Definition: Alternating time and space

Game Semantics: State of machine determines who controls, White wants it to accept, Black wants it to reject. $\mathcal{L}(A) = \{w : \text{White wins the } M\text{-game on input } w\}$.

Examples:

1. MCVP $\in$ **ASPACE**$[\log n]$
2. QSAT $\in$ **ATIME**$[n]$

Theorem: For $s(n) \geq \log n$,

$$\textbf{NSPACE}[s(n)] \subseteq \textbf{ATIME}[(s(n))^2] \subseteq \textbf{DSPACE}[(s(n))^2]$$

$$\textbf{ASPACE}[s(n)] = \textbf{DTIME}[2^{O(s(n))}]$$

Corollary:

$$\textbf{ASPACE}[\log n] = \textbf{P}$$
$$\textbf{ATIME}[n^{O(1)}] = \textbf{PSPACE}$$
$$\textbf{ASPACE}[n^{O(1)}] = \textbf{EXPTIME}$$

The Turing machine and the abstract RAM are *sequential* machines, in that they perform only one operation at a time.

Real computers are largely sequential as well, but:

- Modern computer networks allow us to apply many processors to the same problem (e.g, `SETI@home`),
- Modern programming languages allow for parallel execution threads,
- Modern processors are slightly parallel, with the capacity to do a few things at the same time,
- There have been some experimental *massively parallel* computers such as the Connection Machine, and
- The circuit elements *inside a given chip* operate in parallel.

Can we solve *any* problem a million times faster by applying a million parallel processors to it? Probably not, but as with the P vs. NP question we don't have any theorems confirming our intuition.

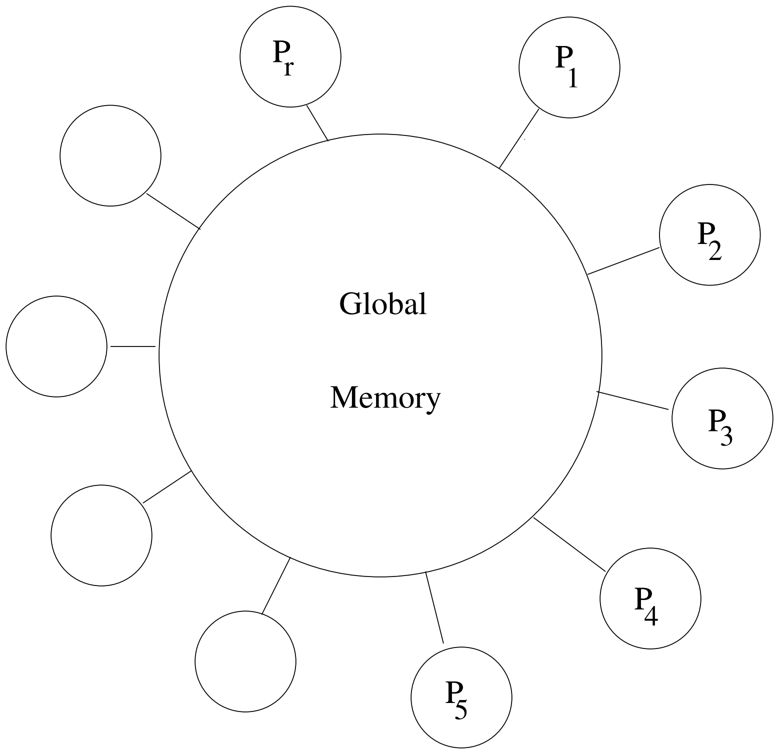
Parallel complexity theory studies the resources needed to solve problems in parallel. To begin such a study we need a formal model of parallel computation, analogous to the Turing machine or RAM.

As it turns out, just as the TM and RAM have similar behavior with respect to time and space, various different parallel models have similar behavior with respect to *parallel time* and *amount of hardware*.

Parallel Random Access Machines

$\mathbf{CRAM}[t(n)]\ = \text{CRCW-PRAM-TIME}[t(n)]\text{-HARD}[n^{O(1)}]$

synchronous, concurrent read and write, uniform,
$n^{O(1)}$ processors and memory



priority write: lowest number processor wins conflict
common write: no conflicts allowed

The alternating Turing machine is another parallel model of sorts, since the "acceptance behavior" depends on the entire set of configurations.

The parallel time measure turns out to be the number of *alternations* between existential and universal states:

Theorem 22.1 *For* $\log n \leq t(n) \leq n^{O(1)}$, $\mathbf{CRAM}[t(n)]$ *is equal to the class of languages of ATM's with space* $O(\log n)$ *and* $O(t(n))$ *alternations.*

We won't prove this here (I might put it on HW#8), but we'll show that ATM's are closely related to another parallel computing model, that of *boolean circuits*.

An important special case of ATM computation is when the number of alternations is bound by a constant. We use the same names for constant-alternation classes that we defined for the Arithmetic Hierarchy in HW#5. For example,

Σ_1**P** consists of the languages of poly-time ATM's that always stay in existential states, that is, **NP**.

Π_1**P** is the same for only universal states, that is, co-**NP**.

Σ_2**P** consists of the languages of poly-time ATM's that have a phase of existential configurations followed by a phase of universals. Π_2**P** is the complement of Σ_2**P**, and so on.

PH is defined to be the union of Σ_i**P** and Π_i**P** for all constant i, or languages of poly-time ATM's with $O(1)$ alternations.

Theorem 22.2 PH $=$ SO.

The proof is a simple generalization of Fagin's Theorem, **NP** $=$ SO$\exists$.

The Logtime Hierarchy

On HW #7 we're looking at ATM's that operate in $O(\log n)$ time, making key use of their random-access input tape. You're asked to prove that such an ATM can decide:

- any language in FO, and also
- the PARITY language, of strings with an odd number of 1's

PARITY is not in FO, though we won't be able to prove such a *lower bound* in this course.

The complexity class LH, the *log-time hierarchy*, is the set of languages decidable in **ATIME**$(\log n)$ with $O(1)$ alternations. Your HW solution will most likely show that FO $\subseteq$ LH. It turns out that with the right definition of FO, FO $=$ LH.

Real computers are built from many copies of small and simple components.

Circuit complexity uses circuits of boolean logic gates as its model of computation.

Circuits are directed acyclic graphs. Inputs are placed at the leaves. Signals proceed up toward the root, r.

Straight-line code: gates are not reused.

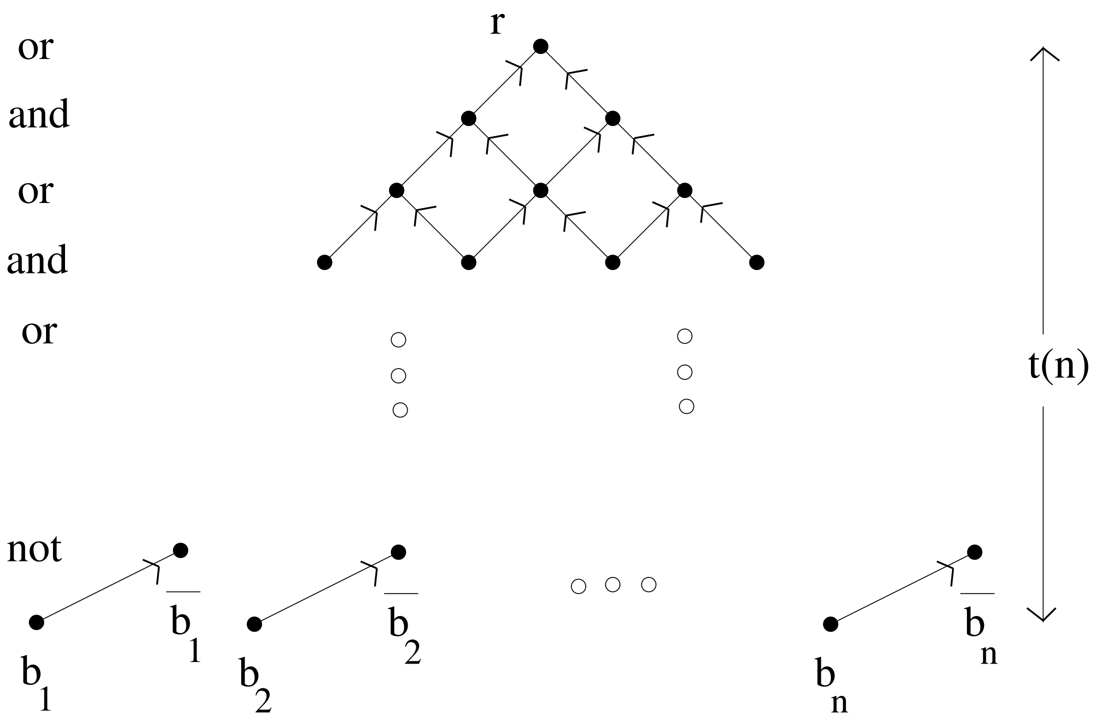
Let $S \subseteq \{0,1\}^\star$ be a decision problem.

Let, $C_1, C_2, C_3, \ldots$ be a circuit family.

C_n has n input bits and one output bit r.

Def: $\{C_i\}_{i \in \mathbf{N}}$ *computes* S iff for all n and for all $w \in \{0,1\}^n$,

$$w \in S \qquad \Leftrightarrow \qquad C_{|w|}(w) = 1 \ .$$



"not" gates are pushed down to bottom

Depth $=$ parallel time

Number of gates $=$ computational work $=$ sequential time

Consider the class **PSIZE** of languages A that are computed by a family of poly-size circuits. That is, for each n, there is a circuit C_n that accepts an input string w iff $w \in A$.

It is easy to see from our construction for Fagin's Theorem that $\mathbf{P} \subseteq \mathbf{PSIZE}$. Also since CVP is in **P**, it seems that **PSIZE** should be no more powerful than **P**.

But as we've defined **PSIZE**, it contains *undecidable* languages! Look at $UK = \{w : |w| \in K\}$ for example. For any input length n, there is a one-gate circuit that decides whether the input is in UK: it either says yes or says no without looking at the input at all. So UK is in *PSIZE*, but it's clearly r.e.-complete as it's just a recoding of K.

But this circuit is *non-uniform*. Given a number n, it is impossible for *any* Turing machine, much less a poly-time TM, to determine what C_n is.

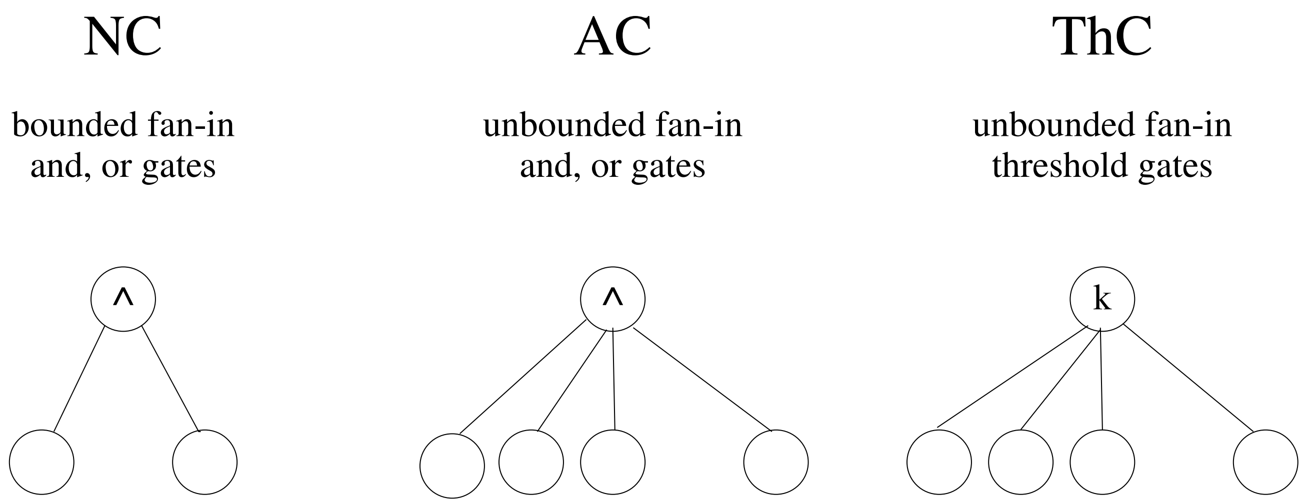
Let's define **P**-*uniform* **PSIZE** to be those languages decided by poly-time circuit families where we can compute C_n from the string 1^n in $n^{O(1)}$ time. Now it's easy to see that **P**-uniform **PSIZE** is contained in **P**, because our machine on input w can first build the circuit $C_{|w|}$ and then solve the CVP problem that tells what $C_{|w|}$ does on input w.

And in fact the tableau construction tells us that **P** is contained in **P**-uniform **PSIZE**, because the circuit from the tableau is easy to construct. In fact it's *very* easy to construct – we could do it in $F(\mathbf{L})$ or even in $F(\mathrm{FO})$. Thus we have:

Theorem: **P** = **P**-uniform **PSIZE** = **L**-uniform **PSIZE** = FO-uniform **PSIZE**.

Definition 22.3 (The NC Hierarchy) Let $t(n)$ be a polynomially bounded function and let $S \subseteq \{0,1\}^\star$ Then S is in the circuit complexity class $\mathbf{NC}[t(n)]$, $\mathbf{AC}[t(n)]$, $\mathbf{ThC}[t(n)]$, respectively iff there exists a uniform family of circuits $C_1, C_2, \ldots$ with the following properties:

1. For all $w \in \{0,1\}^\star, \quad w \in S \quad \Leftrightarrow \quad C_{|w|}(w) = 1$
2. The depth of C_n is $O(t(n))$.
3. $|C_n| \leq n^{O(1)}$
4. The gates of C_n consist of,



For $i = 0, 1, \ldots$,

$$\begin{aligned} \mathbf{NC}^i &= \mathbf{NC}[(\log n)^i] \\ \mathbf{AC}^i &= \mathbf{AC}[(\log n)^i] \\ \mathbf{ThC}^i &= \mathbf{ThC}[(\log n)^i] \end{aligned}$$

$$\mathbf{NC} \quad = \quad \bigcup_{i=0}^{\infty} \mathbf{NC}^i \quad = \quad \bigcup_{i=0}^{\infty} \mathbf{AC}^i \quad = \quad \bigcup_{i=0}^{\infty} \mathbf{ThC}^i$$

We will see that the following inclusions hold:

$$\begin{array}{ccccccc} \mathbf{AC}^0 & \subseteq & \mathbf{ThC}^0 & \subseteq & \mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} & \subseteq & \mathbf{AC}^1 \\ \mathbf{AC}^1 & \subseteq & \mathbf{ThC}^1 & \subseteq & \mathbf{NC}^2 & \subseteq & \mathbf{AC}^2 \\ \mathbf{AC}^2 & \subseteq & \mathbf{ThC}^2 & \subseteq & \mathbf{NC}^3 & \subseteq & \mathbf{AC}^3 \\ \vdots & \subseteq & \vdots & \subseteq & \vdots & \subseteq & \vdots \\ \bigcup_{i=1}^{\infty} \mathbf{AC}^i & = & \bigcup_{i=1}^{\infty} \mathbf{ThC}^i & = & \bigcup_{i=1}^{\infty} \mathbf{NC}^i & = & \mathbf{NC} \end{array}$$

The word *uniform* above means that the map, $f : 1^n \mapsto C_n$ is *very easy* to compute, for example, $f \in F(\mathbf{L})$ or $f \in F(\mathrm{FO})$. Though these *uniformity conditions* are a subject dear to my heart, we won't worry too much about the details of them in this course.

Overall, **NC** consists of those problems that can be solved in *poly-log parallel time* on a parallel computer with *polynomially much hardware*. The question of whether $\mathbf{P} = \mathbf{NC}$ is the *second* most important open question in complexity theory, after the $\mathbf{P} = \mathbf{NP}$ question.

You wouldn't think that *every* problem in **P** can be sped up to polylog time by parallel processing. Some problems appear to be *inherently sequential*. If we prove that a problem is **P**-complete, we know that it is *not* in **NC** unless $\mathbf{P} = \mathbf{NC}$.

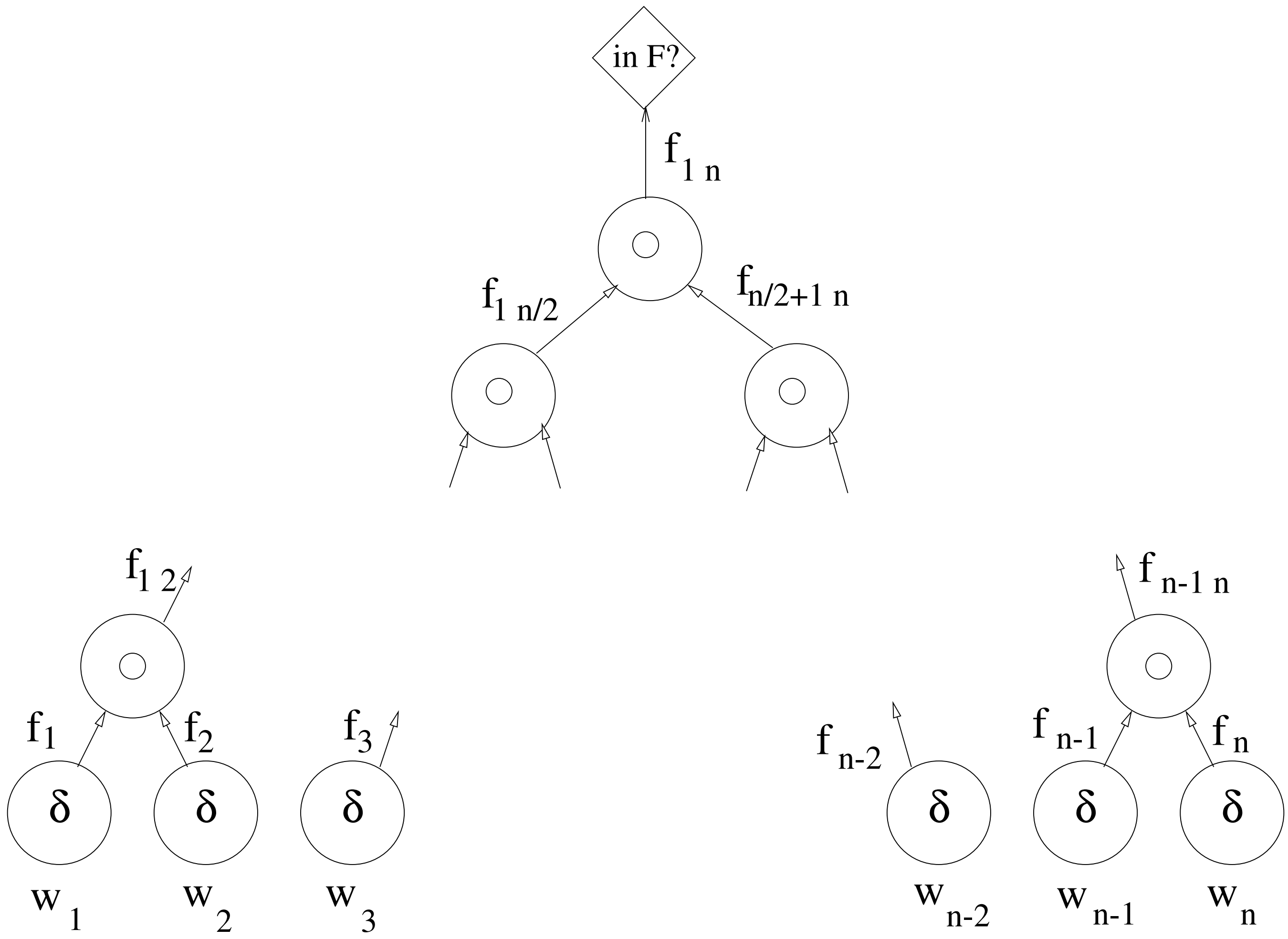
Theorem: CVP, MCVP and HORN-SAT are all **P**-complete.

Proposition 22.4 *Every regular language is in* $\mathbf{NC}^1$.

Proof: Given DFA $D = \langle \Sigma, Q, \delta, s, F \rangle$.

Construct circuits $C_1, C_2, \ldots$, for all $w \in \Sigma^\star$,

$$w \in \mathcal{L}(D) \qquad \Leftrightarrow \qquad C_{|w|}(w) = 1$$



$$f_i(q) = \delta(q, w_i); \qquad w \in \mathcal{L}(D) \qquad \Leftrightarrow \qquad f_{1n}(s) \in F \spadesuit$$

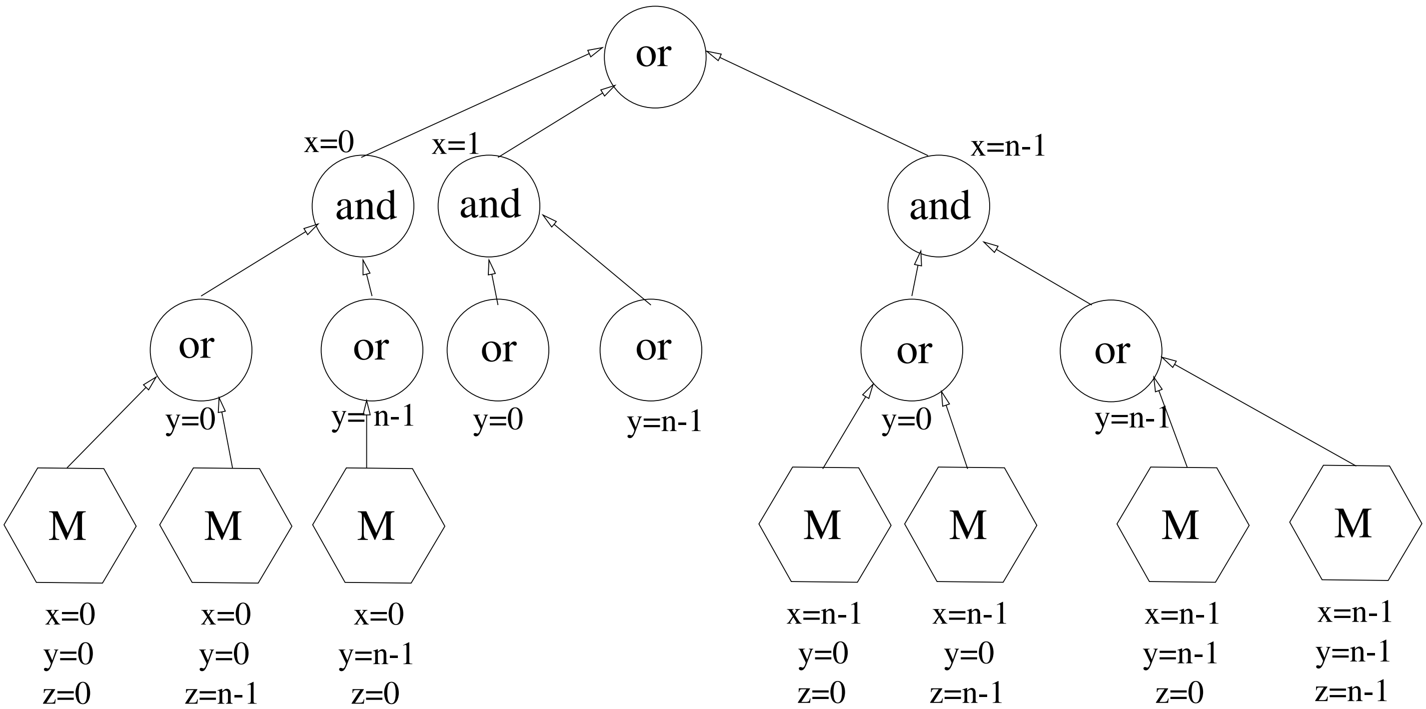
By a very similar argument, you can show that every regular language is in $\mathbf{ATIME}(\log n)$. Actually, with a suitable uniformity condition,

Theorem: (Ruzzo) $\mathbf{NC}^1 = \mathbf{ATIME}(\log n)$.

Theorem 22.5 *(Barrington-Immerman-Straubing)* FO $=$ **AC**^{0}

Proof:(Sketch of one direction, with some uniformity details skipped)

$$\varphi \quad = \quad (\exists x)(\forall y)(\exists z)M(x,y,z)$$



♠

Proposition 22.6 *For* $i = 0, 1, \ldots,$

$$\mathbf{NC}^i \quad \subseteq \quad \mathbf{AC}^i \quad \subseteq \quad \mathbf{ThC}^i \quad \subseteq \quad \mathbf{NC}^{i+1}$$

Proof:

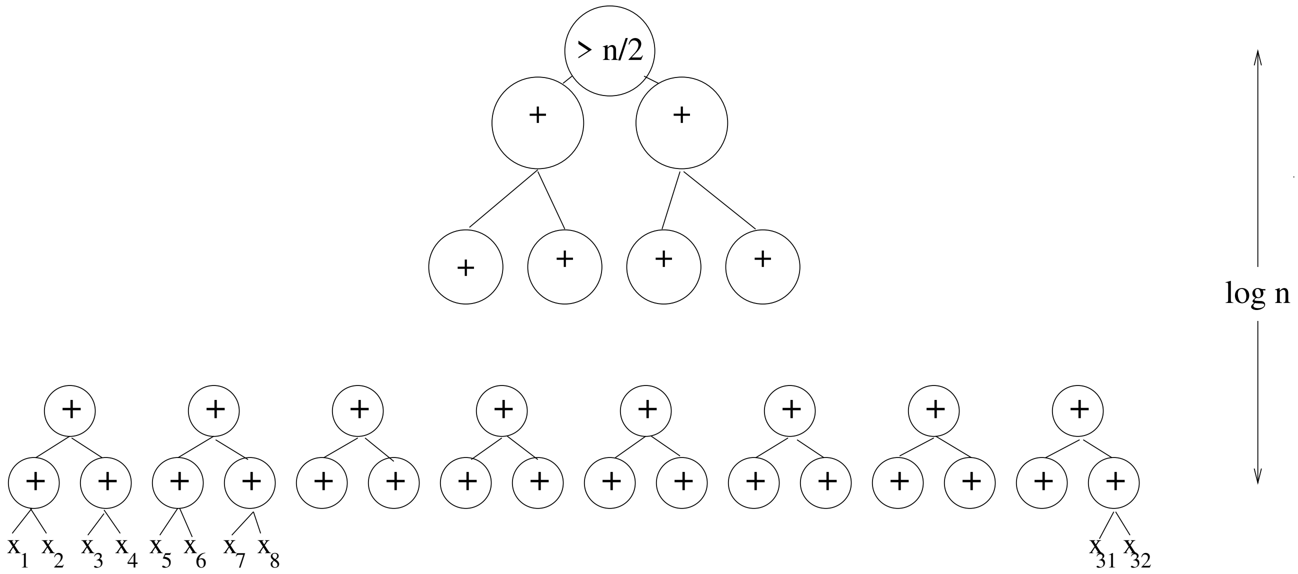
All inclusions but $\mathbf{ThC}^i \subseteq \mathbf{NC}^{i+1}$ are clear.

$\mathrm{MAJ} \;=\; \{w \in \{0,1\}^\star \mid w$ has more than $|w|/2$ "1"s$\} \;\in \mathbf{ThC}^0$

Lemma 22.7 $\quad \mathrm{MAJ} \in \mathbf{NC}^1$

(and the same for any other threshold gate).

The obvious way to try to build an $\mathbf{NC}^1$ circuit for majority is to add the n input bits via a full binary tree of height $\log n$. The problem with this, is that while the sums being added have more and more bits, we must still add them in constant depth.



A solution to this problem is via ambiguous arithmetic notation. Consider a representation of natural numbers in binary, except that digits $0, 1, 2, 3$ may be used. For example 3213 and 3221 are different representations of the decimal number 37 in this ambiguous notation,

$$3213 \;=\; 3 \cdot 2^3 + 2 \cdot 2^2 + 1 \cdot 2^1 + 3 \cdot 2^0 \;=\; 37$$

$$3221 \;=\; 3 \cdot 2^3 + 2 \cdot 2^2 + 2 \cdot 2^1 + 1 \cdot 2^0 \;=\; 37$$

Lemma 22.8 *Adding two n bit numbers in ambiguous notation can be done via an* $\mathbf{NC}^0$ *circuit, i.e., with bounded depth and bounded fan-in.*

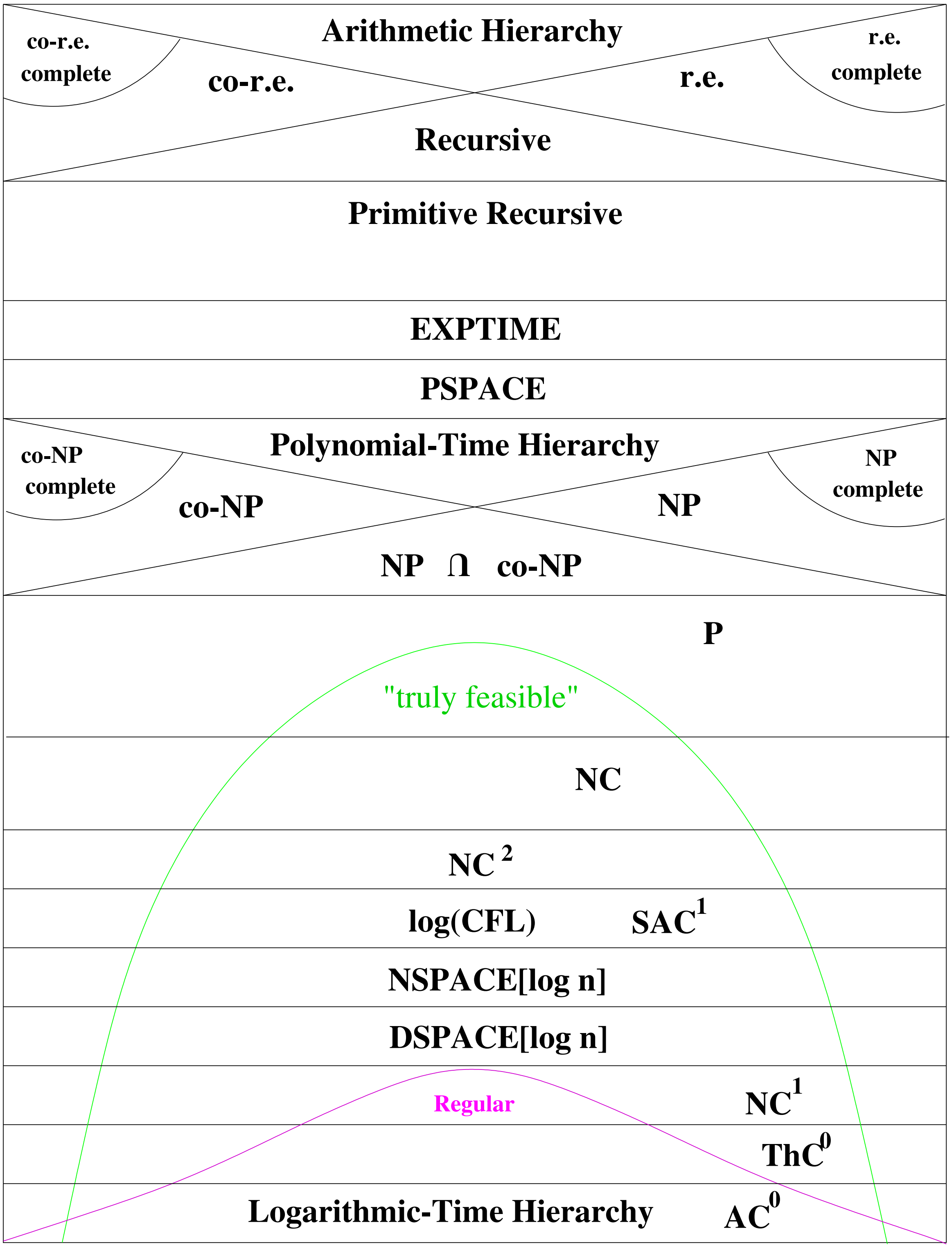
Example:

$$
\begin{array}{rccccc}
\text{carries:} & 3 & 2 & 2 & 3 & \\
 & & & & & \\
 & & 3 & 2 & 1 & 3 \\
+ & & 3 & 2 & 1 & 3 \\
\hline
 & 3 & 2 & 2 & 1 & 0
\end{array}
$$

This is doable in $\mathbf{NC}^0$ because the carry from column i can be computed by looking only at columns i and $i+1$.

Translating from ambiguous notation back to binary, which must be done only once at the end, is just an addition problem. This is first-order, and thus $\mathbf{AC}^0$, and thus $\mathbf{NC}^1$.

♠


Arithmetic Hierarchy
co-r.e. complete
co-r.e.
r.e.
r.e. complete
Recursive
Primitive Recursive
EXPTIME
PSPACE
Polynomial-Time Hierarchy
co-NP complete
co-NP
NP
NP complete
NP ∩ co-NP
P
"truly feasible"
NC
NC 2
log(CFL)
SAC1
NSPACE[log n]
DSPACE[log n]
Regular
NC1
ThC0
Logarithmic-Time Hierarchy
AC0