k-tape Turing machine

turing-machine
turing-machine-def

- A finite set A, called the *alphabet*.
- A collection of k *tapes*, each an infinite sequence of *cells* each containing an element of A. One tape is declared to be the *input* and another one is declared to be the *output*.
- A set S of *states*, including two special states: S_{init} and S_{halt}.
- A *transition function* $\delta : A^k \times S \to A^k \times S \times \{L, N, R\}$.
- A *head* which is in a state and a *position* on each tape.

Complexity class **P**

complexity-class::p
complexity-class-p-def

$f \in \mathbf{P}$ iff there exists a Turing machine T and a polynomial p such that, for all x, T computes $f(x)$ in time at most $p(x)$.

Non-deterministic Turing machine

turing-machine::nondet
non-deterministic-turing-machine-def

A non-deterministic Turing machine T is like a Turing machine except that it has two transition functions, and we say that T computes f if, for all input x, $f(x) = 1$ iff there's a sequence of choices between the two transition functions such that the output tape is 1 when T halts.

Complexity class **NP**

complexity-class::np turing-machine::nondet
complexity-class-np-def

$f \in \mathbf{NP}$ iff there exists a non-deterministic Turing machine T and a polynomial p such that, for all x, T computes $f(x)$ in time at most $p(x)$.

Alternative definition of $\mathbf{NP}$

$f \in \mathbf{NP}$ iff there is a polynomial p and a function $g \in P$ such that, for every x, $f(x) = 1$ iff $\exists y \in \{0,1\}^{p(|x|)}, g(x,y) = 1$.

Proof.

$\Longrightarrow$ Use y to encode the choices made by the non-deterministic TM. Then the TM for g is "Run the non-deterministic TM, reading y to know which transition function to follow.".

$\Longleftarrow$ Non-deterministically write down y and apply g.

$\square$

```
complexity-class:np
complexity-class-np-def-alt
```

Complexity class $\mathbf{co} - \mathbf{NP}$

$f \in \mathbf{co} - \mathbf{NP}$ iff $\neg f \in \mathbf{NP}$

Alternatively, $f \in \mathbf{co} - \mathbf{NP}$ iff there is a polynomial p and a function $g \in P$ such that, for every x, $f(x) = 1$ iff $\forall y \in \{0,1\}^{p(|x|)}, g(x,y) = 1$.

```
complexity-class:co-np
complexity-class-co-np
```

Polynomial hierarchy

Define $\mathbf{\Sigma_0^P}$ and $\mathbf{\Pi_0^P}$ to be $\mathbf{P}$ and

$$\begin{aligned} f \in \mathbf{\Sigma^P_{k+1}} &\iff \exists \text{ polynomial } p \text{ and } g \in \mathbf{\Pi^P_k}, \\ &\qquad \forall x, f(x) = 1 \iff \exists y, g(x,y) = 1 \\ f \in \mathbf{\Pi^P_{k+1}} &\iff \exists \text{ polynomial } p \text{ and } g \in \mathbf{\Sigma^P_k}, \\ &\qquad \forall x, f(x) = 1 \iff \forall y, g(x,y) = 1 \end{aligned}$$

Define

$$\mathbf{PH} = \bigcup_k \mathbf{\Sigma^P_k} \cup \mathbf{\Pi^P_k}$$

```
complexity-class:ph
complexity-class-ph
```

If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{P} = \mathbf{PH}$

Induction on k:

- $\mathbf{\Sigma^P_1}$ by assumption. $\mathbf{\Pi^P_1} = P$ since we can negate, calculate in polynomial time, negate again.
- If $f \in \Sigma^P_{k+1}$, then there exists $g \in \Pi^P_k$ such that $\forall x, f(x) = 1 \iff \exists y, g(x,y) = 1$. By induction hypothesis, $g \in \mathbf{P}$. So $f \in \mathbf{NP} = \mathbf{P}$. Similarly, $\mathbf{\Pi^P_{k+1}} = \mathbf{P}$.

```
complexity-class:p complexity-class:np complexity-class:ph
polynomial-hierarchy-p-np-collapse
```

Polynomial hierarchy collapse

If $\mathbf{\Sigma_k^P} = \mathbf{\Sigma_{k+1}^P}$ or $\mathbf{\Sigma_k^P} = \mathbf{\Pi_k^P}$, then $\mathbf{PH} = \mathbf{\Sigma_k^P}$.

Proof. Prove $\mathbf{\Sigma}_\ell^{\mathbf{P}} = \mathbf{\Pi}_\ell^{\mathbf{P}} = \mathbf{\Sigma_k^P}$ by induction on ℓ. The trick is to use the induction hypothesis to turn $\exists\forall$ or $\forall\exists$ into $\forall\forall$ or $\exists\exists$ and then collapse the quantifiers. $\square$

```
complexity-class:ph
polynomial-hierarchy-collapse
```

Complexity class **PSPACE**

$f \in \mathbf{PSPACE}$ iff f can be computed by a Turing machine using a polynomial amount of tape.

```
complexity-class:pspace
complexity-class-pspace
```

$\mathbf{NP} \subseteq \mathbf{PSPACE}$

Assume $f(x) = 1 \iff \exists y, g(x, y) = 1$ where $g \in \mathbf{P}$. It takes a polynomial amount of space to iterate through all possible y and a polynomial amount of space to compute $g(x, y)$ for each y. Hence it takes a polynomial amount of space to compute f.

```
complexity-class:np complexity-class:pspace
np-subset-pspace
```

$\mathbf{PH} \subseteq \mathbf{PSPACE}$

Prove that $\mathbf{\Sigma_k^P}, \mathbf{\Pi_k^P} \subseteq \mathbf{PSPACE}$ by induction on k, using the fact that it's possible to bruteforce search y of polynomial size such that $g(x, y) = 0$ or $g(x, y) = 1$ in polynomial time if $g \in \mathbf{P}$.

```
complexity-class:ph complexity-class:pspace
ph-subset-pspace
```

Complexity class **EXPTIME**

$f \in$ **EXPTIME** iff f can be computed by a Turing machine in time $\exp(O(n^k))$ for some k

```
complexity-class:exptime
complexity-class-exptime
```

PSPACE $\subseteq$ **EXPTIME**

If a Turing machine takes polynomial space to compute inputs of size n, say $p(n)$, then its **configuration** (combination of the state, the position on the tapes, the values of each cell on the tapes) goes through at most $p(n)^k\,|S|\,|A|^{kp(n)} = \exp(O(p(n)))$ possibilities. Further, if it went through one possibility twice, it would loop. Hence the computation takes exponential time.

```
complexity-class:pspace complexity-class:exptime
pspace-subset-exptime
```

Complexity class **NEXPTIME**

$f \in$ **NEXPTIME** iff there is a polynomial p and a function $g \in$ **EXPTIME** such that, for every x, $f(x) = 1$ iff $\exists y \in \{0,1\}^{p(|x|)}, g(x,y) = 1$.

```
complexity-class:nexptime
complexity-class-nexptime
```

Complexity class **EXPSPACE**

$f \in$ **EXPSPACE** iff f can be computed using tapes of length $\exp(O(n^k))$ for some k

```
complexity-class:expspace
complexity-class-expspace
```

Hierarchy of the basic complexity classes

$$\begin{gathered}\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \\ \subseteq \\ \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME} \subseteq \mathbf{EXPSPACE}\end{gathered}$$

`complexity-class:pspace complexity-class:pspace complexity-class:exptime`
`basic-complexity-classes-hierarchy`

Circuit

A *circuit* is a directed acyclic graph (DAG) such that each vertex is labelled as either an *input*, an **AND** gate, an **OR** gate or a **NOT** gate.

- An input is a vertex of in-degree 0.
- A **NOT** gate has in-degree 1.
- All vertices of in-degree > 1 are **AND** or **OR** gates.
- Vertices of out-degree 0 are outputs.

The value at an **AND**/**OR** is the min / max of its predecessors. The value at a **NOT** is $1 - x$ where x is the value at its predecessor.

`circuit`
`circuit-def`

Fan-in of a circuit

The fan-in of a circuit is the maximum in-degree of any **AND** or **OR** gate.

`circuit`
`fan-in-def`

Straight-line computations

A straight-line computation of $f : \{0,1\}^n \to \{0,1\}$ of length m is a sequence of functions $f_1, \dots, f_m$ starting with $f_i(x) = x_i$ for $i = 1, \dots, n$, ending with $f_m = f$, and for each $i > n$ there are some $j_1, \dots j_k < i$ such that either

$$f_i = f_{j_1} \wedge \cdots \wedge f_{j_k}$$
$$f_i = f_{j_1} \vee \cdots \vee f_{j_k}$$

This is the same as taking intersections and unions of half-spaces in an hypercube in order to get to some set.

`straight-line-computation`
`straight-line-computation-def`

The smallest size of a circuit computing f is the shortest length of a straight line computing f

A straight-line computation is the same as a circuit whose vertices have been totally ordered in a way that respects its edges.

```
circuit straight-line-computation
straight-line-computation-circuit
```

Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size exponential in n.

For every possible input x, build a circuit that recognises x (using at most n **AND** gates and n **NOT** gates) and outputs $f(x)$ if it's recognised. Then take a giant **OR** gate of all of those. This circuit has size at most 2^{n+1} and computes f since

$$f(x_1) \wedge (x_1 = x_i) \vee \dots f(x_{2^n}) \wedge (x_{2^n} = x_i) = f(x_i)$$

```
circuit
circuit-exponential
```

Family of circuits that each computes the output of a Turing machine on inputs of a given size

Let f be a function computed by a k-tapes Turing machine T in time $p(n)$ for inputs of size n. Then there is a family C_n of circuits such that C_n computes f for inputs of size n and

$$|C_n| = O(p(n)^{k+2})$$

Proof. WLOG assume the alphabet is $\{0,1\}$. Encode the configuration of the machine in $|S|$ variables for the state, $2kp(n)$ variables for the position of the heads, $2kp(n)$ variables for the values of the reachable cells. The transition function has $k + |S|$ inputs, hence can be computed in a circuit of size $O(1)$. Updating the variables takes a circuit of size $O(p(n))$. Hence the whole circuit has size $O(p(n)^2)$. □

```
circuit turing-machine
turing-machine-to-circuits
```

Complexity class **P/poly**

$f \in$ **P/poly** if one (hence all) of the following holds

- There is a family C_n of polynomial-size circuits such that $C_{|x|}$ computes $f(x)$.
- There is a polynomial p and a sequence y_n with $|y_n| = p(n)$ and a function $g \in$ **P** such that $f(x) = 1 \iff g(x, y_{|x|}) = 1$. y_n should be thought of as an "advice string" to help compute f.
- There is a sequence of Turing machines T_n and a polynomial p such that T_n has $\leq p(n)$ states and computes $f(x)$ when $|x| = n$.

```
complexity-class:p-poly
complexity-class-p-poly
```

The three definitions of **P/poly** are equivalent

- Family of circuits $\implies$ advice string. Let y_n be an encoding of C_n and let $g(x,y)=1$ if the circuit encoded by y outputs 1 with input x.
- Advice string $\implies$ family of circuits. Let C'_n compute g and make C_n to be C'_n with the last $p(n)$ inputs set to y_n.
- Advice string $\implies$ family of TMs. Let T compute g and let T_n be a Turing machine that prints out y_n and then uses T to compute $g(x,y_n)$.
- Family of TMs $\implies$ advice string. Let y_n be an encoding of T_n and let $g(x,y)=1$ if the Turing machine encoded by y outputs 1 with input x (g is encoded by a universal TM).

`complexity-class:p-poly`
`complexity-class-p-poly-alt`

P-uniformity

A family of circuits C_n is **P**-uniform if there is an algorithm that generates it in polynomial-time (in n).

`p-uniformity`
`p-uniformity-def`

If $\mathbf{P}=\mathbf{NP}$, then search problems are equivalent to decision problems

If $\mathbf{P}=\mathbf{NP}$ and $f\in\mathbf{NP}$, say $f(x)=1 \iff \exists y, |y| = p(|x|)$ and $g(x,y)=1$ where $g\in\mathbf{P}$, then there is some polynomial-time algorithm h such that if $f(x)=1$ then $g(x,h(x))=1$.

Proof. For each i, let g_i be the function with input x and u_i where $|u_i|=i$ and output whether it can be extended by a 1. Now calculate $u_1=g_0(x)$, $u_2=g_1(x,u_1)$, $u_3=g_1(x,u_1,u_2)$, etc. At the end we obtain $h(x)=(u_1,\dots,u_{p(|x|)})$ such that $g(x,u)=1$.
Each g_i is obviously in $\mathbf{NP}=\mathbf{P}$, so h can be computed in polynomial time (assuming the g_i are uniformly in polynomial time). $\square$

`complexity-class:p complexity-class:np`
`p-np-search-decision-problem`

If $\mathbf{NP}\subseteq\mathbf{P}/\mathbf{Poly}$, and $f\in\mathbf{NP}$, then we can compute certificates for f using polynomial-size circuits

Say $f(x)=1 \iff \exists y, g(x,y)=1$ where $g\in\mathbf{P}$. Define $g_i(x,u_1,\dots,u_i)=1 \iff \exists y, g(x,u_1,\dots,u_i,1,y)$. $g_i\in\mathbf{NP}\subseteq\mathbf{P}/\mathbf{Poly}$, hence find a polynomial-size circuit family $C_{i,n}$ that computes g_i. Now put circuits $C_{0,n},\dots,C_{p(n),n}$ together as follows:

- $C_{0,n}$ takes inputs $x_1,\dots,x_n$ and outputs u_1
- $C_{1,n}$ takes inputs $x_1,\dots x_n,u_1$ and outputs u_2
- ...

The resulting circuit C_n is such that if $\exists y, g(x,y)=1$ then $g(x,C_n(x))=1$.

`complexity-class:np complexity-class:p-poly`
`np-subset-p-poly-certificate`

Karp-Lipton theorem

`complexity-class:np complexity-class:p-poly complexity-class:ph karp-lipton`

If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{\Sigma_2^P} = \mathbf{\Pi_2^P}$.

Proof. By symmetry, it's enough to prove $\mathbf{\Pi_2^P} \subseteq \mathbf{\Sigma_2^P}$. Let $f \in \mathbf{\Pi_2^P}$ and let $g \in \mathbf{NP}, h \in \mathbf{P}$ be such that

$$f(x) = 1 \iff \forall y, g(x,y) = 1$$
$$g(x,y) = 1 \iff \exists z, h(x,y,z) = 1$$

By the existence of polynomial certificates when $\mathbf{NP} \subseteq \mathbf{P/poly}$, find a polynomial-size circuit family C_n such that $g(x,y) = 1 \implies h(x,y,C_n(x,y))$. Then

$$f(x) = 1 \implies \exists C, \forall y, h(x,y,C_n(x,y)) = 1$$

The converse is true by assumption. Hence $f \in \mathbf{\Sigma_2^P}$. □

For every k, there's a boolean function f that can be computed by a circuit family of size n^{k+1} but not by a circuit family of size n^k.

`circuit boolean-function-precise-polynomial`

TODO

For every k, there is a boolean function $f \in \mathbf{\Sigma_4^P}$ that cannot be computed by a family of circuits of size n^k

`circuit complexity-class:ph boolean-function-sigma-four-not-polynomial`

For sufficiently large n, the lemma gives us $f'_n : \{0,1\}^n \to \{0,1\}$ that can be computed by a circuit of size n^{k+1} but not by a circuit of size n^k. Choose an ordering of the circuits of size $\leq n^{k+1}$ that's computable in polynomial time. Let C_n be the first circuit in this ordering such that no circuit of size $\leq n^k$ computes the same function as C_n. Let $f(x) = C_{|x|}(x)$. Then

$$\begin{aligned} f(x) = 1 \iff & \exists C_n, |C_n| \leq n^{k+1} \text{ and } C_n(x) = 1 \\ & \text{and } \forall D, |D| \leq n^k, \exists y, C_n(y) \neq D(y) \\ & \text{and } \forall E < C_n, \exists F, |F| \leq n^k, \forall z, E(z) = f(z) \end{aligned}$$

The $\exists\forall\exists\forall$ shows that $f \in \Sigma_4^P$.

Kannan's theorem

`complexity-class:np complexity-class:ph circuit kannan`

For every k, there is a function $f \in \mathbf{\Sigma_2^P} \cap \mathbf{\Pi_2^P}$ that cannot be computed by a circuit family of size n^k.

Proof. If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{\Sigma_2^P} \cap \mathbf{\Pi_2^P}$ by Karp-Lipton. So the function in $\mathbf{\Sigma_4^P}$ that cannot be computed by a circuit of size $\leq n^{k+1}$ does the job.
If $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, then there is some $f \in \mathbf{NP} \subseteq \mathbf{\Sigma_2^P} \cap \mathbf{\Pi_2^P}$ that cannot be computed by *any* polynomial-size circuit family. □

Complexity class **L**

$f \in \mathbf{L}$ iff f can be computed with a logarithmic amount of memory. Formally, $f \in \mathbf{L}$ iff it can be computed by some Turing machine with a read-only input tape, a write-only output tape and worktapes of size $O(\log n)$ for inputs of size n.

`complexity-class:l`
`complexity-class-l`

Complexity class **NL**

$f \in \mathbf{NL}$ iff f can be non-deterministically computed with a logarithmic amount of memory. Formally, $f \in \mathbf{NL}$ iff it can be computed by some non-deterministic Turing machine with a read-only input tape, a write-only output tape and worktapes of size $O(\log n)$ for inputs of size n.

`complexity-class:nl`
`complexity-class-nl`

$\mathbf{NL} \subseteq \mathbf{P}$

Let $f \in \mathbf{NL}$, T be a non-deterministic Turing machine that computes f in log-space and G be the *configuration graph* of T. Then $f(x) = 1$ iff there is a directed path in G from the initial configuration to one such that T has halted with output 1. Since T runs on $O(\log n)$ space, the number of vertices of G is polynomial in n ($|V(G)| \leq 2^{O(\log n)}|S| = n^{O(1)}$). But note that **REACHABILITY**, the problem of determining whether there is a directed path from a vertex x to a subset S of vertices on a directed graph is easily seen to be in $\mathbf{P}$: just brute force search to find the neighbours.

`complexity-class:nl complexity-class:p`
`nl-subset-p`

Low depth-computation classes

The class $\mathbf{NC}^i$ where $i \in \mathbb{N}$ consists of all functions that can be computed by a family of circuits of polynomial size, fan-in 2 and depth $O(\log^i n)$, where *depth* of a circuit is the length of the longest directed path in the associated DAG.

$\mathbf{AC}^i$ is like $\mathbf{NC}^i$ except that we allow unbounded fan-in.

We then define

$$\mathbf{NC} = \bigcup_i \mathbf{NC}^i, \mathbf{AC} = \bigcup_i \mathbf{AC}^i$$

`complexity-class:nc complexity-class:ac`
`complexity-class-nc-ac`

log-space uniformity

$f \in \mathbf{u} - \mathbf{NC}^i$ iff f can be computed by a family of circuits of polynomial size, fan-in 2 and depth $O(\log^i n)$ that can be generated in log-space.

```
complexity-class:nc
log-space-uniformity-def
```

$\mathbf{AC} = \mathbf{NC}$

Obviously, $\mathbf{NC}^i \subseteq \mathbf{AC}^i$. But we also have $\mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$ since a circuit of fan-in k can be replace by a circuit of fan-in 2 which is at most $\log k$ bigger by replacing each gate of in-degree d by $\log d$ gates of in-degree 2.

```
complexity-class:nc complexity-class:ac
ac-eq-nc
```

Complexity classes $\mathbf{RP}$, $\mathbf{co} - \mathbf{RP}$ and $\mathbf{ZPP}$

$f \in \mathbf{RP}$ (randomised polynomial time) iff there is a polynomial p and a function $g \in \mathbf{P}$ such that if $|x| = n$ and $m = p(n)$ then

$$\mathbb{P}_{y \in \{0,1\}^m}(g(x,y) = 1) \begin{cases} = 0 & \text{if } f(x) = 0 \\ \geq \frac{1}{2} & \text{if } f(x) = 1 \end{cases}$$

$f \in \mathbf{co} - \mathbf{RP}$ iff $\neg f \in \mathbf{RP}$.

$\mathbf{ZPP}$ (zero-error probabilistic polynomial time) is $\mathbf{RP} \cap \mathbf{co} - \mathbf{RP}$

```
complexity-class:rp complexity-class:co-rp complexity-class:zpp
complexity-class-rp-co-rp-zpp
```

How to improve the accuracy of an algorithm in $\mathbf{RP}$

Run the algorithm many times. Say we're computing f and $g \in \mathbf{P}$ is such that

$$\mathbb{P}(g(x,y) = 1) \begin{cases} = 0 & \text{if } f(x) = 0 \\ \geq \frac{1}{2} & \text{if } f(x) = 1 \end{cases}$$

Then if $y_1, \ldots y_k$ are independent samples, we get

$$\mathbb{P}(\exists i, g(x, y_i) = 1) \begin{cases} = 0 & \text{if } f(x) = 0 \\ \geq 1 - 2^{-k} & \text{if } f(x) = 1 \end{cases}$$

```
complexity-class:rp
rp-improved-accuracy
```

Complexity class **BPP**

$f \in \mathbf{BPP}$ (bounded-error probabilistic polynomial time) iff there is a polynomial p and a function $g \in \mathbf{P}$ such that if $|x| = n$ and $m = p(n)$ then

$$\mathbb{P}_{y \in \{0,1\}^m}(g(x,y) = f(x)) \geq \frac{2}{3}$$

complexity-class:bpp
complexity-class-bpp

How to improve the accuracy of an algorithm in **BPP**

Run the algorithm many times. Say we're computing f and $g \in \mathbf{P}$ is such that

$$\mathbb{P}(g(x,y) = f(x)) \geq \frac{2}{3}$$

Take $y_1, \dots y_k$ independent samples. Compute $g(x, y_1)$, ..., $g(x, y_k)$. Output the majority. The probability of getting the wrong answer is at most $\exp(-\frac{k}{48})$ by Chernoff.

complexity-class:bpp
bpp-improved-accuracy

$\mathbf{BPP} \subseteq \mathbf{P/poly}$

If $k \geq 48n$, then the probability that the majority is wrong is $< 2^{-n}$. Therefore there exist $y_1, \dots, y_k$ such that for **every** x the majority vote is correct. $y_1 \dots y_k$ serves as an advice string, together with the function that computes the majority vote.

complexity-class:bpp complexity-class:p-poly
bpp-subset-p-poly