Fundamental Algorithmic Techniques XI

December 16, 2025

Outline

Finite and Infinite Programming

Models of Computation

- **Circuits** $\rightarrow$ finite, fixed-size programs (exponential lengths...)
- **Automata** $\rightarrow$ handle infinite inputs/outputs, but limited to regular/simple problems
- **Turing Machine**
  - One per problem
  - Infinite, writable tape
  - Finite set of inner states
  - Transition function/table
- **Universal Turing Machine** Programmable computer!

Turing-Complete Systems

- NAND-TM language
- RAM model (e.g., Python, C)
- Lambda calculus (e.g., Lisp, OCaml, Clojure)
- Cellular automata (e.g., Conway's Game of Life)

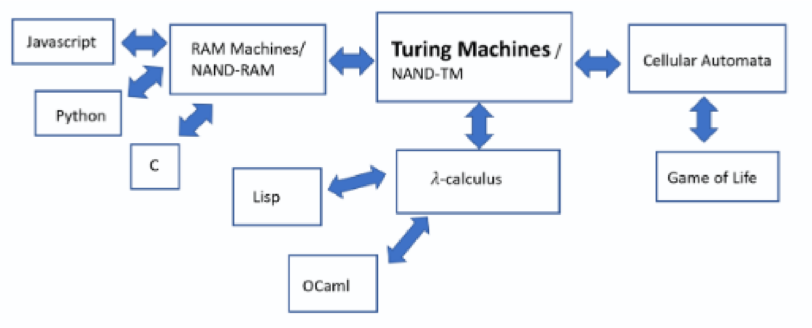
Church–Turing Thesis:

A function on natural numbers is effectively computable

$\Leftrightarrow$

It is computable by a Turing machine.

Turing Complete & Computability



Let's have a closer look at computation types of classes!

Incomputability by Turing Machine

One can prove that **infinitely many** functions:

$$\mathcal{F} : \{0,1\}^* \to 0,1$$

are **uncomputable** by a Turing Machine.
Well-known examples:

- **Halting Problem**: machine $M : \{0,1\}^* \to \{0,1\}$, $\forall x \in \{0,1\}^*$,
  - $\mathrm{Halt}(M,x) = 1$ if $M(x)$ halts
  - else $\mathrm{Halt}(M,x) = 1$

  Proof sketch: function $\tilde{M}$ with infinite loop if $M(x)$ halts.
- **Busy Beaver**:
  For a Turing Machine of n states: M_n, find longest max amount of steps before M_n halts!

Incomputability by Turing Machine

One can prove that **infinitely many** functions:

$$\mathcal{F} : \{0,1\}^* \to \{0,1\}$$

are **uncomputable** by a Turing Machine.

Well-known examples:

- **Halting Problem**:

  $$\mathrm{Halt}(M,x) = \begin{cases} 1 & \text{if Turing machine } M \text{ halts on input } x, \\ 0 & \text{otherwise.} \end{cases}$$

  This function is uncomputable.

- **Busy Beaver**:

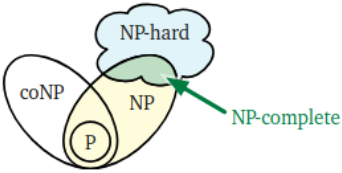
  For an n-state Turing machine M_n, find the maximum number of steps M_n can run before halting (over all inputs and all such machines). Grows faster than any computable function!

Computational Complexity Classes — The Big Picture

Decision problems (answer: Yes/No)

Class	**Meaning**	**Example**
P	Solvable in poly-time	Sorting, shortest path
NP	Verifiable in poly-time	SAT, TSP (decision)
coNP	"No" answers verifiable in poly-time	Formula validity
NP-complete	In NP + NP-hard	3-SAT, Clique
NP-hard	At least as hard as any NP problem	Halting Problem, opt. TSP
Uncomputable	Uncomputable as seen above	Halting Problem, Busy Beave

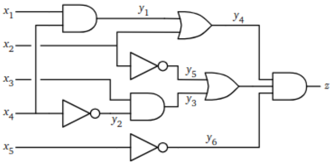
NP-hard, and NP-complete



Relationships among P, NP, NP-complete, and NP-hard classes. One assumes $P \neq NP$ but it is unproven.

- **NP-hard**: A problem Π is NP-hard if a polynomial-time algorithm for Π would imply a polynomial-time algorithm for every problem in NP.
- **NP-complete**: A problem that is both NP-hard and an element of NP.

SAT, 3SAT, and NP-Completeness



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

Every Boolean circuit $\mathcal{C} : \{0,1\}^n \rightarrow \{0,1\}$ can be converted in $\mathcal{O}(n)$ time to an equivalent Boolean formula or satisfiability problem **SAT**.

Circuit-SAT $\leq_p$ SAT

The **Cook–Levin Theorem** shows:
Any NP computation can be encoded as a Boolean circuit $\rightarrow$ then as a formula.
Thus, **SAT is NP-hard**.

SAT, 3SAT, and NP-Completeness

SAT $\in$ NP

SAT serves as a polynomial-size certificate $\rightarrow$ verifiable in poly-time.

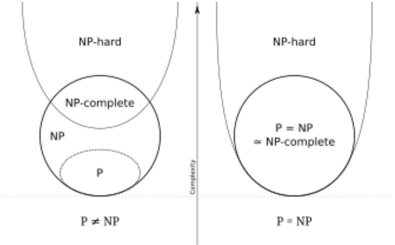
$\Rightarrow$ **SAT is NP-complete**.

3SAT

Restrict SAT to CNF formulas with **exactly 3 literals per clause**.
3SAT is also NP-complete (via polynomial reduction from SAT).
Widely used in hardness proofs.

Karp (1972) showed 21 problems (including 3SAT, Clique, Vertex Cover) are NP-complete via reductions from SAT.

$P \neq NP$ versus $P = NP$?



Relationships among P, NP, NP-complete, and NP-hard classes.
The standard assumption is $P \neq NP$, but this remains unproven.

If $P = NP$:

- Every efficiently verifiable solution can also be efficiently found.
- **Breakdown** of modern cryptography (e.g., RSA, ECC).
- **Revolution** in optimization, logistics, scheduling.
- **Transformative advances** in AI, machine learning, and automated reasoning.
- Many currently intractable problems become tractable.

Status: unsolved question

Shannon Entropy — Information is Surprise

How much information does a random variable carry?

$$H(X) = -\sum p_i \log_2 p_i \quad \text{(in bits per symbol)}$$

Coin	$P(\text{Heads})$	$H(X)$
Fair	50%	**1.00 bit** $\leftarrow$ maximum uncertainty
99% heads	99%	≈ 0.08 bit $\leftarrow$ boring
1% heads	1%	≈ 6.6 bits $\leftarrow$ shocking when it lands!

Entropy = average surprise

1 bit = one perfect yes/no question

English text: $H \approx 1$ bit/character $\rightarrow$ 1 MB of text can be compressed to 125 KB (in theory)

The Source Coding Theorem — The Hard Limit

Shannon's Source Coding Theorem (1948):
For a source with entropy $H(X)$ bits/symbol:

- You **cannot** compress below $H(X)$ bits/symbol on average
- You **can** get arbitrarily close — but only for very long messages
- In practice: English ≈ 1 bit/character $\rightarrow$ best possible compression $\approx 12.5\%$ of raw text

Consequences for LLMs training:

Shannon Entropy versus Cross entropy loss: $H(P) = \sum_i p_i \cdot \log(p_i)$ versus $H(P, Q) = \sum_i p_i \cdot \log(q_i)$

- Modern LLMs reach 7–8 bits/token $\rightarrow$ within 10–20% of the theoretical limit.
- Limit for cross entropy loss around 1 (as $Q \rightarrow P$).

Kolmogorov Complexity

Kolmogorov: "What is the shortest program that outputs this exact string?"

K(x) = length of shortest program that prints x and halts

$\rightarrow$ The **true** information content of one object (not a distribution)

Uncomputable ($\sim$ because of Halting Problem)

Examples:

- simple objects: $K(x) = \log(n)$
- random objects: $K(x) = n + \mathrm{O}(\log(n))$ (expensive!)
- $x = 2^m$: $K(x) = \log(m)$ (finite code $f(z) = 2^z$ + description of m)

String	Length	K(x) in bits
`01010101...` (1 million times)	8 MB	$\approx$ 100 bits
`= 3.14159...` (first million digits)	8 MB	$\approx$ 200 KB
`War and Peace`	3 MB	$\approx$ 4–5 MB
Random noise (1 MB)	1 MB	$\approx 8 \cdot 10^6$ bits (incompress!)

Solomonoff - Kolmogorov Complexity

Definition:
task $T : \{0,1\}^* \rightarrow \{0,1\}$.

$$K(T) = min_{p \in T} \mid p \mid,$$

where $\mid p \mid$ is length of the code.

Theorem:
Consider two Universal Turing machines M, N (Kolmogorov):

$$K_N - C \leq K_M \leq K_N + C.$$

With C a constant. Complexity is equivalent for two programming languages, up to the compiler differences (that become negligeable when K_M large).

Nb of lines that can be written by humans is small compared to learning machines.

Solomonoff Induction & Completeness

1. Bayesian model selection Given data D and a theory T, Bayes' rule gives the posterior:

$$\mathbb{P}[T \mid D] = \frac{\mathbb{P}[D \mid T]\,\mathbb{P}[T]}{\mathbb{P}[D \mid T]\,\mathbb{P}[T] + \sum_{A \neq T} \mathbb{P}[D \mid A]\,\mathbb{P}[A]}$$

2. Predicting future data F The optimal predictive distribution marginalizes over all theories:

$$\mathbb{P}[F \mid D] = \mathbb{E}_T\Big[\mathbb{P}[F \mid T, D]\Big] = \sum_T \mathbb{P}[F \mid T, D]\,\mathbb{P}[T \mid D]$$

3. Solomonoff completeness (error bound) Let T^* be the *perfect theory*. The *total expected prediction error* of Solomonoff induction is bounded by the **Kolmogorov complexity** of T^* (simplified: learning works!!!):

$$\sum_{t=1}^{\infty} \mathbb{E}\Big[\,|\mathbb{P}(F_t \mid D_{<t}) - \mathbb{P}_{\text{true}}(F_t \mid D_{<t})|\,\Big] \;\lesssim\; K(T^*)$$

LLMs are gradient descent trying to be Solomonoff induction with 175 billion parameters.