Recall From Last Time

Definition: Alternating time and space

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

Examples:

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

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

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

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

Corollary:

$$\begin{aligned} \mathbf{ASPACE}[\log n] &= \mathbf{P} \\ \mathbf{ATIME}[n^{O(1)}] &= \mathbf{PSPACE} \\ \mathbf{ASPACE}[n^{O(1)}] &= \mathbf{EXPTIME} \end{aligned}$$

Parallel Computation

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.

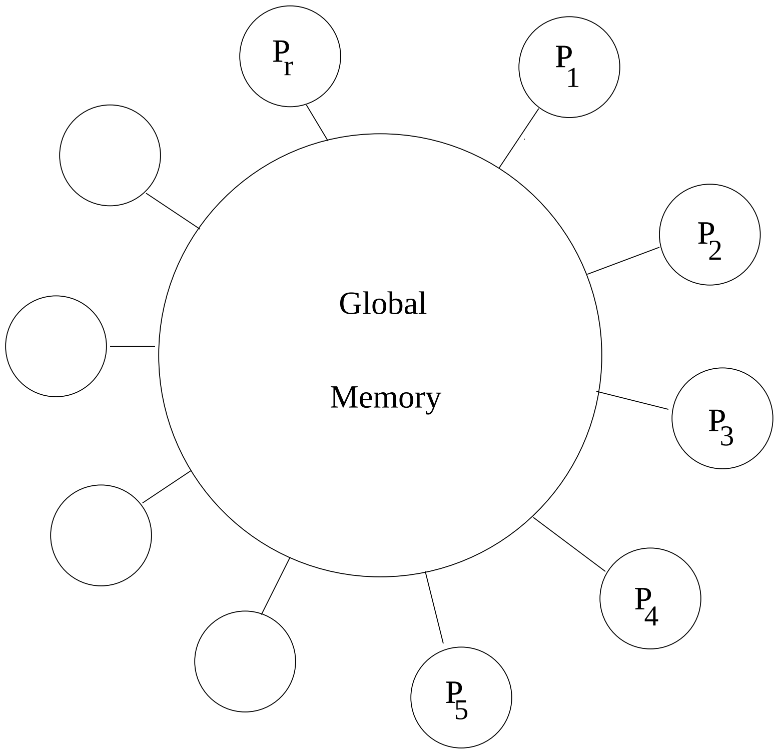
Parallel complexity theory studies the resources needed to solve problems in parallel. To begin such a study we need a formal model of parallel computation, analogous to the Turing machine or RAM.

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

CRAM$[t(n)]$ = CRCW-PRAM-TIME$[t(n)]$-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: write conflict crashes machine

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 23.1 *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 a piece of the proof on HW#8), but we’ll show that ATM’s are closely related to another parallel computing model, that of *boolean circuits*. First, however, a digression:

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 23.2 PH $=$ SO.

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

The Logtime Hierarchy

On Spring 2003's HW #7 we looked at ATM's that operate in $O(\log n)$ time, making key use of their random-access input tape. We proved then 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. A careful solution shows that FO $\subseteq$ LH. It turns out that with the right definition of FO, FO $=$ LH.

On this term's HW#7 we look at the similar *logspace hierarchy* LSH, which you'll prove *collapses to* (is equal to) **NL**.

Circuit Complexity

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.

The code is *straight-line* in that gates are not reused (because the graph is acyclic. In fact the *boolean straight-line programs* of Lecture 1 and HW#1 were simply a reformulation of boolean circuits.

Circuit Families and Languages:

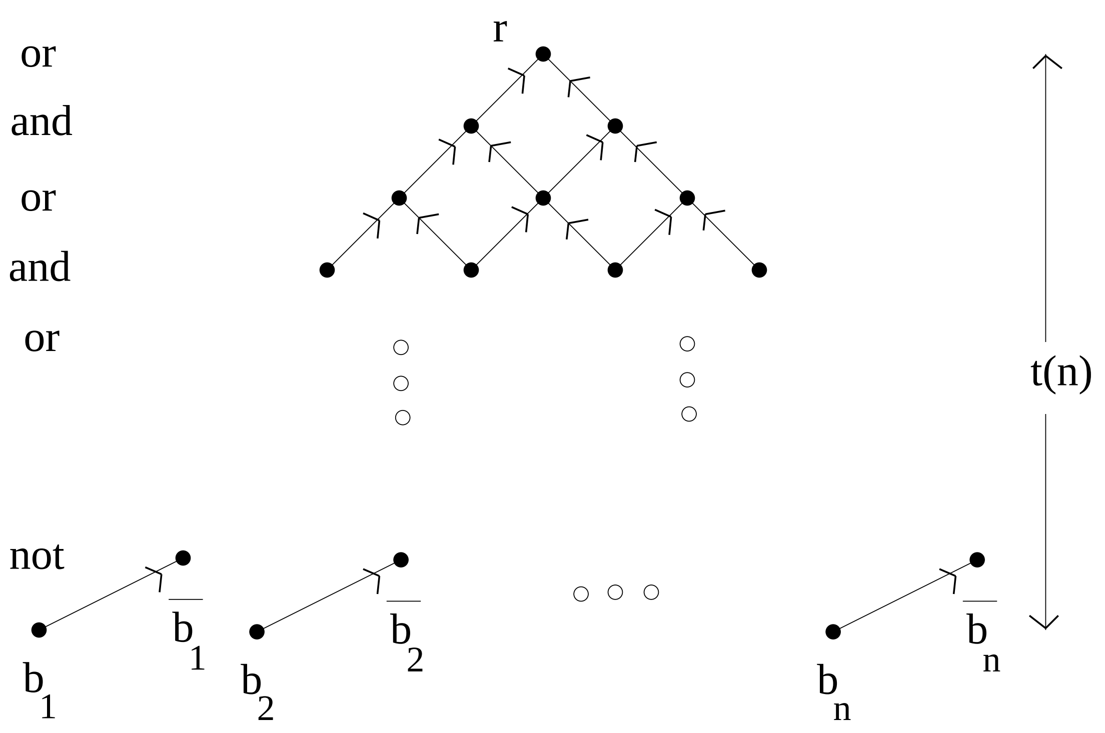
Let $S \subseteq \{0,1\}^\star$ be a language (or decision problem).

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

Each circuit C_n has n input bits and one output bit r.

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

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



It turns out that NOT gates in the middle of the circuit can be pushed down to the bottom and eliminated without changing the parameters of the circuit, so we will keep circuits in this form.

Size and Depth:

The **depth** of the circuit is the length of the longest path from an input to an output. (Compare the depth of an SLP in HW#1.) The depth measures the **parallel time**, the total delay for the circuit to compute its value, in terms of the individual gate delays.

The **size** of the circuit is the number of gates, counting the input gates and their negations. This is the length of the corresponding SLP. It represents the **computational work** used to solve the problem, and corresponds roughly to the **sequential time** needed to evaluate the circuit.

These size and depth parameters become **functions of the input size** n once we consider a circuit family instead of a single circuit. These become our cost measures and we use them to define complexity classes.

Circuit Uniformity

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 **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.

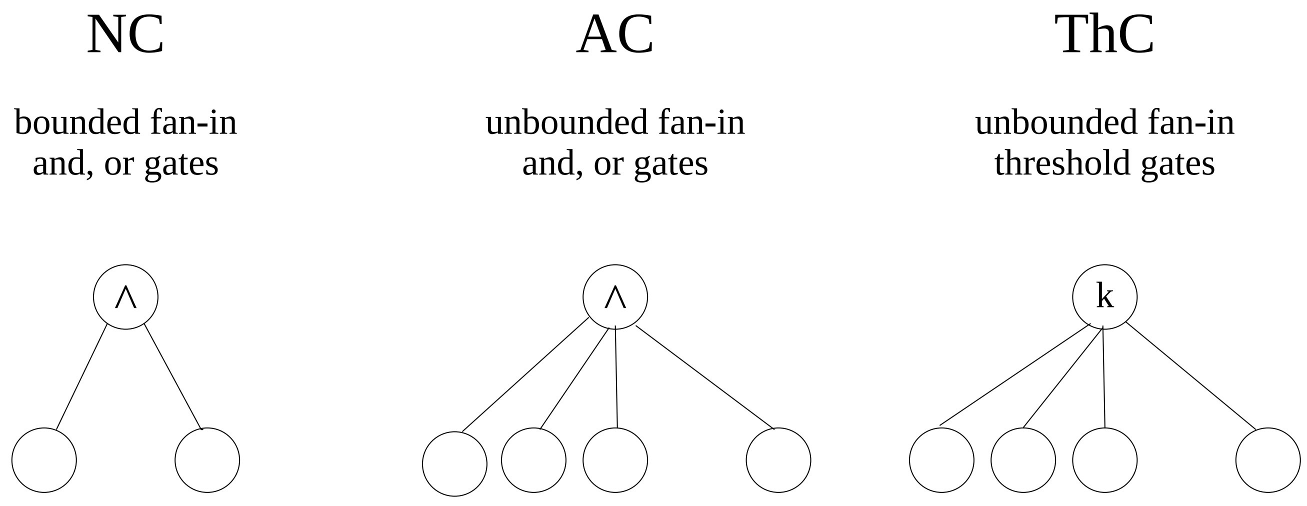
Let's define **P**-*uniform* **PSIZE** to be those languages decided by poly-time circuit families where we can compute C_n from the string 1^n in $n^{O(1)}$ time. Now it's easy to see that **P**-uniform **PSIZE** is contained in **P**, because our machine on input w can first build the circuit $C_{|w|}$ 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 $F(\text{FO})$. Thus we have:

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

Definition 23.3 (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 **NC**$[t(n)]$, **AC**$[t(n)]$, **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:


NC
bounded fan-in and, or gates
∧
AC
unbounded fan-in and, or gates
∧
ThC
unbounded fan-in threshold gates
k

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 **P** $=$ **NC** is the *second* most important open question in complexity theory, after the **P** $=$ **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 **P** $=$ **NC**.

Theorem 23.4 *CVP, MCVP (monotone CVP) and HORN-SAT are all* **P**-*complete.*

Proof: CVP is the set of pairs (C, x) such that circuit C accepts input string x (which must thus be of the right length). This is pretty clearly in **P** because we can evaluate C gate by gate.

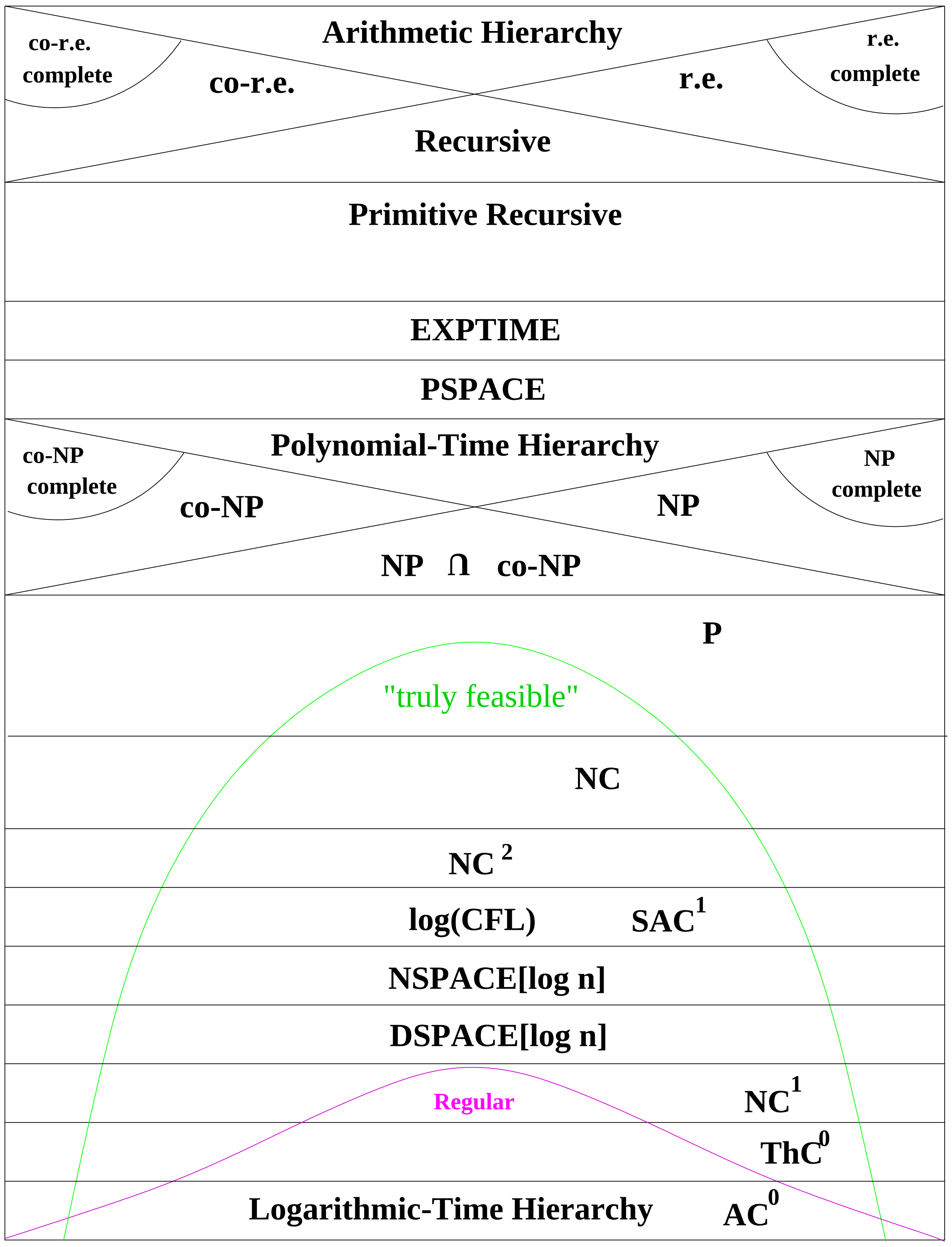
To reduce an arbitrary **P** language to CVP, we use the tableau construction from the Fagin or Cook-Levin proofs (or from the proof that **P** is contained in alternating logspace). Since each cell of the tableau depends on those just below it, and any function of $O(1)$ boolean inputs has circuits of $O(1)$ size and depth (HW#1), we can build a circuit that computes each cell value. The output gate of this circuit tells us whether the machine accepts the input by looking at the first cell of the tape at the last time step.

The monotone problem MCVP is clearly still in **P**. To reduce CVP to MCVP we can use *double-rail logic*. For each gate g of the orginal circuit C we make two gates, one to compute g and the other to compute $\neg g$. If we have done this inductively for g's children, it's easy to do it for g with AND and OR gates (there are four cases).

HORN-SAT is the language of satisfiable boolean formulas in HORN-CNF – AND's of clauses, each of which is of the form $(x_1 \wedge \ldots \wedge x_k) \rightarrow y$ where the x_i's and y are variables (not negated). We solve it in **P** with a greedy algorithm, starting with all variables false and setting true only those required by the clauses, in successive passes.

To reduce MCVP to HORN-SAT, we make a variable for each gate and one or two clauses for the effect of each gate. If g is an AND gate with children h_1 and h_2, we add the Horn clause $(h_1 \wedge h_2) \rightarrow g$. If g is the OR of h_1 and h_2, we add $h_1 \rightarrow g$ and $h_2 \rightarrow g$. We then force the inputs to the correct values and force the output gate's variable to 1, and we are satisfiable iff the circuit actually computes 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