 21.12):

- If a polytime PTM produces the correct output with probability $\geq \frac{1}{2} + |w|^{-c}$,
- then some polytime PTM produces the correct output with probability $\geq 1 - 2^{-|w|^d}$.

In words: even a weak bound on the error is enough to obtain almost arbitrary certainty in polynomial time!

Corollary 21.15: Defining the class BPP with any bounded error probability $< \frac{1}{2}$ instead of $\frac{1}{3}$ leads to the same class of languages.

Corollary 21.16: For any language in BPP, there is a polynomial time algorithm with exponentially low probability of error.

BPP is practical

We found (Theorem 21.12):

- If a polytime PTM produces the correct output with probability $\geq \frac{1}{2} + |w|^{-c}$,
- then some polytime PTM produces the correct output with probability $\geq 1 - 2^{-|w|^d}$.

In words: even a weak bound on the error is enough to obtain almost arbitrary certainty in polynomial time!

Corollary 21.15: Defining the class BPP with any bounded error probability $< \frac{1}{2}$ instead of $\frac{1}{3}$ leads to the same class of languages.

Corollary 21.16: For any language in BPP, there is a polynomial time algorithm with exponentially low probability of error.

BPP might be better than P for describing what is "tractable in practice"!

Summary and open questions

We have already seen that BPP is robust against the actual error bound

Moreover, it is not hard to show the following:

- BPP is closed under complement (exercise)
- $\mathsf{BPP}^{\mathsf{BPP}} = \mathsf{BPP}$ (exercise)

We have not discussed several important questions:

- What happens if we assume unfair coins? $(\Pr[\text{heads}] \neq \frac{1}{2})$
- How does BPP relate to other complexity classes?
- Which problems are in BPP and which are BPP-complete?

Robustness using unfair coins (1)

Would a PTM have greater power if its random number generator would output 1 with probability $\rho \neq \frac{1}{2}$?

Proposition 22.1: A coin with $\Pr[\text{heads}] = \rho$ can be simulated by a PTM in expected time $O(1)$ provided that the ith bit of ρ is computable in polynomial time w.r.t. i.

Robustness using unfair coins (1)

Would a PTM have greater power if its random number generator would output 1 with probability $\rho \neq \frac{1}{2}$?

Proposition 22.1: A coin with $\Pr[\text{heads}] = \rho$ can be simulated by a PTM in expected time $O(1)$ provided that the ith bit of ρ is computable in polynomial time w.r.t. i.

Proof: Let $0.\rho_1\rho_2\rho_3\cdots$ be the binary expansion of ρ. Starting with $i = 1$, do:

- Compute a random bit $b_i \in \{0, 1\}$
- If $\rho_i > b_i$, return "heads"
- If $\rho_i < b_i$, return "tails"
- If $\rho_i = b_i$, increment i and repeat procedure.

Robustness using unfair coins (1)

Would a PTM have greater power if its random number generator would output 1 with probability $\rho \neq \frac{1}{2}$?

Proposition 22.1: A coin with $\Pr[\text{heads}] = \rho$ can be simulated by a PTM in expected time $O(1)$ provided that the ith bit of ρ is computable in polynomial time w.r.t. i.

Proof: Let $0.\rho_1\rho_2\rho_3\cdots$ be the binary expansion of ρ. Starting with $i = 1$, do:

- Compute a random bit $b_i \in \{0, 1\}$
- If $\rho_i > b_i$, return "heads"
- If $\rho_i < b_i$, return "tails"
- If $\rho_i = b_i$, increment i and repeat procedure.

Analysis:

- The simulation reaches step $i + 1$ with probability $(\frac{1}{2})^i$
- Combined probability of "heads": $\sum_i \rho_i \frac{1}{2^i} = \rho$
- The expected runtime is $O(\sum_i i^c \frac{1}{2^i})$, where c is a constant degree capturing the polynomial effort of computing ρ_i – this can be shown to be in $O(1)$. $\square$

Robustness using unfair coins (2)

Note: Proposition 22.1 requires ρ to be efficiently computable. Unfair coins with hard to compute probabilities would indeed increase the computational power.

Robustness using unfair coins (2)

Note: Proposition 22.1 requires ρ to be efficiently computable. Unfair coins with hard to compute probabilities would indeed increase the computational power.

Conversely, we may ask if a PTM with unfair coin could simulate a fair coin:

Proposition 22.2: A coin with $\Pr[\text{heads}] = \frac{1}{2}$ can be simulated by a TM that may use a coin with heads-probability ρ in time $O(\frac{1}{\rho(1-\rho)})$.

Proof: See exercise (for the basic technique of simulating fair coins with arbitrary ones)

Note that the previous result does not require ρ to be computable.

Robustness using unfair coins (2)

Note: Proposition 22.1 requires ρ to be efficiently computable. Unfair coins with hard to compute probabilities would indeed increase the computational power.

Conversely, we may ask if a PTM with unfair coin could simulate a fair coin:

Proposition 22.2: A coin with $\Pr[\text{heads}] = \frac{1}{2}$ can be simulated by a TM that may use a coin with heads-probability ρ in time $O(\frac{1}{\rho(1-\rho)})$.

Proof: See exercise (for the basic technique of simulating fair coins with arbitrary ones)

Note that the previous result does not require ρ to be computable.

Conclusion: BPP is rather robust against the use of different coins.

Polynomial Identity Testing

A problem in BPP

We give an example of a problem in BPP that is not known to be in P.

Polynomial Identity Testing (PIT):

- Task: Determine if two polynomial functions are equal, i.e., have the same results on all inputs
- The polynomials can be multivariate (i.e., contain more than two variables)

A problem in BPP

We give an example of a problem in BPP that is not known to be in P.

Polynomial Identity Testing (PIT):

- Task: Determine if two polynomial functions are equal, i.e., have the same results on all inputs
- The polynomials can be multivariate (i.e., contain more than two variables)
- Challenge: The polynomials are not given in their normal form (as a sum of monomials)

A problem in BPP

We give an example of a problem in BPP that is not known to be in P.

Polynomial Identity Testing (PIT):

- Task: Determine if two polynomial functions are equal, i.e., have the same results on all inputs
- The polynomials can be multivariate (i.e., contain more than two variables)
- Challenge: The polynomials are not given in their normal form (as a sum of monomials)

Approach: Reduce the question "$f = g$?" to the question "$f - g = 0$?," i.e., to the question if a given polynomial is equal to zero.

A problem in BPP

We give an example of a problem in BPP that is not known to be in P.

Polynomial Identity Testing (PIT):

- Task: Determine if two polynomial functions are equal, i.e., have the same results on all inputs
- The polynomials can be multivariate (i.e., contain more than two variables)
- Challenge: The polynomials are not given in their normal form (as a sum of monomials)

Approach: Reduce the question "$f = g$?" to the question "$f - g = 0$?," i.e., to the question if a given polynomial is equal to zero.

Example 22.3: We may ask if $(x + y)(x - y)$ equals $x^2 - y^2$. To answer this, we can test if the polynomial function $(x + y)(x - y) - (x^2 - y^2)$ equals zero.

Algebraic circuits and **ZeroP**

The representation we assume for polynomials in PIT are algebraic circuits:

- Algebraic circuits are like Boolean circuits but operate on integer numbers
- Gates perform arithmetic operations $+$, $-$, and $\times$, or have constant output 1
- There is one output

Note: it is easy to express the difference of the functions encoded in two algebraic circuits

ZeroP

Input:	Algebraic circuit C
Problem:	Does C return 0 on all inputs?

How difficult is ZeroP?

Observation:

- Algebraic circuits can encode polynomials very efficiently: a small circuit can express a polynomial that is large when written in the usual form

Example 22.4: It is easy to find a circuit of size $2k$ for $\prod_{i=1}^{k}(x_i + y_i)$ (assuming binary fan-in for multiplication gates), but writing this function as a sum of monomials requires 2^k monomials of the form $z_1 \cdot z_2 \cdots z_k$ where $z_i \in \{x_i, y_i\}$.

- Nevertheless, the output of a circuit is easy to compute

How difficult is **ZeroP**?

Observation:

- Algebraic circuits can encode polynomials very efficiently:
  a small circuit can express a polynomial that is large when written in the usual form

Example 22.4: It is easy to find a circuit of size $2k$ for $\prod_{i=1}^{k}(x_i + y_i)$ (assuming binary fan-in for multiplication gates), but writing this function as a sum of monomials requires 2^k monomials of the form $z_1 \cdot z_2 \cdots z_k$ where $z_i \in \{x_i, y_i\}$.

- Nevertheless, the output of a circuit is easy to compute

Surprisingly (?): There is an efficient probabilistic algorithm for **ZeroP**

How frequently do non-zero polynomials compute zero?

The total degree of a (multivariate) monomial is the sum of the degrees of all of its variables, and the total degree of a polynomial is the maximal degree of its monomials.

The following property is the key to showing **ZeroP** $\in$ BPP:

Lemma 22.5 (Schwartz-Zippel Lemma): Consider a non-zero multivariate polynomial $p(x_1, \ldots, x_m)$ of total degree $\leq d$, and a finite set S of integers. If $a_1, \ldots, a_m$ are chosen randomly (with replacement) from S, then

$$\Pr\left[p(a_1, \ldots, a_m) = 0\right] \leq \frac{d}{|S|}$$

Proof: See Exercise.

A probabilistic algorithm for **ZeroP** (1)

By Schwartz-Zippel, we just need to randomly sample numbers from a large enough set S to find a non-zero value with high probability, namely $1 - \frac{d}{|S|}$.

What is the degree d of a polynomial encoded in an algebraic circuit?

A probabilistic algorithm for **Zero**P (1)

By Schwartz-Zippel, we just need to randomly sample numbers from a large enough set S to find a non-zero value with high probability, namely $1 - \frac{d}{|S|}$.

What is the degree d of a polynomial encoded in an algebraic circuit?

A circuit of size n can compute degrees of at most 2^n.

A probabilistic algorithm for ZEROP (1)

By Schwartz-Zippel, we just need to randomly sample numbers from a large enough set S to find a non-zero value with high probability, namely $1 - \frac{d}{|S|}$.

What is the degree d of a polynomial encoded in an algebraic circuit?

A circuit of size n can compute degrees of at most 2^n.

$\leadsto$ for a set S of size $3 \cdot 2^n$, we expect a non-zero value with probability $\geq 1 - \frac{2^n}{3 \cdot 2^n} = \frac{2}{3}$

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 3 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit to compute $p(a_1, \ldots, a_m)$
- Accept if $p(a_1, \ldots, a_m) = 0$ and reject otherwise.

A probabilistic algorithm for ZeroP (1)

By Schwartz-Zippel, we just need to randomly sample numbers from a large enough set S to find a non-zero value with high probability, namely $1 - \frac{d}{|S|}$.

What is the degree d of a polynomial encoded in an algebraic circuit?

A circuit of size n can compute degrees of at most 2^n.

$\leadsto$ for a set S of size $3 \cdot 2^n$, we expect a non-zero value with probability $\geq 1 - \frac{2^n}{3 \cdot 2^n} = \frac{2}{3}$

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 3 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit to compute $p(a_1, \ldots, a_m)$
- Accept if $p(a_1, \ldots, a_m) = 0$ and reject otherwise.

Analysis: If $p \in$ **ZeroP**, the algorithm will always accept. Otherwise, if $p \notin$ **ZeroP**, it will reject with probability $\geq \frac{2}{3}$.

A probabilistic algorithm for **ZeroP** (2)

Did we show **ZeroP** $\in$ BPP?

A probabilistic algorithm for **ZeroP** (2)

Did we show **ZeroP** $\in$ BPP?

There is a problem with our algorithm:

- We can sample the numbers a_i in polynomial time (polynomial number of bits)
- But if the degree of the polynomial is as high as 2^n, then the output can be as high as $(3 \cdot 2^n)^{2^n}$, requiring $O(2^n)$ bits to store!

A probabilistic algorithm for **ZeroP** (2)

Did we show **ZeroP** $\in$ BPP?

There is a problem with our algorithm:

- We can sample the numbers a_i in polynomial time (polynomial number of bits)
- But if the degree of the polynomial is as high as 2^n, then the output can be as high as $(3 \cdot 2^n)^{2^n}$, requiring $O(2^n)$ bits to store!

One can solve this problem as follows:

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

ZeroP ∈ BPP

Algorithm (with fingerprinting): For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

ZeroP $\in$ BPP

Algorithm (with fingerprinting): For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

Analysis: (additional details in Arora & Barak, Section 7.2.3)

- If $p(a_1, \ldots, a_m) = 0$ then $p(a_1, \ldots, a_m) = 0 \mod k$, so the algorithm surely accepts
- If $p(a_1, \ldots, a_m) \neq 0$ then $p(a_1, \ldots, a_m) \neq 0 \mod k$ if k does not divide $p(a_1, \ldots, a_m)$

ZeroP $\in$ BPP

Algorithm (with fingerprinting): For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

Analysis: (additional details in Arora & Barak, Section 7.2.3)

- If $p(a_1, \ldots, a_m) = 0$ then $p(a_1, \ldots, a_m) = 0 \mod k$, so the algorithm surely accepts
- If $p(a_1, \ldots, a_m) \neq 0$ then $p(a_1, \ldots, a_m) \neq 0 \mod k$ if k does not divide $p(a_1, \ldots, a_m)$
- Claim: the probability of k dividing $p(a_1, \ldots, a_m)$ is $\leq \frac{1}{4n}$. Proof sketch:
  - We can restrict to cases where k (by random chance) is prime: for large n, there are at least $\frac{2^{2n}}{2n}$ prime numbers $\leq 2^{2n}$ (Prime Number Theorem)
  - A number has only logarithmically many prime factors ($O(n \cdot 2^n)$ in our case)
  - One can estimate that k with probability $\geq \frac{1}{4n}$ is both (i) a prime number and (ii) not among the prime factors of $p(a_1, \ldots, a_m)$ $\square$

Note: This does not yield a probability of error $\leq \frac{1}{3}$, but error probability $\leq \frac{1}{10} + \frac{9}{10}(1 - \frac{1}{4n})^{4n} \leq \frac{1}{10} + \frac{9}{10}\frac{1}{e} \leq 0.44$, which suffices.

Further problems in BPP

Other algorithms in BPP include:

- Testing for perfect matching in a bipartite graph

  Informally: checking whether every member of two equal-sized populations of heterosexual men and women can engage in monogamous partnerships according to their expressed preferences.

  - Can be reduced to checking if a variable matrix has non-zero determinant
  - Similar to **ZeroP**, one can use Schwartz-Zippel here
- Primality testing (**Primes**)
  - A classical probabilistic algorithm discovered in the 1970s
  - In 2002, Agrawal, Kayal, and Saxena found a deterministic polynomial algorithm
- “Monte-Carlo algorithms”
  - These are a general class of algorithms with “probably correct” output
  - BPP contains polynomial-time Monte-Carlo algorithms

BPP-complete problems

We can define hardness for BPP with respect to polynomial many-one reductions.

BPP-complete problems

We can define hardness for BPP with respect to polynomial many-one reductions.

However, surprisingly, no problem is known to be BPP-complete.

Why can't we get BPP-complete problems from the word problem of BPP Turing machines?

- Accept tuples of the form $\langle \mathcal{M}, w, 1^n \rangle$, where
- $\mathcal{M}$ is a PTM, w a word, and 1^n is a number in unary encoding,
- provided that the probability that $\mathcal{M}$ accepts w in n steps is $\geq \frac{2}{3}$

BPP-complete problems

We can define hardness for BPP with respect to polynomial many-one reductions.

However, surprisingly, no problem is known to be BPP-complete.

Why can't we get BPP-complete problems from the word problem of BPP Turing machines?

- Accept tuples of the form $\langle \mathcal{M}, w, 1^n \rangle$, where
- $\mathcal{M}$ is a PTM, w a word, and 1^n is a number in unary encoding,
- provided that the probability that $\mathcal{M}$ accepts w in n steps is $\geq \frac{2}{3}$

Because we do not know if this problem is in BPP!

- It is unclear if an algorithm exists that rejects words not in this language with probability $\geq \frac{2}{3}$
- In particular, $\mathcal{M}$ might not satisfy the BPP-conditions for accepting any language – the input is not really a BPP word problem in this case!

Semantic vs. syntactic classes

A better definition of the BPP word problem might be:

- Accept tuples of the form $\langle \mathcal{M}, w, 1^n \rangle$, where
- $\mathcal{M}$ is a PTM, w a word, and 1^n is a number in unary encoding, if
- (1) for all inputs v, the probability of $\mathcal{M}$ to accept v in $p(|v|)$ steps is either $\geq \frac{2}{3}$ or $\leq \frac{1}{3}$,
- (2) the probability that $\mathcal{M}$ accepts w in $p(|w|)$ steps is $\geq \frac{2}{3}$

Semantic vs. syntactic classes

A better definition of the BPP word problem might be:

- Accept tuples of the form $\langle \mathcal{M}, w, 1^n \rangle$, where
- $\mathcal{M}$ is a PTM, w a word, and 1^n is a number in unary encoding, if
- (1) for all inputs v, the probability of $\mathcal{M}$ to accept v in $p(|v|)$ steps is either $\geq \frac{2}{3}$ or $\leq \frac{1}{3}$,
- (2) the probability that $\mathcal{M}$ accepts w in $p(|w|)$ steps is $\geq \frac{2}{3}$

Unfortunately, that's undecidable:

It is undecidable if a PTM $\mathcal{M}$ accepts any language with the BPP-conditions (1)

- The BPP acceptance conditions are "semantic" conditions, and some PTMs do not satisfy these conditions for any language
- In contrast, e.g., every NTM accepts some language; and we can effectively enumerate polytime NTMs for all languages in NP ("syntactic" condition)

Summary and Outlook

BPP provides a robust notion of practical probabilistic algorithm

Polynomial identity testing is in BPP (and not know to be in P)

BPP is different from many other classes in that it has a "semantic" definition based on the behaviour rather than merely the syntax of TMs

What's next?

- More relationships to more (probabilistic) classes
- Quantum Computing
- Summary