# ACM India at a Glance

- **ACM**: world's largest educational and scientific computing society
  - **Mission**: advancing computing as science and profession
  - **Members**: ~100,000 worldwide, ~11000 in India
  - Comprising students, faculty, professionals
- **ACM India Chapters**: ~200 student chapters, ~20 professional chapters
- **ACM-W India**: empowering women in computing
- **Research Initiatives**
  - Student research: ARCS Symposium, best doctoral dissertation, partial travel grant
  - Research conferences: CODS-COMAD, ISEC, AIMS
- **ACM India Annual Event**
  - Discuss recent trends in technology and celebrate India's achievements in computing
- **Education Initiatives**
  - Summer and winter schools: ~2 week full-time course on technology area
  - Compute: Symposium focused on improving quality of computing education in India
  - CSpathshala: inculcate computational thinking in schools
- **Learning and Professional Development**
  - Eminent Speaker Program
  - Industry Webinars, Education Webinars
  - Blogs: theoreticians and practitioners sharing ideas, opinions
  - ACM global resources: Digital Library, ACM Learning Center
- **New prestigious awards instituted**
  - Acknowledge and celebrate outstanding contributions
- **ACM Membership in India**
  - Student? student member form
  - Professional? professional member form

# Why should we learn Theory of Computation?

R. Ramanujam

The Institute of Mathematical Sciences, Chennai, India

email: jam@imsc.res.in

ACM Education Webinar

March 11, 2021

# A quote

Some words well worth listening to:

> *We should explain, before proceeding, that it is not our object to consider this program with reference to the actual arrangement of the data on the Variables of the engine, but simply as an abstract question of the nature and number of the operations required to be perfomed during its complete solution.*

# A quote

Some words well worth listening to:

> *We should explain, before proceeding, that it is not our object to consider this program with reference to the actual arrangement of the data on the Variables of the engine, but simply as an abstract question of the nature and number of the operations required to be perfomed during its complete solution.*
>
> *Ada Augusta Byron King, Countess of Lovelace (1843)*

# Another quote

*We are presently in the midst of a third intellectual revolution. The first came with Newton: the planets obey physical laws. The second came with Darwin: biology obeys genetic laws. In today's third revolution, we are coming to realize that even minds and societies emerge from interacting laws that can be regarded as computations.*
*Everything is computation.*

*Rudy Rucker*

# A few questions

Some questions we need answers to:

- Why should we study theory of computation?

# A few questions

Some questions we need answers to:

- Why should we study theory of computation?
- Why should we study theory of computation?

# A few questions

Some questions we need answers to:

- Why should we study theory of computation?
- Why should we study theory of computation?
- What can we expect a (reasonably sincere) student to learn from a ToC course ?

# A science ?

Is computer science really a science ?

# A science ?

Is computer science really a science ?

- Europeans call it information science. But the question remains.
- What is the distinguishing characteristic of science?

# A science ?

Is computer science really a science ?

- Europeans call it information science. But the question remains.
- What is the distinguishing characteristic of science?
- Is there a role for experimentation in computer science?

# A machine science ?

Can we really have any machine science at all?

# A machine science ?

Can we really have any machine science at all?

- If we can have one, what would be the elements of a machine science?

# A machine science ?

Can we really have any machine science at all?

- If we can have one, what would be the elements of a machine science?
- A proposal for a pressure cooker science!

# A machine science ?

Can we really have any machine science at all?

- If we can have one, what would be the elements of a machine science?
- A proposal for a pressure cooker science!
- What are the elements of a technology independent formulation of pressure cooker science?

# What is a computer?

We have a view of a computer as a problem solver.

- When did you last solve a problem using a computer?

# What is a computer?

We have a view of a computer as a problem solver.

- When did you last solve a problem using a computer?
- What software do you use when you switch on your laptop or smartphone?

# What is a computer?

We have a view of a computer as a problem solver.

- When did you last solve a problem using a computer?
- What software do you use when you switch on your laptop or smartphone?
- Are operating systems, browsers and mobile apps problem solvers? What problem does On24 solve?

# What is a computer?

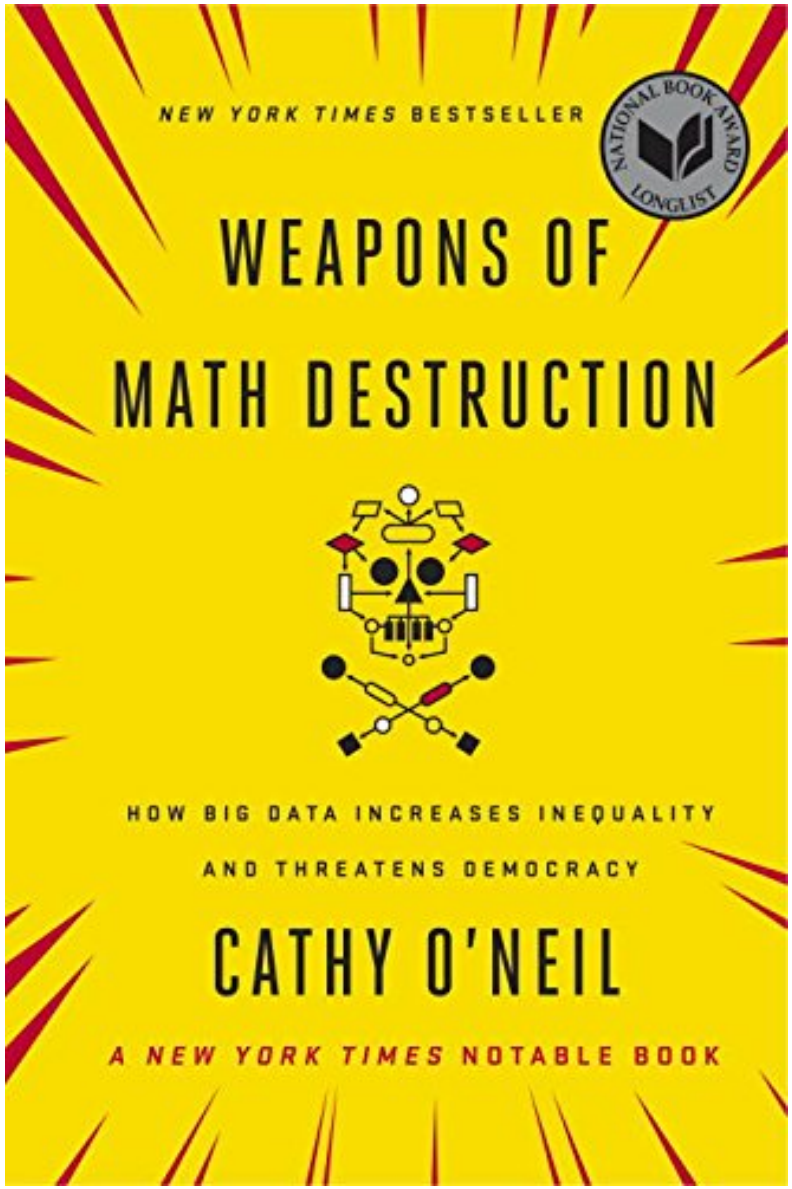
We have a view of a computer as a problem solver.

- When did you last solve a problem using a computer?
- What software do you use when you switch on your laptop or smartphone?
- Are operating systems, browsers and mobile apps problem solvers? What problem does On24 solve?
- Computer as communicator, as a reactive system.

# Why bother?

Why should we worry about such things at all?

# The age of big data




Source: Amazon

# Algorithms in our lives

As algorithms interweave into our social lives more and more, the need for philosophical / logical foundations becomes stronger.

- We use a phrase like my Inbox folder (say on Gmail) much like my pen but the two have very different logical properties.

# Algorithms in our lives

As algorithms interweave into our social lives more and more, the need for philosophical / logical foundations becomes stronger.

- We use a phrase like my Inbox folder (say on Gmail) much like my pen but the two have very different logical properties.
- Similarly we speak of my community and my identity, but they are different in the digital and physical worlds.

# Algorithms in our lives

As algorithms interweave into our social lives more and more, the need for philosophical / logical foundations becomes stronger.

- We use a phrase like my Inbox folder (say on Gmail) much like my pen but the two have very different logical properties.
- Similarly we speak of my community and my identity, but they are different in the digital and physical worlds.
- The analogy is to food: from the days we got food directly from farms to now: until the last century, social algorithms admitted experiential understanding, but that is changing rapidly.

# The central questions

If we do want to understand the foundations of computation, what are the central questions to ask? William Rapaport suggests the following.

- What can be computed, and how?
- What can be efficiently computed, and how?
- What can be practically computed, and how?
- What can be physically computed, and how?
- What can be ethically computed, and how?

# The Great Insights: 1

Rapaport offers the following as the Great Insights of computer science.

- Representational Insight: All information about computation can be represented in binary. (Bacon, Leibniz, Morse, Boole, Ramsey, Turing)
  Two nouns, 0 and 1, will do.

# The Great Insights: 1

Rapaport offers the following as the Great Insights of computer science.

- Representational Insight: All information about computation can be represented in binary. (Bacon, Leibniz, Morse, Boole, Ramsey, Turing)
  Two nouns, 0 and 1, will do.
- Processing Insight: Every algorithm can be expressed using the language of simple machines. (Turing)
  Three verbs: *move*, *write* and *erase* will do.

# The Great Insights: 1

Rapaport offers the following as the Great Insights of computer science.

- Representational Insight: All information about computation can be represented in binary. (Bacon, Leibniz, Morse, Boole, Ramsey, Turing)
  Two nouns, 0 and 1, will do.
- Processing Insight: Every algorithm can be expressed using the language of simple machines. (Turing)
  Three verbs: *move*, *write* and *erase* will do.
- Structural Insight: All computer programs can be written using only sequencing, choice and repetition. (Böhm and Jacopini)
  Three rules of grammar will do.

# The Great Insights: 2

After the linguistic insights, we have procedural insights:

- Computablity Insight: The informal notion of effective procedure corresponds to computability by Turing machine.
  The Church Turing thesis has withstood many tests by now.

# The Great Insights: 2

After the linguistic insights, we have procedural insights:

- Computablity Insight: The informal notion of effective procedure corresponds to computability by Turing machine.
  The Church Turing thesis has withstood many tests by now.
- Implementation Insight: Turing machines can be implemented physically (using only one kind of *logic gate*). (Turing, Kay, Denning, Piccinini).

As new models of computation like Quantum computing, DNA computing and Chaos-based computing emerge, the basic insights continue to be strengthened.

# Computational thinking

Elements of computational thinking (Jeannette Wing, 2006).

- Decomposition
- Pattern recognition
- Abstraction
- Algorithm design

The questions raised by CT are subsumed by the ones above; the difference is a matter of emphasis.

# The hardware and the software

In the internet era, old notions of computers, hardware and software are becoming obsolete.

- What is an algorithm? What is computation? Is it possible to talk of these with no reference to any machine?
- What is a computing agent? What are its boundaries? How do we ascribe action and agency to it? (Can we ascribe life to it? consciousness?)
- Is there an intrinsic meaning that sets a computing agent apart from other entities?
- What is an adequate ontology of computation?

# The Ethical

As I have indicated earlier, algorithms are running our lives now, so ethical considerations are relevant.

- Where should computing agents be used? Why?
- Are there personal, social and political contexts where the use of computing agents is undesirable?
- Who decides whether we should use them and how?
- Can we ascribe ethics to agent behaviour?
- What are the ethical dimensions of collective belief and action mediated by computing agents? Are collective identities of computing agents meaningful?

# Theory of computation

So what does theory of computation teach us, really?

# State transition systems

This is the oldest model, coming from the natural sciences.

- A system is one of several possible states.
- Every input or stimulus causes a change of state.
- Desirable behaviour of the system corresponds to partitioning states into two: *good* and *bad*.
- It is reasonable to start the system from a designated state.

# A model

We already have a mathematical model on hand.

- A system is a tuple $S = (Q, \Sigma, \delta, q_0, G)$ where:
- $Q$ is the set of possible states, $\Sigma$ is the set of possible inputs;
- $\delta : (Q \times \Sigma) \to Q$ is the transition fucntion;
- $q_0 \in Q$ is the initial state and $G \subseteq Q$ is the set of good states.

# Outputs

We have not said anything about outputs of the system!

- We can add another set $\Gamma$ of possible outputs;
- A map $\theta : Q \to \Gamma$ can then specify what outputs the system gives depending on its state.
- However, we can study the behaviour of the system simply by seeing how it reacts to inputs (in terms of the state change).

# Outputs

We have not said anything about outputs of the system!

- We can add another set $\Gamma$ of possible outputs;
- A map $\theta : Q \to \Gamma$ can then specify what outputs the system gives depending on its state.
- However, we can study the behaviour of the system simply by seeing how it reacts to inputs (in terms of the state change).
- Input – output behaviour is split into input – state – output, and the latter can be derived.

# An exercise

The best way to understand system design is to try and design one.

- We take up a vending machine, that gives a cup of tea for Rs 2.
- It can take Re 1 coins and Rs 2 coins.
- What are its states, inputs, transitions? How do we study its behaviour?

# Behaviour

Sequences of inputs provide a covenient tool for studying behaviour.

- Every sequence of inputs leads to a unique state.
- If that state is good, we consider this sequence of inputs from the environment suitable or acceptable for the system.
- Otherwise, the enviornment behaviour is deemed unacceptable to the system.
- Note that it is the behaviour of the environment that is accepted or rejected **by** the system.

# Elements of automata theory

Finite state automata capture the notion of regular behaviour.

- Nondeterminism.

# Elements of automata theory

Finite state automata capture the notion of regular behaviour.

- Nondeterminism.
- Robustness.

# Elements of automata theory

Finite state automata capture the notion of regular behaviour.

- Nondeterminism.
- Robustness.
- Optimization.

# Elements of automata theory

Finite state automata capture the notion of regular behaviour.

- Nondeterminism.
- Robustness.
- Optimization.
- Existence of algorithms for analysis.

# Programs

Regular expressions: a small programming language.

- Equivalence of machines and programs.

# Programs

Regular expressions: a small programming language.

- Equivalence of machines and programs.
- Unified by an algebraic notion of equivalence.

# Programs

Regular expressions: a small programming language.

- Equivalence of machines and programs.
- Unified by an algebraic notion of equivalence.
- Languages definable in a suitable logic. (The monadic second order logic of sequences.)

# Generalisations

A theory of regular anything !

- Inputs need not be words, can be trees. Leads to a theory of automata on XML documents and databases.

# Generalisations

A theory of regular anything !

- Inputs need not be words, can be trees. Leads to a theory of automata on XML documents and databases.
- Inputs can be pictures: Leads to a theory of picture languages, with applications to image processing.

# Generalisations

A theory of regular anything !

- Inputs need not be words, can be trees. Leads to a theory of automata on XML documents and databases.
- Inputs can be pictures: Leads to a theory of picture languages, with applications to image processing.
- Inputs can be DNA sequences: Leads to a nice theory of molecular computation.
- Inputs can be infinite sequences: Perhaps the most extensive application of automata theory today, in hardware and software verification.

# Infinite state systems

Finite state systems = bounded memory.

- Extend machine by storage space.

# Infinite state systems

Finite state systems = bounded memory.

- Extend machine by storage space.
- Unbounded memory means that such storage space is not fixed.

# Infinite state systems

Finite state systems = bounded memory.

- Extend machine by storage space.
- Unbounded memory means that such storage space is not fixed.
- Design such a machine.

# Alan Turing

British mathematician, who solved the Enigma code.

# Turing 1936

One of the most remarkable papers ever written. In a singe paper, Alan Turing managed to accomplish all this:

- Defined the notion of effective procedure mathematically.

# Turing 1936

One of the most remarkable papers ever written. In a singe paper, Alan Turing managed to accomplish all this:

- Defined the notion of effective procedure mathematically.
- Showed that it can simulate any reasonable notion of computation and hence is as general as any intuitive computing machine.

# Turing 1936

One of the most remarkable papers ever written. In a singe paper, Alan Turing managed to accomplish all this:

- Defined the notion of effective procedure mathematically.
- Showed that it can simulate any reasonable notion of computation and hence is as general as any intuitive computing machine.
- Showed that it has limitations, and used it to show the unsolvability of an important problem in logic.

# Turing 1936

One of the most remarkable papers ever written. In a singe paper, Alan Turing managed to accomplish all this:

- Defined the notion of effective procedure mathematically.
- Showed that it can simulate any reasonable notion of computation and hence is as general as any intuitive computing machine.
- Showed that it has limitations, and used it to show the unsolvability of an important problem in logic.
- Constructed a Universal machine that can take any other machine as input and behave like it.

# Elements of Computability theory

The central elements to understand:

- Nondeterminism.

# Elements of Computability theory

The central elements to understand:

- Nondeterminism.
- Robustness.

# Elements of Computability theory

The central elements to understand:

- Nondeterminism.
- Robustness.
- Equivalence of programs and machines.

# Elements of Computability theory

The central elements to understand:

- Nondeterminism.
- Robustness.
- Equivalence of programs and machines.
- Unsolvability.

# Elements of Computability theory

The central elements to understand:

- Nondeterminism.
- Robustness.
- Equivalence of programs and machines.
- Unsolvability.
- Resource limited machines.

# A nice problem

Assume that you are given a finite set of colours $C$.

- You are given two functions: $R, U : C \to 2^C$ where we ensure that for all $c \in C$, $R(c)$ and $U(c)$ are nonempty.

# A nice problem

Assume that you are given a finite set of colours $C$.

- You are given two functions: $R, U : C \to 2^C$ where we ensure that for all $c \in C$, $R(c)$ and $U(c)$ are nonempty.
- You are asked to give a map $\chi : (N \times N) \to C$, such that for all $m, n \in N$, $\chi(m, n+1) \in R(\chi(m, n))$ and $\chi(m+1, n) \in U(\chi(m, n))$.

# A nice problem

Assume that you are given a finite set of colours $C$.

- You are given two functions: $R, U : C \to 2^C$ where we ensure that for all $c \in C$, $R(c)$ and $U(c)$ are nonempty.
- You are asked to give a map $\chi : (N \times N) \to C$, such that for all $m, n \in N$, $\chi(m, n+1) \in R(\chi(m, n))$ and $\chi(m+1, n) \in U(\chi(m, n))$.
- This problem is unsolvable, and we can reduce the non-halting problem of Turing machines to it.

# Halting problem for TMs

Input: A Turing machine $M$ and an input string $w$.
Question: Does $M$ halt on $w$?

- Suppose this is solvable. Then there must exist a (universal) Turing machine $U$ that takes a coding of $M$ and $w$ as input and outputs $\#1$ if $M$ halts on $w$.

# Halting problem for TMs

Input: A Turing machine $M$ and an input string $w$.
Question: Does $M$ halt on $w$?

- Suppose this is solvable. Then there must exist a (universal) Turing machine $U$ that takes a coding of $M$ and $w$ as input and outputs $\#1$ if $M$ halts on $w$.
- How do we construct such universal machines? We can discuss this later, but Turing already showed in 1936 how to do this.

# Halting problem (contd)

If Halting problem is solvable then we have a UTM U that implements it.

- If this is true, then there exists a TM $D$ that takes a coding of a machine $M$ and outputs $\#1$ if $M$ does not halt on $|M|$, where the latter is the code of $M$.

# Halting problem (contd)

If Halting problem is solvable then we have a UTM U that implements it.

- If this is true, then there exists a TM $D$ that takes a coding of a machine $M$ and outputs $\#1$ if $M$ does not halt on $|M|$, where the latter is the code of $M$.
- That is, $D$ decides if $M$ running on itselfis non-terminating.

# Halting problem (contd)

If Halting problem is solvable then we have a UTM U that implements it.

- If this is true, then there exists a TM $D$ that takes a coding of a machine $M$ and outputs $\#1$ if $M$ does not halt on $|M|$, where the latter is the code of $M$.
- That is, $D$ decides if $M$ running on itselfis non-terminating.
- Running $D$ on itself reveals a paradox: running $D$ on itself terminates (and accepts) iff running $D$ on itself is non-terminating.

# Halting problem (contd)

If Halting problem is solvable then we have a UTM U that implements it.

- If this is true, then there exists a TM $D$ that takes a coding of a machine $M$ and outputs $\#1$ if $M$ does not halt on $|M|$, where the latter is the code of $M$.
- That is, $D$ decides if $M$ running on itselfis non-terminating.
- Running $D$ on itself reveals a paradox: running $D$ on itself terminates (and accepts) iff running $D$ on itself is non-terminating.
- No such $D$ can exist; therefore no such $U$ exists either. Hence the halting problem for TMs is unsolvable.

# Coping with unsolvability

Every non-trivial question on arbitrary programs is undecidable.

- But the termination of a specific program is often easy to decide.

# Coping with unsolvability

Every non-trivial question on arbitrary programs is undecidable.

- But the termination of a specific program is often easy to decide.
- Further, the static analysis of a program may already reveal a great deal of information without needing to look at runtime behaviour.

# Coping with unsolvability

Every non-trivial question on arbitrary programs is undecidable.

- But the termination of a specific program is often easy to decide.
- Further, the static analysis of a program may already reveal a great deal of information without needing to look at runtime behaviour.
- The class of infinite state systems that we can analyse is a major area of research in this century.

# New models of computation

The last few decades have given new models, all equivalent to TMs.

- Biologically inspired models of computing.
- Quantum computers.
- Chaos-based computation.
- Circuits as computation. (Used in complexity theory.)

# Why study ToC?

The study of ToC is ultimately about learning two skills that are crucial for all computer scientists.

# Why study ToC?

The study of ToC is ultimately about learning two skills that are crucial for all computer scientists.

- **Intuition**: How to think critically about abstract computation.
- **Language**: How to talk about abstract computation.
- As teachers, our main role is to share the conviction that thinking and talking about abstract computation is crucial for computer science students.

# The power of abstraction

Abstraction is the most crucial component of computational thinking.

ToC teaches you:

- The power of finite state abstraction.

# The power of abstraction

Abstraction is the most crucial component of computational thinking.

ToC teaches you:

- The power of finite state abstraction.
- How to analyse the limitations of your abstraction.

# The power of abstraction

Abstraction is the most crucial component of computational thinking.

ToC teaches you:

- The power of finite state abstraction.
- How to analyse the limitations of your abstraction.
- Most importantly, resource consciousness.

# Teaching computer science

What does ToC mean for computer science teachers?

# Teaching computer science

What does ToC mean for computer science teachers?

- How do we understand computers without worrying about technology?
- What is the abstract relationship between computers and programs?
- With the omnipresence of computers and programs, students may consider every problem to be solvable by computer. This is wrong.
- Hardware will change, sofware will change. The model will not.

# A quote

A note of caution from one of the eminent thinkers of computer science:

> *The trouble with computer science is that it is so applicable that you can be blamed for not doing immediately applicable research. There is such an urgent demand for immediate applications that if you do work with long term implications, you can be criticised. That is a great danger, because computer science is so broad, there is so much computing happening, so much communication, we will all be lost without a basic theory.*

Robin Milner

ACM Turing Award Lecture, 1992

# Discussion time

Thank you.
Questions, comments, suggestions welcome; also, please write to jam@imsc.res.in.