A **verifier** for A is a TM V such that $A = \{w | \exists c V(\langle w, c\rangle)\}$.
Running time of V is is a function $T_V(|w|)$ (ignore the length of c).

NP is the class of languages with polytime verifiers.
Or, the class of languages decided by polytime nondet TMs.

co**NP** is the set of L's such that $\overline{L} \in \mathbf{NP}$.

Cook's Theorem: SAT $\in \mathbf{NP} \iff \mathbf{P} = \mathbf{NP}$

A is **polytime mapping reducible** to B, $A \leq_p B$ if a polytime f exists such that $w \in A \iff f(w) \in B$.

If $A \leq_p B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.

CLIQUE$= \{\langle G, k\rangle |\ G$ has a k-clique $\}$

3SAT $\leq_p$ CLIQUE:

Given $\phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k$ in 3CNF, let G_ϕ denote the graph where each clause $(l_1 \vee l_2 \vee l_3)$ is represented by three nodes, with no edges between them, and no edges between opposite literals in different clauses, and all the other edges.

Then, ϕ is satisfiable $\iff$ G_ϕ has a k-clique.

Let $f(\langle \phi \rangle) = \langle G_\phi, k\rangle$.

f is clearly polytime, so the result follows.

B is **NP**-complete if it is in **NP**, and for every $A \in \mathbf{NP}$, $A \leq_p B$.

Thm: If B is **NP**-complete, and $B \in \mathbf{P}$, then **P**=**NP**.

Thm: If B is **NP**-complete, and $B \leq_p C$, for $C \in \mathbf{NP}$, then C is **NP**-complete.

Theorem [Cook-Levin]: SAT is **NP**-complete.

Proof: If $A \in \mathbf{NP}$, then $A = L(N)$, where N is a nondet TM that runs in time $n^k + k$, for some $k \in \mathbb{N}$.

$\therefore$ each configuration of N can be described with a string of length $n^k + c$ (k, c constants).

Any branch of computation has length at most $n^k + k$. So any branch can be described with an $(n^k + k) \times (n^k + c)$ table.

To determine whether N accepts w, we must determine if an accepting table exists.

We must construct a reduction that works as follows:

> On input w, we build ϕ_w such that $w \in A \iff \phi_w$ is satisfiable; i.e., $f(\langle w \rangle) = \langle \phi_w \rangle$.

We construct ϕ_w based on the accepting table insight: the variables are x_{ijs}, where x_{ijs} is true if position (i, j) in the table contains symbol $s \in Q \cup \Gamma \cup \{\#\}$.

Then,

$$\phi_w = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

where the first ensures that only one symbol is assigned to each cell, the second ensures that the first row of the table is the initial configuraion, i.e.,

$$\# q_0 w_1 w_2 \dots w_n \sqcup \sqcup \dots \sqcup \#$$

ϕ_{accept} ensures that the last row is an accepting row. (Here we need a trick; once we get an accepting row, we repeat it until we reach the $n^k + c$ many rows.) It is easy to give it:

$$\bigvee_{1 \leq i,j \leq n^k+k} x_{ijq_{\text{accept}}}$$

ϕ_{move} ensures that each row follows from the previous by a legal move (or, in the case that the previous row was accepting, the next one is its copy).

This is easy to implement, because each row is exactly like the previous one, except in the four squares surrounding the state.

Claim: ϕ_w can be constructed in polytime in $|w|$.

QED.

Corollary: 3SAT is **NP**-complete.

Proof: ϕ_w can be given in 3CNF, using tricks such as $(a_1 \vee a_2 \vee \ldots \vee a_n)$ is equivalent to:

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z}_1 \vee a_3 \vee z_2) \wedge (\overline{z}_2 \vee a_4 \vee z_3) \wedge \ldots \wedge (\overline{z}_{n-3} \vee a_{n-1} \vee a_n)$$

Cor: CLIQUE is **NP**-complete.

Now we can show that many others are **NP**-complete:

VERTEX-COVER
HAMPATH
SUBSET-SUM
TRAVELLING-SALESMAN

SPACE($f(n)$) and NSPACE($f(n)$)

SAT $\in$ SPACE(n)

Savitch's Theorem: NSPACE($f(n)$) $\subseteq$ SPACE($f^2(n)$)

Proof: Given two configurations C_1, C_2 of nondet N, to check if $C_1 \vdash^t C_2$, check if $\exists C$ such that $C_1 \vdash^{t/2} C \wedge C \vdash^{t/2} C_2$.

If C is any configuration on w ($|w| = n$), then $|C|$ is $O(f(n))$, and any branch will take at most $2^{O(f(n))}$ many steps, so need $\log(2^{O(f(n))}) = O(f(n))$ many bits to store t.

Run the above procedure with $C_1 = C_{\text{start}}, C_2 = C_{\text{accept}}$, and $t = 2^{O(f(n))}$.

Depth of recursion is $\log(2^{O(f(n))}) = O(f(n))$; so we have $O(f(n))$ configurations on the tape at any one time, each of size $O(f(n))$.

$\therefore$ the above procedure requires $O(f^2(n))$ deterministic space.

PSPACE $= \bigcup_k$ SPACE(n^k) and **NPSPACE** $= \bigcup_k$ NSPACE(n^k)

By Savitch's Theorem, **PSPACE** = **NPSPACE** !!!

A TM that runs in space $\log(n) \leq f(n)$ has at most $n \cdot 2^{O(f(n))} = 2^{\log(n)+O(f(n))} = 2^{O(f(n))}$ many different configurations.

A halting TM never repeats a configuration, so it runs in time $\leq$ number of configurations.

EXPTIME $= \bigcup_k$ TIME(2^{n^k})

$\therefore$ **P** $\subseteq$ **NP** $\subseteq$ **PSPACE** = **NPSPACE** $\subseteq$ **EXPTIME**

A Quantified Boolean Formula, is defined as the usual Boolean formulas, together with two cases: if ϕ is a QBF, then so is $\exists x \phi$ and $\forall x \phi$.

Semantics: $[\forall/\exists] x \phi \equiv \phi(x=0)[\wedge/\vee]\phi(x=1)$

A QBF is fully quantified if it has no free variables.

TQBF $= \{\langle\phi\rangle |\ \phi$ is a true QBF $\}$

Thm: TQBF is **PSPACE**-complete

Proof: TQBF is in **PSPACE**; just try all the assignments, overwriting the previous try.

Let A be any language decided by TM M in space n^k. We show $A \leq_p$ TQBF.

Let $\phi_{c_1,c_2,t}$ be a QBF, where:

c_1, c_2 are two groups of variables representing configurations of M on a particular w ($|w| = n$),

and t is a parameter denoting the number of steps of M, so $t \leq 2^{O(n^k)}$.

We show how to construct $\phi_{c_1,c_2,t}$.

$\phi_{c_1,c_2,1}$ is quantifier-free, and formalizes the notion that "$(c_1 = c_2)$"$\vee$"$(c_1 \vdash c_2)$" ("$c_1 \vdash c_2$" can be formalized in the style of ϕ_{move}).

For $t > 1$, we build our formula recursively as follows: $\phi_{c_1,c_2,t}$ is

$$\begin{aligned} &\exists m_1[\phi_{c_1,m_1,\lceil \frac{t}{2}\rceil} \wedge \phi_{m_1,c_2,\lceil \frac{t}{2}\rceil}] \\ &\equiv \exists m_1 \forall x \forall y[((x = c_1 \wedge y = m_1) \vee (x = m_1 \wedge y = c_2)) \supset \phi_{x,y,\lceil \frac{t}{2}\rceil}] \end{aligned}$$

We can say that:

$$w \in A \iff \langle \phi_{c_{\text{start}},c_{\text{accept}},2^{O(n^k)}} \rangle \in \text{TQBF}$$

In other, words, our polytime reduction f is:

$$f(w) = \langle \phi_{c_{\text{start}},c_{\text{accept}},2^{O(n^k)}} \rangle$$

Games

Formula game: $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\psi]$.

FORMULA-GAME $= \{\langle \phi \rangle |$ Player E has winning strategy $\}$

A winning strategy (for player E) is a choice of x_1 such that no matter what player A chooses for x_2, E can choose a value for x_3, such that no mater what A chooses for x_4, etc., ϕ is made true by the resulting assignment.
FORMULA-GAME is **PSPACE**-complete.

Geography: Peoria $\rightarrow$ Amherst $\rightarrow$ Tucson $\rightarrow$ Nashua $\ldots$

GG (Generalized Geography) is a directed graph, with a designated starting node, and Player I selects an edge out of the start node, and then Player II selects an edge out of that node, etc. The looser has nowhere to go (cannot repeat nodes).
GG is **PSPACE**-complete.

For log-space classes, we consider the input tape, which is read-only, and we bound the (second) work-tape:

$\mathbf{L} = \mathrm{SPACE}(\log(n))$

$\mathbf{NL} = \mathrm{NSPACE}(\log(n))$

Idea: computation with pointers to the input.

Ex. $\{0^n 1^n | n \geq 1\} \in \mathbf{L}$; maintain two counters, each of size $\log(n)$, to count the number of zeros and ones.

Ex. $\mathrm{PATH} = \{\langle G, s, t\rangle | s \rightsquigarrow t\} \in \mathbf{NL}$.

Open Problems: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ and $\mathbf{L} \stackrel{?}{=} \mathbf{NL}$.

Logspace transducer: TM M with read-only input tape, write-only output tape, and logspace bounded work-tape. f is logspace computable if it is computed by a logspace transducer.

$A \leq_L B$ if A is mapping reducible to B by a logspace f.

Thm: $A \leq_L B$, and $B \in \mathbf{L}$, then $A \in \mathbf{L}$.

Proof: $f(w)$ maybe too long to fit in $\log(|w|)$-space. Trick: compute each bit of $f(w)$ only when required. M_A keeps track of where the head of M_B is on $f(w)$, and everytime the head of M_B moves, M_A recomputes $f(w)$ until it gets the appropriate bit.

Thm: PATH is **NL**-complete.

Proof: Let $A = L(N)$ where N is a nondet logspace TM.

Let (G_w, s_w, t_w) be the transition graph of N on w; i.e., the nodes are configurations, and (C_1, C_2) is an edge iff $C_1 \vdash C_2$.

$w \in A$ iff there is a path from s_w (inital configuration) to t_w *the* accepting configuration (*which we can force to be unique*).

Thus, our reduction is: $f(w) = \langle (G_w, s_w, t_w) \rangle$, and it remains to show that f is logspace computable.

Cor: $\mathbf{NL} \subseteq \mathbf{P}$.

Proof: A logspace transducer is also a polytime transducer. Also, PATH $\in \mathbf{P}$.

(Surprising) Thm [Immerman-Szelepcsényi]: $\mathbf{NL} = \mathrm{co}\mathbf{NL}$.

Proof: Enough to show co$\mathbf{NL} \subseteq \mathbf{NL}$.

Thus, we show that $\overline{\text{PATH}} \in \mathbf{NL}$.

First, we compute (nondet) $c =$ number of nodes reachable from s.

To this end, define $A_i =$ set of nodes of distance at most i from s. So $A_0 = \{s\}$, $A_i \subseteq A_{i+1}$.

A_m contains all the nodes reachable from s.

Then, $c_i = |A_i|$, and $c = c_m = |A_m|$.

We compute c_{i+1} from c_i as follows:

To determine if $v \in A_{i+1}$, we go through each node u of G, and test if it is in A_i (to test, we guess a path from s to u of length at most i, and check that it is indeed a path).

As we have c_i, we know how many u's we should get (we can only get too few u's, in which case the branch dies).

For each $u \in A_i$, we check if (u, v) is an edge.

Once we have c, we (nondet) select exactly c nodes reachable from s, not including t, and check (nondet) that they are reachable.

The remaining nodes, including t, are not reachable so we accept.

QED

$\mathbf{L} \subseteq \mathbf{NL} = \mathrm{co}\mathbf{NL} \subseteq \mathbf{P}$

Aside:

A grammar is context-sensitive if it has rules of the form $\alpha \longrightarrow \beta$, where $|\alpha| \leq |\beta|$.

A grammar is a rewriting system if it has arbitrary rules of the form $\alpha \longrightarrow \beta$.

Thm: Let CSL be the class of languages with context-sensitive grammars. Then, CSL $\subseteq$ NSPACE(n).

Thm: Let RS be the class of languages for which there are rewriting systems. Let TR be the class of Turing-recognizable languages. The, RS $=$ TR.

Hierarchy Theorems

A function $f : \mathbb{N} \longrightarrow \mathbb{N}$, where $f(n) \geq \log(n)$, is called space constructible if the function that maps 1^n to the binary representation of $f(n)$ is computable in space $O(f(n))$.

Eg., $\log(n), n\log(n), n^2, n^k$ for any k, etc.

Space Hierarchy Theorem: For any space constructible function f, there exists a language A that is decidable in space $O(f(n))$ but not in space $o(f(n))$.

Recall that:

1. $g(n) \in O(f(n))$ if $\exists c, n_0$ s.t. $\forall n \geq n_0$, $g(n) \leq c \cdot f(n)$.
2. $g(n) \in o(f(n))$ if $\lim_{n\to\infty} \dfrac{g(n)}{f(n)} = 0$.

Proof: The following $O(f(n))$ space algorithm B decides a language A that is not decidable in $o(f(n))$.

B is given as follows: on input w,

1. Let $n = |w|$.
2. Compute $f(n)$, and mark off this much tape.
   (If later stages attempt to use more, *reject*.)
3. If $w \neq \langle M \rangle 10^*$, *reject*.
4. Simulate M on w, counting number of steps.
   If steps exceed $2^{f(n)}$, *reject*.
5. If M accepts, *reject*.
   If M rejects, *accept*.

Machine B is a decider.

Let $A = L(B)$, so A is decided in space $O(f(n))$.

Next we show that A cannot be decided in space $o(f(n))$.

Suppose that M decides A in space $g(n) \in o(f(n))$.

$\exists n_0$, such that $\forall n \geq n_0$, $f(n) > g(n)$. Thus, on input $w = \langle M \rangle 10^{n_0}$ (which has length $> n_0$), B can simulate the computation of M on w without exceeding the alloted space.

But then B will do the opposite of M, so M cannot decide A, which contradicts our assumption.

Cor: For any two functions $g, f : \mathbb{N} \longrightarrow \mathbb{N}$, where $g(n) \in o(f(n))$,

$$\mathrm{SPACE}(g(n)) \subsetneq \mathrm{SPACE}(f(n))$$

In particular, for $0 \leq c_1 < c_2$, $\mathrm{SPACE}(n^{c_1}) \subsetneq \mathrm{SPACE}(n^{c_2})$

Cor: $\mathbf{NL} \subsetneq \mathbf{PSPACE}$.

Proof: Savitch's Theorem shows that $\mathrm{NL} \subseteq \mathrm{SPACE}(\log^2(n))$. On the other hand, we now know that $\mathrm{SPACE}(\log^2(n)) \subsetneq \mathrm{SPACE}(n)$.

Cor: $\mathbf{PSPACE} \subsetneq \mathbf{EXPSPACE}$.

Proof: For any constant k, $\mathrm{SPACE}(n^k) \subseteq \mathrm{SPACE}(n^{\log n})$, which is strictly contained in $\mathrm{SPACE}(2^n)$.

A function $t : \mathbb{N} \longrightarrow \mathbb{N}$, where $t(n) \geq n \log(n)$, is called time constructible if the function that maps 1^n to the binary representation of $t(n)$ is computable in time $O(t(n))$.

Time Hierarchy Theorem: For any time constructible function t, there exists a language A that is decidable in time $O(t(n))$, but not in time $o(t(n)/\log(t(n)))$.

Proof: Consider the following $O(t(n))$ algorithm B, which on input w works as follows:

1. $n = |w|$.
2. Compute $t(n)$, and store the value $\lceil t(n)/\log(t(n)) \rceil$ in a binary counter. Decrement this counter before steps 3.,4., and 5. If it hits 0, *reject*.
3. If $w \neq \langle M \rangle 10^*$ for some TM M, *reject*.
4. Simulate M on w.
5. If M accepts, *reject*.
   If M rejects, *accept*.

Let $A = L(B)$, so A is decidable in time $O(t(n))$.

To show that A is *not* decidable in time $o(t(n)/\log(t(n)))$, we use an argument similar to the proof of the space hierarchy theorem:

Assume M decides A in time $g(n) = o(t(n)/\log(t(n)))$, and get a contradiction by considering an input $\langle M \rangle 10^n$ for some $n \geq n_0$.

Cor: If $t, s : \mathbb{N} \longrightarrow \mathbb{N}$, where $t(n) = o(s(n)/\log(s(n)))$, and s is time constructible, then: $\mathrm{TIME}(t(n)) \subsetneqq \mathrm{TIME}(s(n))$

Cor: If $1 \leq c_1 < c_2$, then $\mathrm{TIME}(n^{c_1}) \subsetneqq \mathrm{TIME}(n^{c_2})$

Cor: $\mathbf{P} \subsetneqq \mathbf{EXPTIME}$.

Recall that regular expressions are built up from $\emptyset, \varepsilon, a \in \Sigma$, by using the regular operations $\cup, \circ, *$.

The language $\mathrm{EQ_{REX}}$ given by
$\{\langle R, S\rangle |\ R, S$ are reg. exp. and $L(R) = L(S)\}$ is in **PSPACE**.

Introduce an exponentiation operator, $R^k = R \uparrow k = R \circ R \circ \cdots \circ R$.

Let $\mathrm{EQ_{REX\uparrow}}$ be like the set above except we allow regular expressions with exponentiation.

Thm: $\mathrm{EQ_{REX\uparrow}}$ is EXPSPACE-complete.

Cor: $\mathrm{EQ_{REX\uparrow}}$ is *intractable*.

Proof of Cor: If it were in **P**, then it would certainly be in **PSPACE**, but by completness it would follow that **EXPSPACE** $\subseteq$ **PSPACE**, contradicting the separation we obtained from the Hierarchy Theorems.

Relativization

An **oracle** is a language A. An **oracle TM** M^A is an ordinary TM with an extra tape called the **oracle tape**.

M writes a string on the oracle tape, queries the oracle, and gets the answer in one step.

Let $\mathbf{P}^A$ and $\mathbf{NP}^A$ be the set of languages decidable with polytime det (respectively, nondet) TMs with an oracle for A.

Ex. $\mathbf{NP}, \textbf{co-NP} \subseteq \mathbf{P}^{\mathrm{SAT}}$.

Ex. NON-MIN-FORMULA =
$\{\langle\phi\rangle |\ \phi$ is not a minimal Boolean formula $\}$ is in $\mathbf{NP}^{\mathrm{SAT}}$.
(Non-equiv of Boolean formulas is in $\mathbf{NP}$, so equiv is in co$\mathbf{NP}$, so a nondet TM can guess a smaller ϕ' and test using the oracle that $\phi' \equiv \phi$.)

So, $\mathbf{NP}^{\mathrm{SAT}}$ is believed to be bigger than $\mathbf{NP}$.

Thm: There exist oracles A, B such that:
(i) $\mathbf{P}^A \neq \mathbf{NP}^A$ and (ii) $\mathbf{P}^B = \mathbf{NP}^B$

Proof: For (ii) let $B =$ TQBF. Then:

$$\mathbf{NP}^{\mathrm{TQBF}} \overset{(1)}{\subseteq} \mathbf{NPSPACE} \overset{(2)}{\subseteq} \mathbf{PSPACE} \overset{(3)}{\subseteq} \mathbf{P}^{\mathrm{TQBF}}$$

For (i), given any oracle A, define L_A as follows:

$$\{w | \exists x \in A \text{ such that } |x| = |w|\}$$

Clearly, $L_A \in \mathbf{NP}^A$.

We now present an A which ensures that $L_A \notin \mathbf{P}$.

Let $M_1, M_2, M_3, \ldots$, be a list of polytime oracle TMs.
Arrange it so that M_i runs in time n^i (how?)
We build A in stages to ensure that $L(M_i^A) \neq L_A$.

At **stage i** A is finite, so we pick n such that $n > |w|$ for all $w \in A$, and $2^n > n^i$ (running time of M_i).

We now extend A in such a way so that $L(M_i^A) \neq L_A$

Run $M_i(1^n)$: each time M_i queries the oracle with a string y, if $y \overset{?}{\in} A$ has been determined, we do nothing to A. (Note that all the strings whose status has been determined are shorter than n.)

If y's status is undetermined, we answer NO (i.e., declare $y \notin A$).

If $M_i(1^n) =$ yes, declare all remaining y, $|y| = n$, not in A. If $M_i(1^n) =$ no, find a y_0 such that $|y_0| = n$, and y_0 has not been queried (it must exist, because in n^i time M_i cannot query all $2^n > n^i$ y's of length n), and put y_0 in A. $\underline{M_i(1^n) = \text{yes iff } 1^n \notin L_A}$.

Circuit Complexity

A **Boolean circuit** is a collection of **gates** and **inputs** connected by **wires**. Cycles are not permitted. Gates take three forms: AND, OR, NOT.

To a Boolean circuit C with n input variables and k output gates we associate a function:

$$f_C : \{0,1\}^n \longrightarrow \{0,1\}^k$$

A **circuit family** C is an infinite list of circuits $(C_0, C_1, C_2, \ldots)$, where C_n has n input variables. We say that C decides a language A if

$$w \in A \iff C_n(w) = 1$$

where $n = |w|$.

The **size** of a circuit is the number of gates that it contains.

The **size complexity** of a circuit family C is $f(n)$ if $|C_n| \leq f(n)$ for all n.

The **depth** of a circuit is the length of the longest path from the inputs to an output gate.

The **circuit complexity** of a language is the size complexity of a minimal circuit family for that language. Same for **circuit depth complexity**.

Thm: Suppose that $A \in \mathrm{TIME}(t(n))$, then A has a circuit complexity $O(t^2(n))$.

Proof: Let M be the TM that decides A in time $t(n)$. For each n, build C_n to simulate M on inputs of size n: the gates of C_n are organized in $t(n)$ many rows. Each row is wired into the previous row so that it can compute the new configuration from the previous configuration. Each row has $O(t(n))$ many gates.

Let CIRCUIT-SAT be the language:

$$\{\langle C\rangle \mid C \text{ is a satisfiable boolean circuit }\}$$

Thm: CIRCUIT-SAT is **NP**-complete.

Proof: CIRCUIT-SAT$\in$ **NP**, and SAT is polytime many-one reducible to CIRCUIT-SAT.

Note that alternatively, we could have used the proof of the previous theorem to show that $\forall A \in \mathbf{NP}$, $A \leq_p$ CIRCUIT-SAT.

This would have given us a second proof of Cook's Thm. Now, we would show that CIRCUIT-SAT $\leq_p$ 3SAT to complete this proof.

P/poly is the class of languages with polynomial size circuits.

$\mathbf{NC}^i$ is the class of languages with polynomial size circuits of *bounded fan-in* and depth $O(\log^i(n))$.

$\mathbf{AC}^i$ is the class of languages with polynomial size circuits of *un-bounded fan-in* and depth $O(\log^i(n))$.

Let $\mathbf{NC} = \bigcup_i \mathbf{NC}^i$.

Two versions: uniform and non-uniform.

In the uniform version we require that circuit C_n be constructible by a TM in log-space on input 1^n.

Assume uniform unless otherwise stated, because:

Thm: Non-uniform $\mathbf{AC}^0$ contains undecidable languages.

Proof: Let $U = \{1^n |\ n_{\text{binary}} = \langle M, w\rangle \text{ s.t. } M \text{ accepts } w\ \}$. Let C_i be $\text{AND}(x_1, x_2, \ldots, x_i)$ if $1^i \in U$, and FALSE otherwise.

Thm: $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$.

Proof: An AND or an OR of un-bounded fan-in can be simulated with a balanced binary-tree of ANDs or ORs of fan-in 2.

Thm: Addition of two integers is in $\mathbf{AC}^0$.

Proof: Let $a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0$ be the two inputs (n-bit integers, where a_0, b_0 are the least significant bits).

The i-th bit of the sum is $a_i \oplus b_i \oplus c_i$, where c_i is the carry bit left over after summing the $(i-1)$ first bits.

$c_i = 1$ iff $\exists j < i$ such that $a_j = b_j = 1$ and $\forall k$, $j < k < i$, $a_k = 1 \vee b_k = 1$. This can be stated with:

$$c_i = \bigvee_{0 \leq j < i} \left[(a_j \wedge b_j) \wedge \bigwedge_{j<k<i} (a_k \vee b_k) \right]$$

Thm: Repeated addition is in $\mathbf{NC}^1$.

Proof: Using the previous theorem, add each pair, then add the resulting pairs, and so on. But this only shows that repeated addition is in $\mathbf{AC}^1$, which is a weaker claim.

To show it is in $\mathbf{NC}^1$, we first design an $\mathbf{NC}^0$ circuit which takes as input three integers, and returns two whose sum is the same as the sum of the original three.

$$a_i, b_i, c_i \mapsto x_i = \underbrace{a_i \oplus b_i \oplus c_i}_{\text{sum } \textit{without} \text{ carries}}, y_i = \underbrace{(a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)}_{\text{the carry}}$$

Each step reduces the number of integers to be added by 1/3. So in a logarithmic number of steps we are left with two integers. We now add those two as in the previous theorem.

Let **BF** be the class of languages recognizable by a family of polynomial size Boolean Formulas.

That is, $L \in \mathbf{BF}$ iff $\exists\ \phi_1, \phi_2, \phi_3, \ldots$, where ϕ_i has i variables, and a polynomial p such that $|\phi_i| \leq p(i)$, and

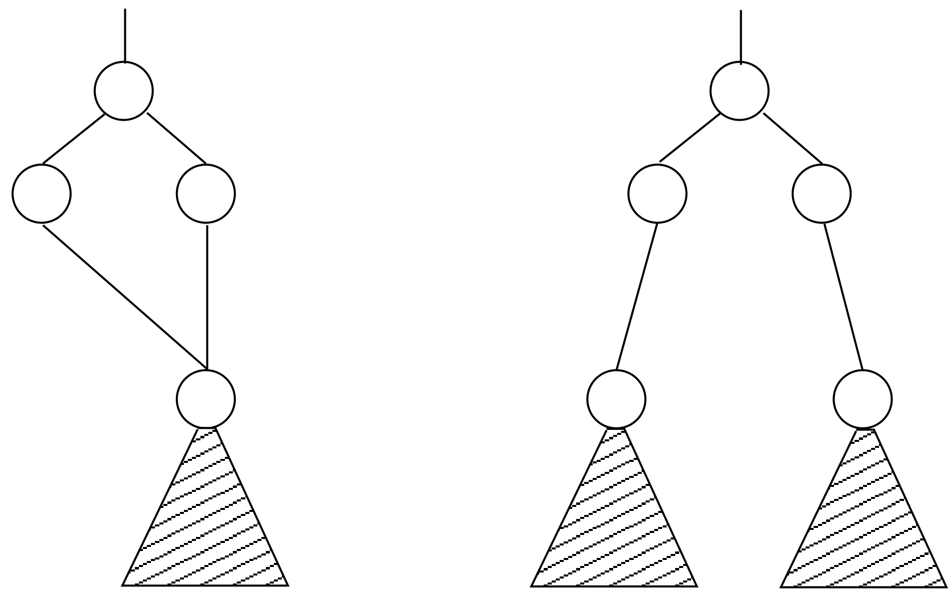
$$w \in L \iff \phi_{|w|}(w_1, w_2, \ldots, w_n) \text{ is true}$$

Thm: $\mathbf{NC}^1 = \mathbf{BF}$.

Proof: Suppose that we have an $\mathbf{NC}^1$ circuit. We can easily write it as a formula; the only issue is that a subcircuit can be reused many times.

We replicate reused subcircuits. When doing this, we go from level to level, at most doubling the size of the circuit at each step.

Since the depth is $O(\log(n))$, we know that this process can increase the size of the circuit by at most a polynomial factor.



For the other direction, suppose that we are given a Boolean formula.

Again, it is straightforward to translate it into a circuit, but there might be a problem with depth, e.g., when B_n is:

$$a_1 \wedge (a_2 \wedge (a_3 \wedge (a_4 \wedge \ldots (a_{n-1} \wedge a_n) \ldots)))$$

The depth is $O(n)$.