Computational Complexity Theory

Lecture 1: Intro; Turing machines

Department of Computer Science,
Indian Institute of Science

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

a. Decision problem

Example: Is vertex t reachable from vertex s in graph G?

Is n a prime number?

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

  a. Decision problem

  b. Search problem

Example: Find a satisfying assignment for a Boolean formula.

Find a prime between n and 2n.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

  a. Decision problem

  b. Search problem

  c. Counting problem

Example: Count the number of cycles in a graph.

Count the number of perfect matchings in a graph.

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **problems** come in various flavors:

  a. Decision problem

  b. Search problem

  c. Counting problem

  d. Optimization problem

Example: Find a minimum size vertex cover in a graph

About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- **Algorithms** are methods for solving problems; they are studied using formal models of computation, like Turing machines.

↓

- a memory with head (like a RAM)
- a finite control (like a processor)

(…more later in this lecture)

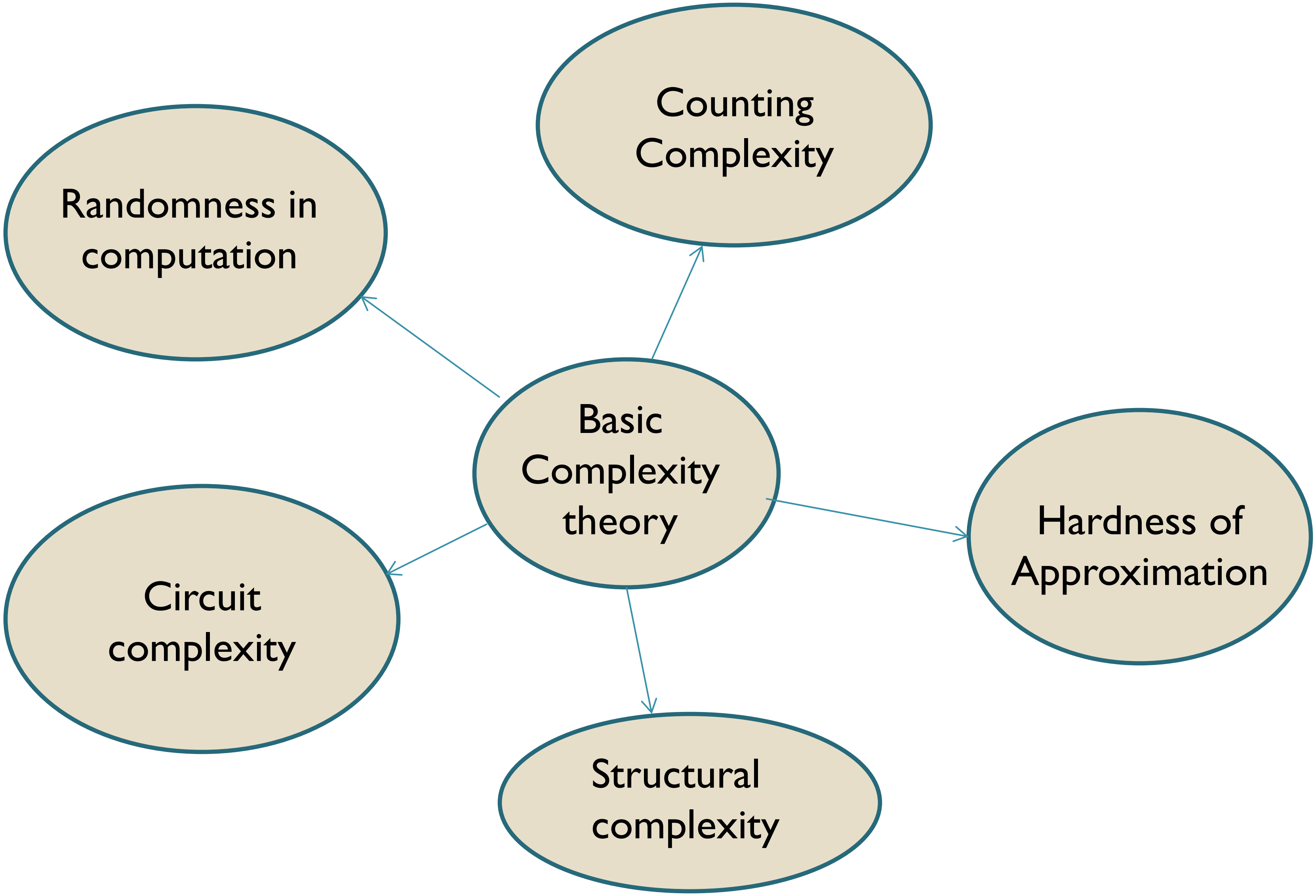
About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **resources** (required by models of computation) can be:
  - Time (bit operations)
  - Space (memory cells)

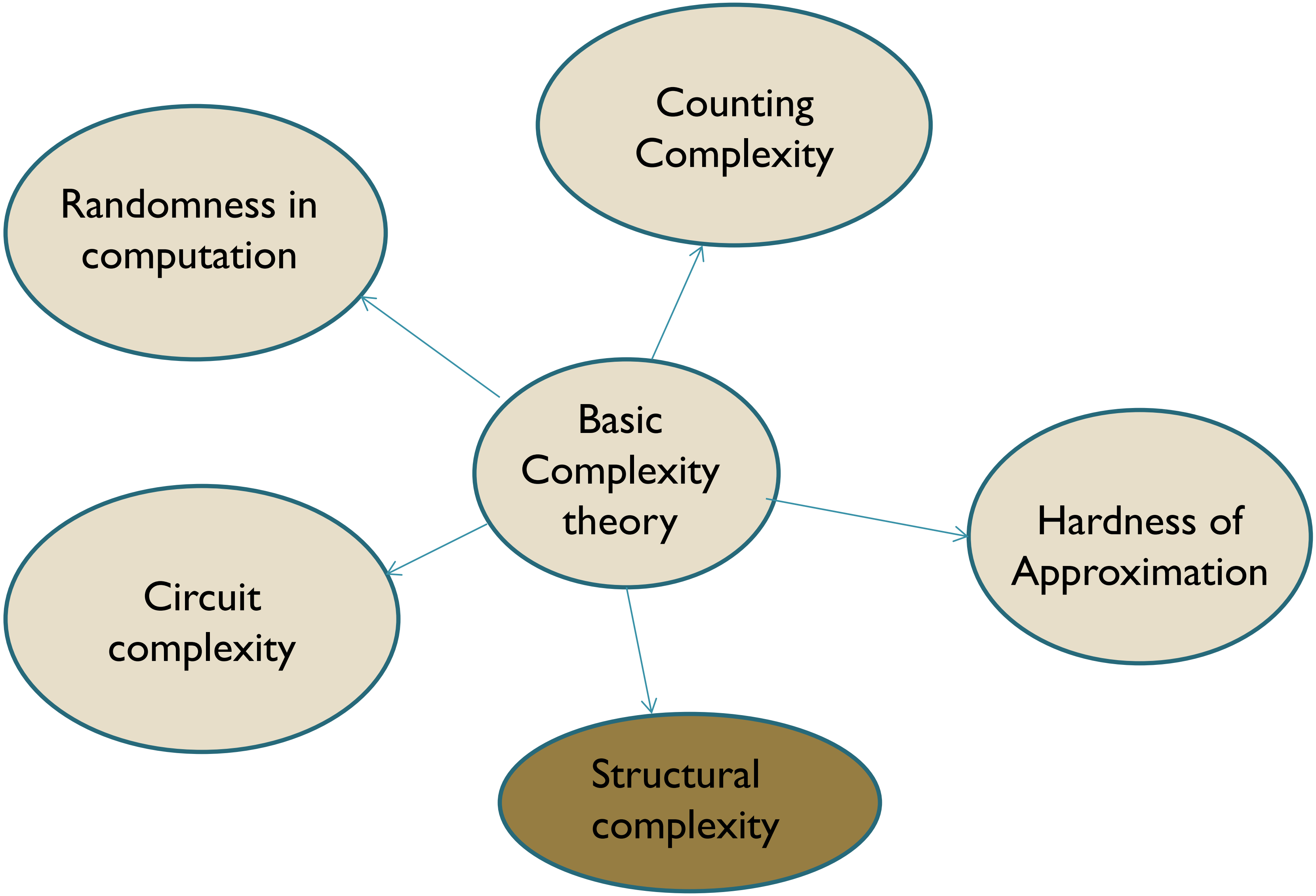
About the course

- Computational complexity attempts to classify computational problems based on the amount of resources required by algorithms to solve them.
- Computational **resources** (required by models of computation) can be:
  - Time (bit operations)
  - Space (memory cells)
  - Random bits (magic bits: 0 w. p ½ and 1 w.p ½)
  - Communication (bit exchanges)

Topics to be covered in this course



Topics to be covered in this course



Counting Complexity

Randomness in computation

Basic Complexity theory

Hardness of Approximation

Circuit complexity

Structural complexity

Structural Complexity

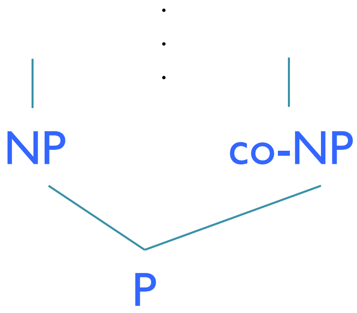
- Classes P, NP, co-NP… NP-completeness.
  - How hard is it to check satisfiability of a Boolean formula?
  - What if the formula has exactly one or no satisfying assignment?

Structural Complexity

- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
    - How much space is required to check s-t connectivity?

Structural Complexity

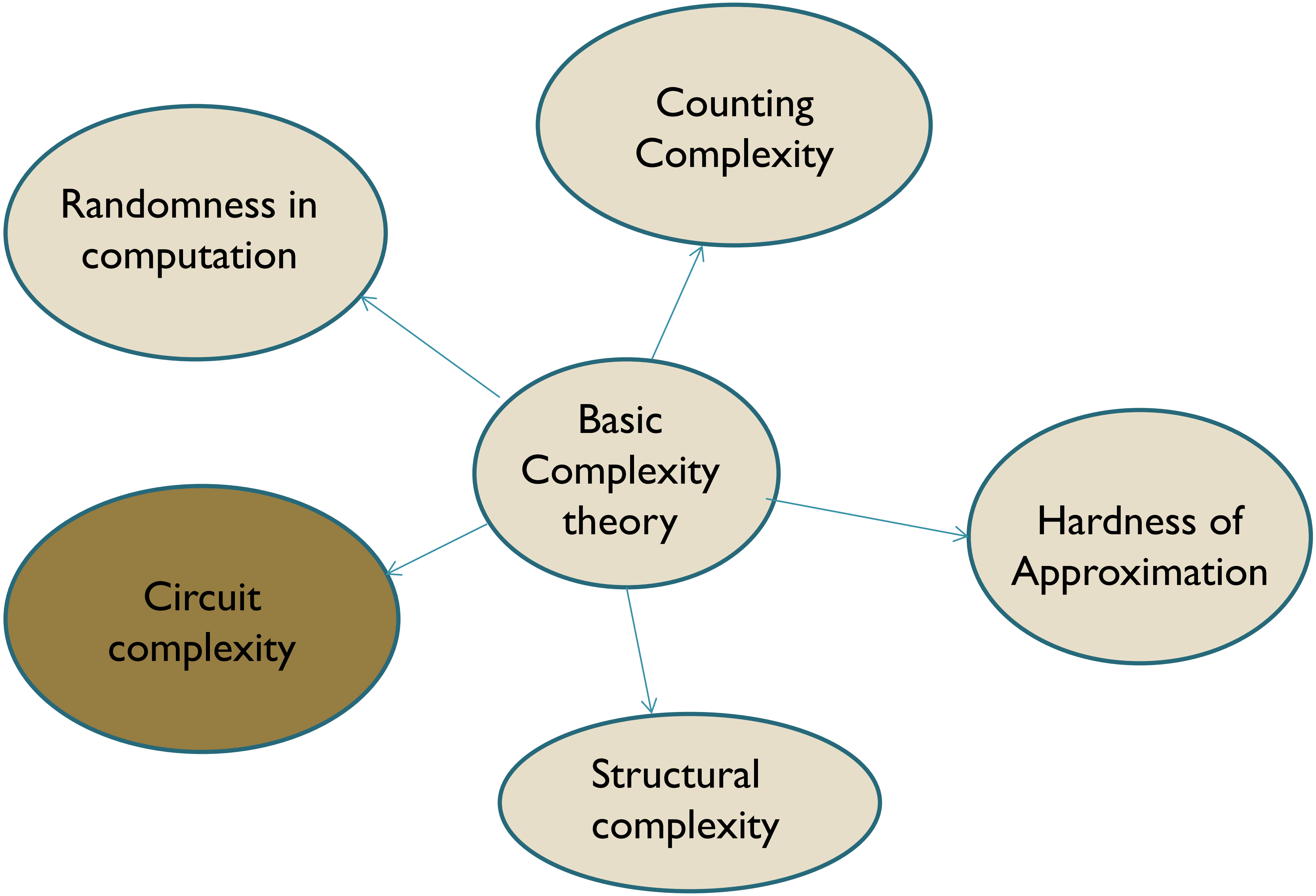
- Classes P, NP, co-NP… NP-completeness.
- Space bounded computation.
- Polynomial Hierarchy (PH).



- How hard is it to check if the largest independent set in G has size k ?

- How hard is it to check if there is a circuit of size k that computes the same Boolean function as a given Boolean circuit C ?

Topics to be covered in this course



Basic Complexity theory

Counting Complexity

Randomness in computation

Hardness of Approximation

Circuit complexity

Structural complexity

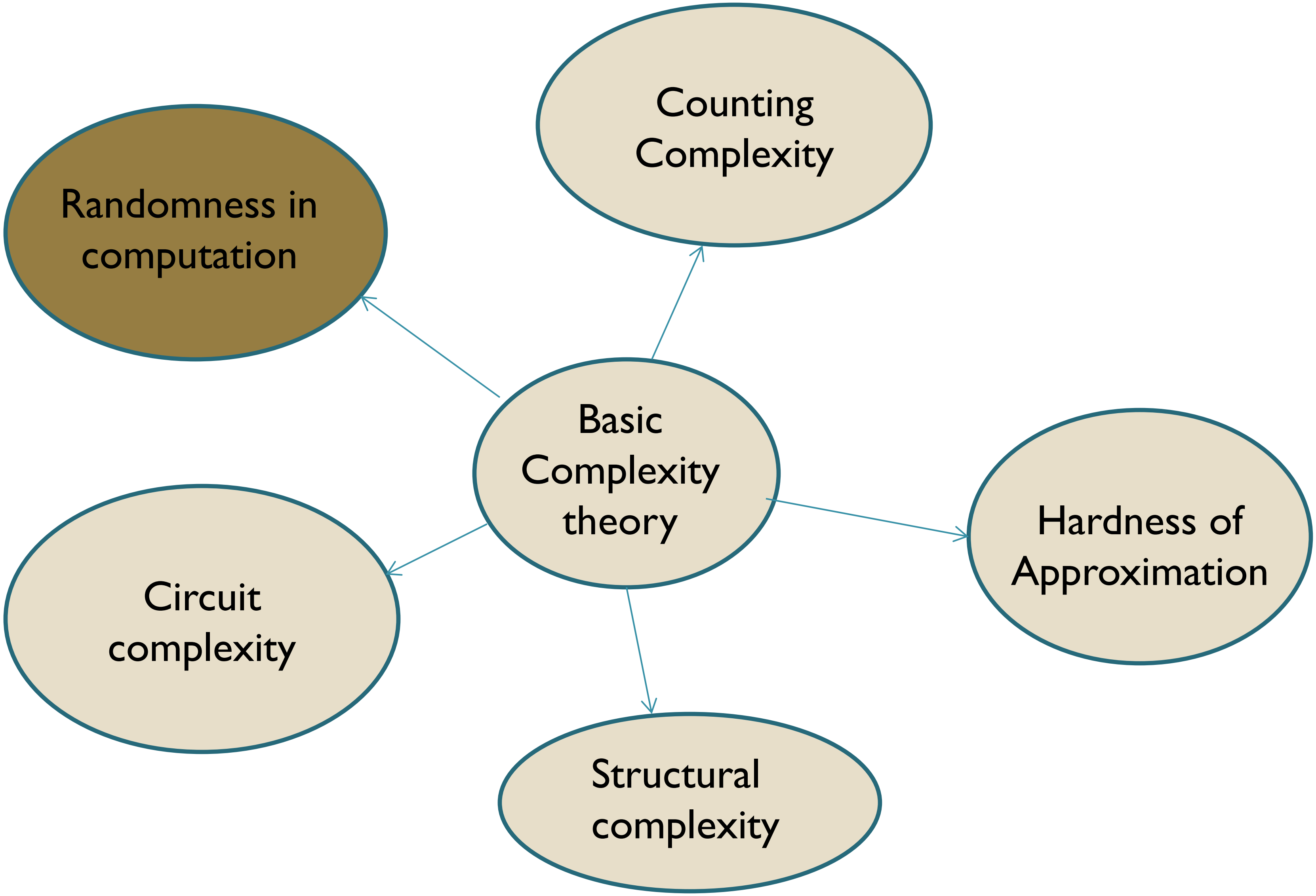
Circuit Complexity

- The internal workings of an algorithm can be viewed as a Boolean circuit -- a nice combinatorial model of computation that is closely related to Turing Machines.
- The size, depth & width of a circuit correspond to the sequential, parallel & space complexity, respectively, of the algorithm that it represents.

Circuit Complexity

- The internal workings of an algorithm can be viewed as a Boolean circuit -- a nice combinatorial model of computation that is closely related to Turing Machines.
- The size, depth & width of a circuit correspond to the sequential, parallel & space complexity, respectively, of the algorithm that it represents.
- Proving $\mathrm{P} \neq \mathrm{NP}$ reduces to showing circuit lower bounds.
    - We will see lower bounds for restricted classes of circuits.

Topics to be covered in this course



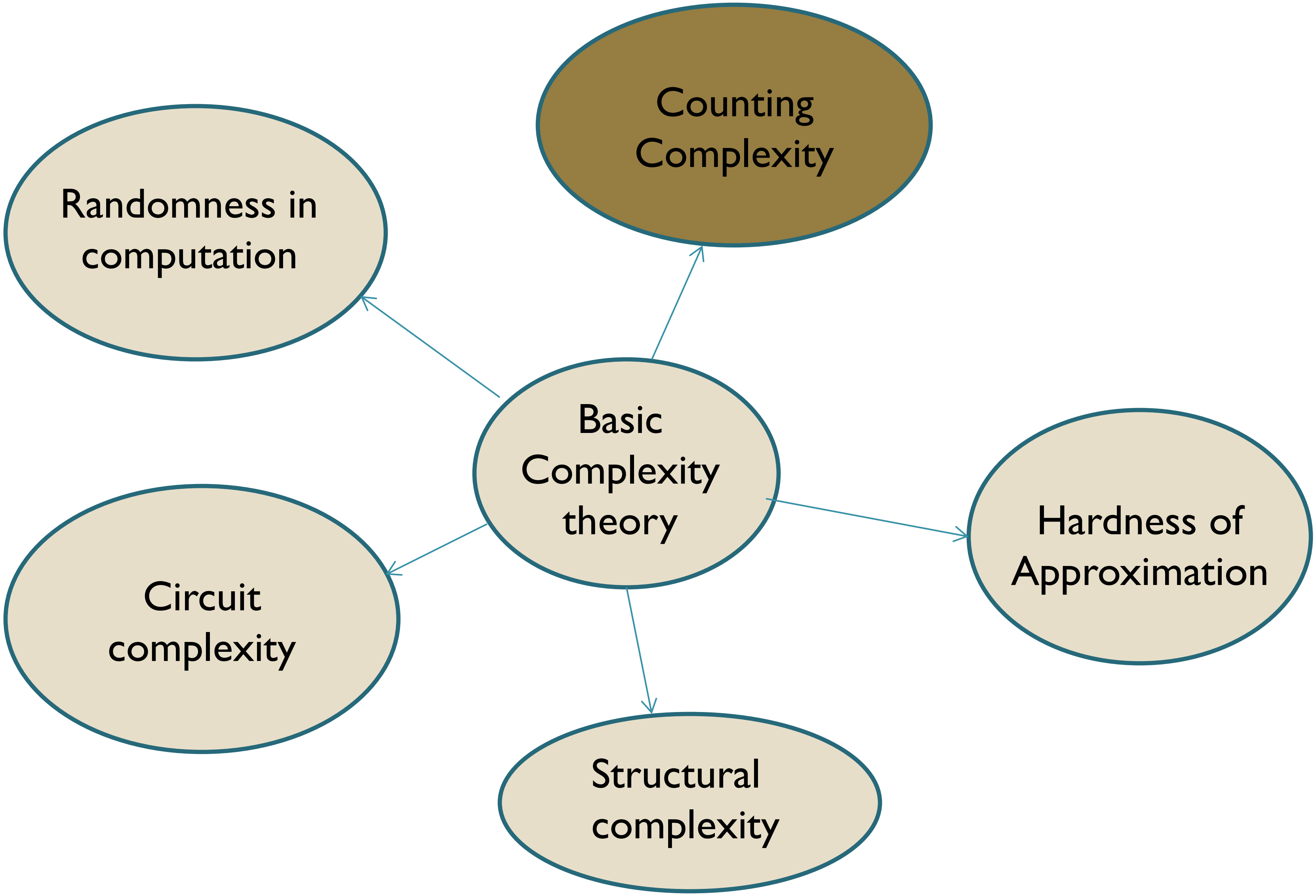
Randomness in Computation

- Probabilistic complexity classes (BPP, RP, co-RP).
  - Does randomization help in improving efficiency?
  - Quicksort has O(n log n) expected time but O(n^2) worst case time.
  - Can SAT be solved in polynomial time using randomness?

    Theorem (Schoening, 1999): 3SAT can be solved in *randomized* $O((4/3)^n)$ time.

  - Access to random bits can help improve computational efficiency… but, to what extent?

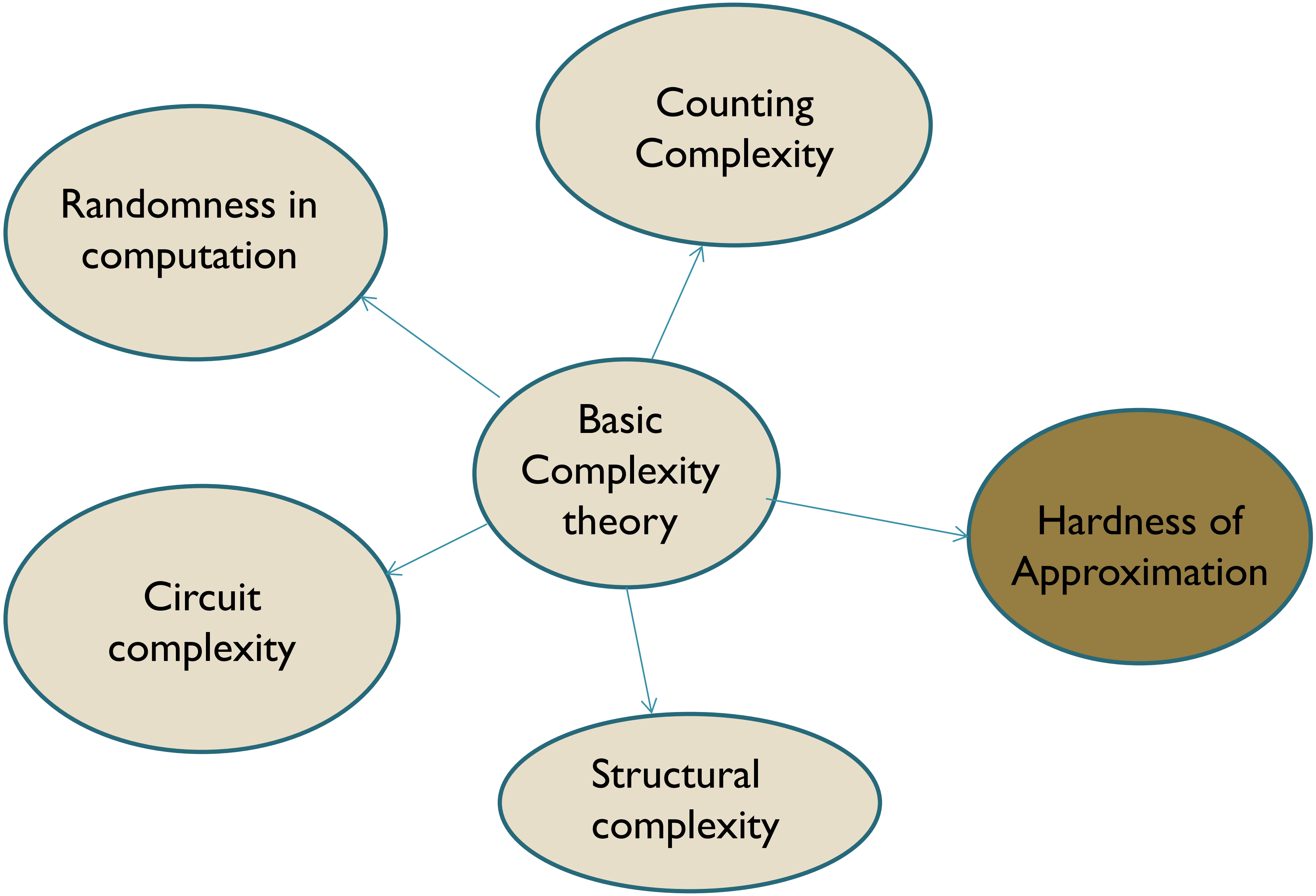
Topics to be covered in this course



Counting Complexity

- Counting complexity classes (class #P).
  - How hard is it to count the number of perfect matchings in a graph?
  - How hard is it to count the number of cycles in a graph?
  - Can we compute the number of simple paths between s and t in G efficiently?
  - Is counting much harder than the corresponding decision problem?

Topics to be covered in this course



Basic Complexity theory

Counting Complexity

Randomness in computation

Circuit complexity

Structural complexity

Hardness of Approximation

Hardness of Approximation

- Probabilistically Checkable Proofs (PCPs).

Hardness of approximation results.

Theorem (Hastad, 1997): If there's a poly-time algorithm to compute an assignment that satisfies at least 7/8 + ε fraction of the clauses of an input 3SAT, for any constant ε > 0, then P = NP.

Course Info

- **Course no.:** E0 224 **Credits:** 3:1
- **Instructor:** Chandan Saha
- **Lecture time:** M, W 2-3:30 pm. **Venue:** CSA 112

- **Course homepage:**

https://www.csa.iisc.ac.in/~chandan/courses/complexity22/home.html

Course Info

- **Prerequisites**: Basic familiarity with algorithms; Mathematical maturity.
- **Primary reference:** Computational Complexity – A Modern Approach by Sanjeev Arora and Boaz Barak.
- **Lectures:** Slides will be posted on the course homepage.
- **Number of lectures:** ~27.

Course Info

- **Grading policy**: Three assignments - 45%
  One presentation - 25%
  Final exam - 30%

Assignments

- **First assignment:** Will posted on Aug 31; due date will be Sep 14.
- **Second assignment:** Will posted on Sep 30; due date will be Oct 14.
- **Third assignment:** Will posted on Oct 31; due date will be Nov 14.
- **Mode:** Assignments will be posted on the course homepage. You need to e-mail me your assignment as a pdf file (use Latex).

Presentations

- A group of 2 students would present a paper/result.
- **Duration of a presentation:** 1-1.5 hr.
- **Mode:** In class, use slides.

- I will start giving topics to present from mid-Sep. All topics will be handed out by mid-Oct.
- You will get ~4 weeks to prepare a presentation.
- We will finish all the presentations by Nov 23 (Wed).

Final exam

- Would be a 3 hr long written test.

- **When?** First week of Dec.

Let’s begin…

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model (introduced by Alan Turing in 1936).

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model (introduced by Alan Turing in 1936).
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model (introduced by Alan Turing in 1936).
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules
- Turing machines ↔ A mathematical way to describe algorithms.

Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model (introduced by Alan Turing in 1936).
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules

(e.g. of a physical realization of a TM is a simple adder)

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γis a finite set of alphabets. (Every memory cell contains an element of Γ)

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)

has a blank symbol

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)
- Q is a finite set of states. (special states: q_{start} , q_{halt})
- δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L,S,R\}^k$

Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)
- Q is a finite set of states. (special states: q_{start} , q_{halt})
- δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L,S,R\}^k$

known as transition function; it captures the dynamics of M

Turing Machines: Computation

- Start configuration.
  - All tapes other than the input tape contain blank symbols.
  - The input tape contains the input string.
  - All the head positions are at the start of the tapes.
  - The machine is in the start state q_{start} .

Turing Machines: Computation

- Start configuration.
  - All tapes other than the input tape contain blank symbols.
  - The input tape contains the input string.
  - All the head positions are at the start of the tapes.
  - The machine is in the start state q_{start} .
- Computation.
  - A **step of computation** is performed by applying δ.
- Halting.
  - Once the machine enters q_{halt} it stops computation.

Turing Machines: Running time

- Let f: $\{0,1\}^* \rightarrow \{0,1\}^*$ and T: N → N and M be a Turing machine.

- Definition. We say M ***computes*** f if on every x in $\{0,1\}^*$, M halts with f(x) on its output tape beginning from the start configuration with x on its input tape.

Turing Machines: Running time

- Let f: {0,1}* → {0,1}* and T: N → N and M be a Turing machine.

- Definition. We say M ***computes*** f if on every x in {0,1}*, M halts with f(x) on its output tape beginning from the start configuration with x on its input tape.

- Definition. M computes f *in T(|x|)* ***time***, if for every x in {0,1}*, M halts within T(|x|) steps of computation and outputs f(x).

Turing Machines

- In this course, we would be dealing with
  - Turing machines that halt on every input.
  - Computational problems that can be solved by Turing machines.

Turing Machines

- In this course, we would be dealing with
  - ➢ Turing machines that halt on every input.
  - ➢ Computational problems that can be solved by Turing machines.

- Can every computational problem be solved using Turing machines?

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - A typical input instance:

$$\left.\begin{array}{l} x^2y + 5y^3 = 3 \\[2em] x^2 + z^5 - 3y^2 = 0 \\[2em] y^2 - 4z^6 = 0 \end{array}\right\} \text{Integer solutions for x, y, z?}$$

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

(Hilbert's tenth problem, 1900)

Turing Machines: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

- Theorem. There doesn't exist any algorithm (realizable by a TM) to solve this problem. (Davis, Putnam, Robinson, Matiyasevich 1970)

Why Turing Machines?

- TMs are natural and intuitive.
- Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated by a Turing machine".*

--- [quoted from Arora-Barak's book]

Why Turing Machines?

- TMs are natural and intuitive.

- Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated by a Turing machine"*.

  --- [quoted from Arora-Barak's book]

- Several other computational models can be simulated by TMs.

Why Turing Machines?

- TMs are natural and intuitive.

- Strong Church-Turing thesis: *“Every physically realizable computation device – whether it’s based on silicon, DNA, neurons or some other alien technology – can be simulated efficiently by a Turing machine”.*

Possible exception: Quantum machines!

Basic facts about TMs

Turing Machines

- Time constructible functions. A function $T: N \rightarrow N$ is *time constructible* if $T(n) \geq n$ and there's a TM that computes the function that maps x to $\underbrace{T(|x|)}_{\text{in binary}}$ in $O(T(|x|))$ time.

- Examples: $T(n) = n^2$, or 2^n, or $n \log n$

Turing Machines: Robustness

- Let f: {0,1}* → {0,1}* and T: N → N be a time constructible function.

- Binary alphabets suffice.
  - If a TM M computes f in T(n) time using Γ as the alphabet set, then there's another TM M' that computes f in time 4.log |Γ| .T(n) using {0, 1, blank} as the alphabet set.

Turing Machines: Robustness

- Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ and $T: \mathbb{N} \rightarrow \mathbb{N}$ be a time constructible function.

- Binary alphabets suffice.
  - If a TM M computes f in $T(n)$ time using Γ as the alphabet set, then there's another TM M' that computes f in time $4.\log |\Gamma| .T(n)$ using $\{0, 1, \text{blank}\}$ as the alphabet set.

- A single tape suffices.
  - If a TM M computes f in $T(n)$ time using k tapes then there's another TM M' that computes f in time $5k \,.\, T(n)^2$ using a single tape that is used for input, work and output.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

…simply encode the description of the TM.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

  …invalid strings map to a fixed, trivial TM.

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

- Every TM has infinitely many string representations.

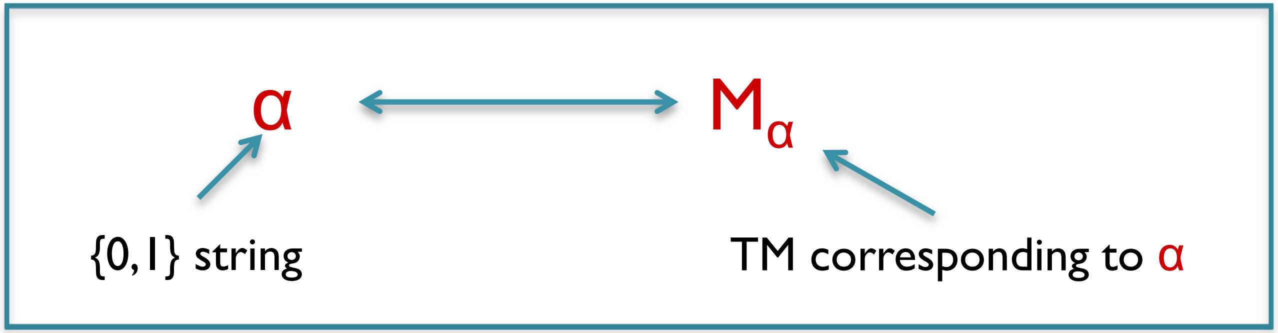
  … allow padding with arbitrary number of 0's

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

- Every TM has infinitely many string representations.



α $\longleftrightarrow$ M_α

{0,1} string

TM corresponding to α

Turing Machines: As strings

- Every TM can be represented by a finite string over {0,1}.

- Every string over {0,1} represents some TM.

- Every TM has infinitely many string representations.

- A TM (i.e., its string representation) can be given as an input to another TM !!

Universal Turing Machines

- Theorem. There exists a TM U that on every input x, α in $\{0,1\}^*$ outputs $M_\alpha(x)$.

- Further, if M_α halts within T steps then U halts within $C. T. \log T$ steps, where C is a constant that depends only on M_α 's alphabet size, number of states and number of tapes.

Universal Turing Machines

- Theorem. There exists a TM U that on every input x, α in {0,1}* outputs $M_\alpha(x)$.

- Further, if M_α halts within T steps then U halts within C. T. log T steps, where C is a constant that depends only on M_α 's alphabet size, number of states and number of tapes.

- Physical realization of UTMs are modern day electronic computers.