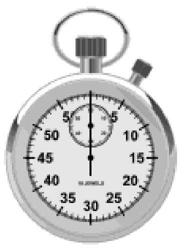
Resource-Bounded Computation

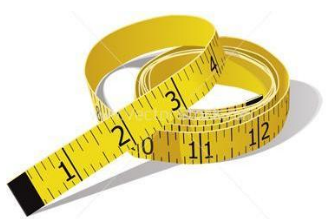
Previously: can something be done?

Now: how efficiently can it be done?



Goal: conserve computational resources:

Time, space, other resources?



Def: L is decidable within time O(t(n)) if some TM M that decides L always halts on all w∈Σ* within O(t(|w|)) steps / time.

Def: L is decidable within space O(s(n)) if some TM M that decides L always halts on all w∈Σ* while never using more than O(s(|w|)) space / tape cells.

Complexity Classes

Def: DTIME(t(n))={L | L is decidable within time O(t(n)) by some deterministic TM}

Def: NTIME(t(n))={L | L is decidable within time O(t(n)) by some non-deterministic TM}

Def: DSPACE(s(n))={L | L decidable within space O(s(n)) by some deterministic TM}

Def: NSPACE(s(n))={L | L decidable within space O(s(n)) by some non-deterministic TM}

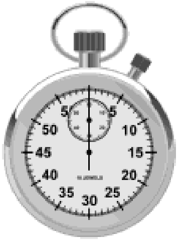
Properties!

Time is Tape Dependent

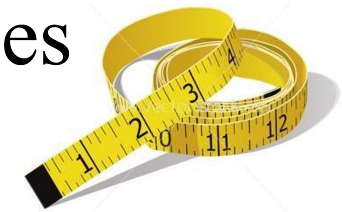
Theorem: The time depends on the # of TM tapes.

Idea: more tapes can enable higher efficiency.

Ex: $\{0^n1^n \mid n>0\}$ is in DTIME(n^2) for 1-tape TM's, and is in DTIME(n) for 2-tape TM's.



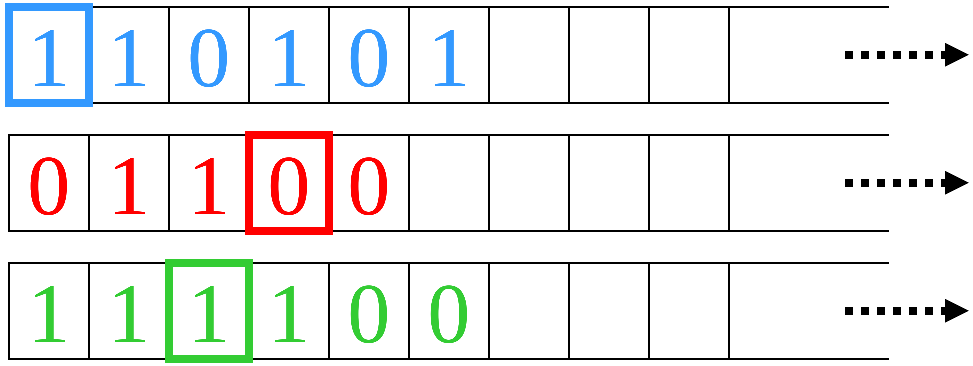
Note: For multi-tape TM's, input tape space does not "count" in the total space s(n). This enables analyzing sub-linear space complexities.



Space is Tape Independent

Theorem: The space does not depend on the # tapes.

Proof:



Idea: Tapes can be "interlaced" space-efficiently:

Note: This does not asymptotically increase the overall space (but can increase the total time).

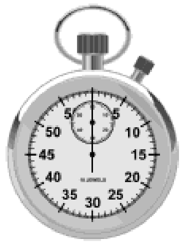
Theorem: A 1-tape TM can simulate a t(n)-time-bounded k-tape TM in time $O(k \cdot t^2(n))$.

Space-Time Relations

Theorem: If $t(n) < t'(n)\ \forall n>1$ then:

$$DTIME(t(n)) \subseteq DTIME(t'(n))$$

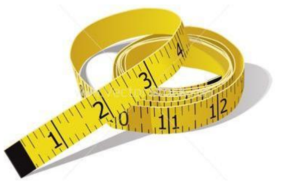
$$NTIME(t(n)) \subseteq NTIME(t'(n))$$



Theorem: If $s(n) < s'(n)\ \forall n>1$ then:

$$DSPACE(s(n)) \subseteq DSPACE(s'(n))$$

$$NSPACE(s(n)) \subseteq NSPACE(s'(n))$$

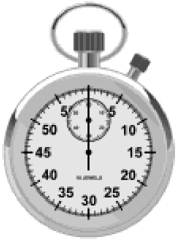


Example: $NTIME(n) \subseteq NTIME(n^2)$

Example : $DSPACE(\log n) \subseteq DSPACE(n)$

Examples of Space & Time Usage

Let $L_1 = \{0^n 1^n \mid n > 0\}$:

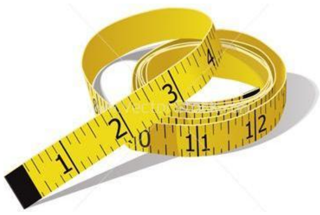


For 1-tape TM's:

$L_1 \in \text{DTIME}(n^2)$

$L_1 \in \text{DSPACE}(n)$

$L_1 \in \text{DTIME}(n \log n)$



For 2-tape TM's:

$L_1 \in \text{DTIME}(n)$

$L_1 \in \text{DSPACE}(\log n)$

Examples of Space & Time Usage

Let $L_2 = \Sigma^*$ 

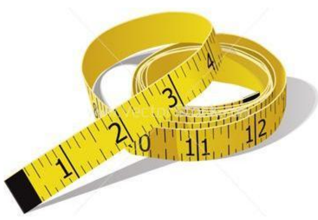
$L_2 \in DTIME(n)$

Theorem: every regular language is in DTIME(n)

$L_2 \in DSPACE(1)$

Theorem: every regular language is in DSPACE(1)

$L_2 \in DTIME(1)$



Let $L_3 = \{w\$w \mid w \text{ in } \Sigma^*\}$

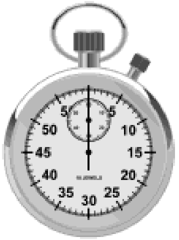
$L_3 \in DTIME(n^2)$

$L_3 \in DSPACE(n)$

$L_3 \in DSPACE(\log n)$

Special Time Classes

Def: $P = \bigcup_{\forall k>1} \text{DTIME}(n^k)$



P ≡ deterministic polynomial time

Note: P is robust / model-independent

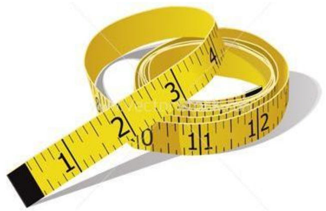
Def: $NP = \bigcup_{\forall k>1} \text{NTIME}(n^k)$

NP ≡ non-deterministic polynomial time

Theorem: $P \subseteq NP$

Conjecture: $P = NP$?   Million $ question!

Other Special Space Classes



Def: $PSPACE = \bigcup_{\forall k>1} DSPACE(n^k)$

PSPACE ≡ deterministic polynomial space

Def: $NPSPACE = \bigcup_{\forall k>1} NSPACE(n^k)$

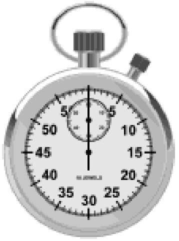
NPSPACE ≡ non-deterministic polynomial space

Theorem: PSPACE ⊆ NPSPACE (obvious)

Theorem: PSPACE = NPSPACE (not obvious)

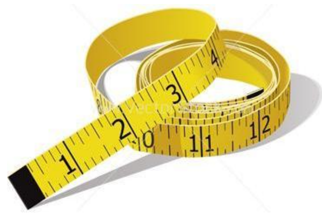
Other Special Space Classes

Def: $\text{EXPTIME} = \bigcup_{\forall k>1} \text{DTIME}(2^{n^k})$



$\text{EXPTIME} \equiv$ exponential time

Def: $\text{EXPSPACE} = \bigcup_{\forall k>1} \text{DSPACE}(2^{n^k})$

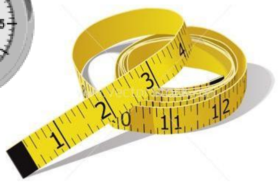


$\text{EXPSPACE} \equiv$ exponential space

Def: $\text{L} = \text{LOGSPACE} = \text{DSPACE}(\log n)$

Def: $\text{NL} = \text{NLOGSPACE} = \text{NSPACE}(\log n)$

Space/Time Relationships



Theorem: DTIME(f(n)) $\subseteq$ DSPACE(f(n))

Theorem: DTIME(f(n)) $\subseteq$ DSPACE(f(n) / log(f(n)))

Theorem: NTIME(f(n)) $\subseteq$ DTIME($c^{f(n)}$)
for some c depending on the language.

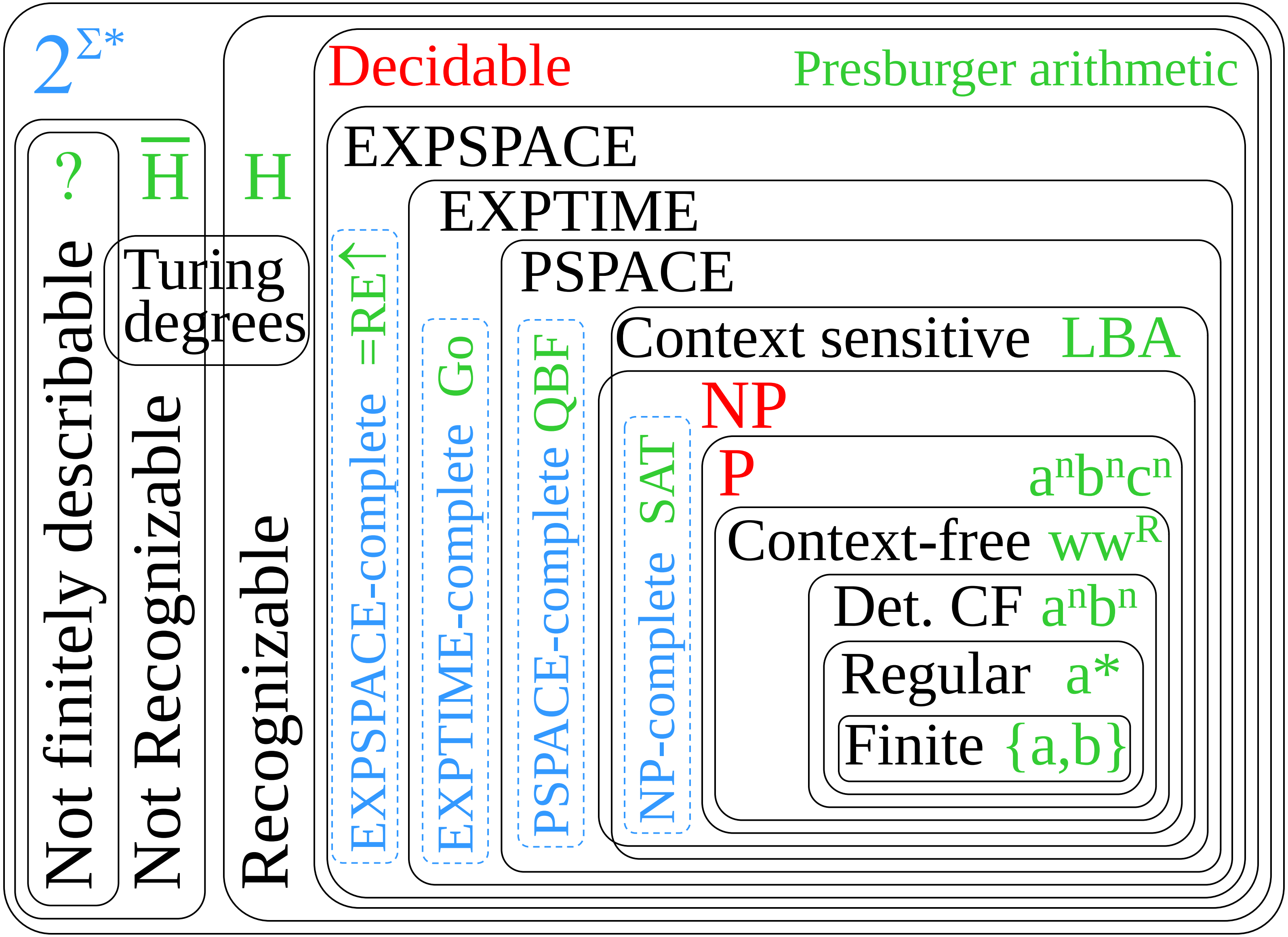
Theorem: DSPACE(f(n)) $\subseteq$ DTIME($c^{f(n)}$)
for some c, depending on the language.

Theorem [Savitch]: NSPACE(f(n)) $\subseteq$ DSPACE(f^2(n))

Corollary: PSPACE = NPSPACE

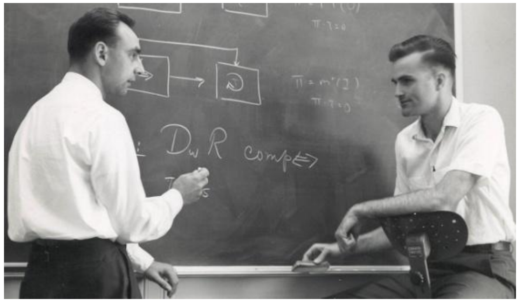
Theorem: NSPACE(n^r) $\subseteq$ DSPACE($n^{r+\varepsilon}$) $\forall$ r>0, ε>0

The Extended Chomsky Hierarchy



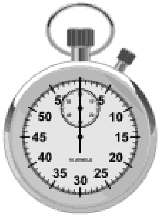

RESEARCH INSTITUTE
UNANSWERED QUESTIONS
UNQUESTIONED ANSWERS

Time Complexity Hierarchy



Juris Hartmanis   Richard Stearns

Theorem: for any $t(n)>0$ there exists a decidable language $L \notin \text{DTIME}(t(n))$.



$\Rightarrow$ No time complexity class contains <u>all</u> the decidable languages, and the time hierarchy is ∞!

$\Rightarrow$ There are decidable languages that take arbitrarily long times to decide!

Note: $t(n)$ must be computable & everywhere defined

Proof: (by diagonalization)

Fix lexicographic orders for TM's: M_1, M_2, M_3, . . .

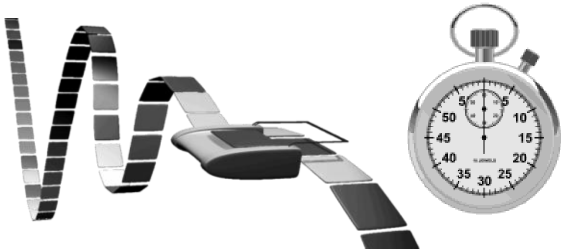
Interpret TM inputs $i \in \Sigma^*$ as encodings of integers:

a=1, b=2, aa=3, ab=4, ba=5, bb=6, aaa=7, …

Time Complexity Hierarchy (proof)

Define L={i | M_i does not accept i within t(i) time}

Note: L is decidable (by simulation)



Q: is L∈DTIME(t(n)) ?

Assume (towards contradiction) L∈DTIME(t(n))

i.e., ∃ a fixed K∈N such that Turing machine M_K decides L within time bound t(n)

i → If M_i accepts i within t(i) time then → Reject

else → Accept

$M_K \Rightarrow$ decides / accepts L

Time Complexity Hierarchy (proof)

K → If M_K accepts K within t(K) time then → Reject
else → Accept

$M_K \Rightarrow$ decides / accepts L

Consider whether $K \in L$:

$K \in L \Rightarrow M_K$ must accept K within t(K) time

$\Rightarrow M_K$ must reject $K \Rightarrow K \notin L$

$K \notin L \Rightarrow M_K$ must reject K within t(K) time

$\Rightarrow M_K$ must accept $K \Rightarrow K \in L$

So $(K \in L) \Leftrightarrow (K \notin L)$, a <u>contradiction</u>!

$\Rightarrow$ Assumption is false $\Rightarrow L \notin$ DTIME(t(n)) ■

Time Hierarchy (another proof)

Consider all t(n)-time-bounded TM's on all inputs:

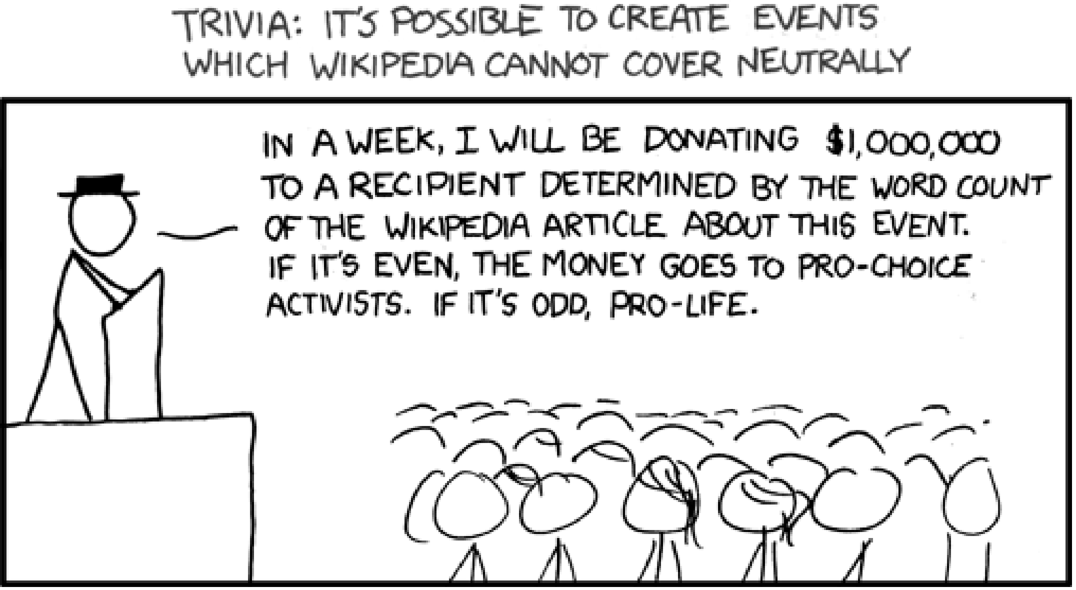
i =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
$w_i \in \Sigma^*$ =	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	aaaa	…
$M_1(i) \Rightarrow$	√	×	√	√	×	×	√	√	√	×	×	×	×	√	×	…
$M_2(i) \Rightarrow$	×	×	√	×	×	√	×	×	√	×	√	√	×	×	×	…
$M_3(i) \Rightarrow$	√	√	√	×	√	×	×	×	√	√	√	√	×	√	×	…
$M_4(i) \Rightarrow$	√	×	√	×	√	√	×	√	√	×	×	√	×	×	×	…
$M_5(i) \Rightarrow$	×	√	√	×	×	√	√	×	√	√	√	√	×	√	×	…
…																
$M'(i) \Rightarrow$	×	√	×	√	√	…										

M' … is t(n) time-bounded.

But M' computes a different function than any M_j

$\Rightarrow$ Contradiction!

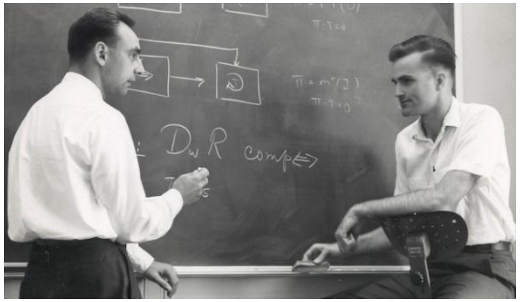
"Lexicographic order."


TRIVIA: IT'S POSSIBLE TO CREATE EVENTS WHICH WIKIPEDIA CANNOT COVER NEUTRALLY
IN A WEEK, I WILL BE DONATING $1,000,000 TO A RECIPIENT DETERMINED BY THE WORD COUNT OF THE WIKIPEDIA ARTICLE ABOUT THIS EVENT. IF IT'S EVEN, THE MONEY GOES TO PRO-CHOICE ACTIVISTS. IF IT'S ODD, PRO-LIFE.


NOT CONTENT WITH NORMAL RESTRAINING ORDERS, MY EX GOT CREATIVE.
WAIT... I CAN'T GET CLOSER THAN 500 YARDS OF YOU... OR MORE THAN 600 YARDS AWAY?
YOU'LL HAVE TO MOVE SOMEWHERE WITHIN THIS RING.

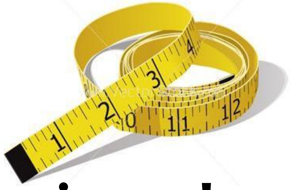

Leibniz, Boole and Gödel worked with logic.
I work with logic.
I am Leibniz, Boole and Gödel.
s.harris

Space Complexity Hierarchy



Juris Hartmanis Richard Stearns

Theorem: for any s(n)>0 there exists a decidable language L∉DSPACE(s(n)).



⇒No space complexity class contains <u>all</u> the decidable languages, and the space hierarchy is ∞!

⇒There are decidable languages that take arbitrarily much space to decide!

Note: s(n) must be computable & everywhere defined

Proof: (by diagonalization)

Fix lexicographic orders for TM's: M_1, M_2, M_3, . . .

Interpret TM inputs $i \in \Sigma^*$ as encodings of integers:

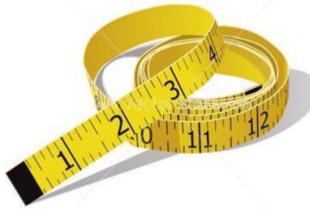
a=1, b=2, aa=3, ab=4, ba=5, bb=6, aaa=7, …

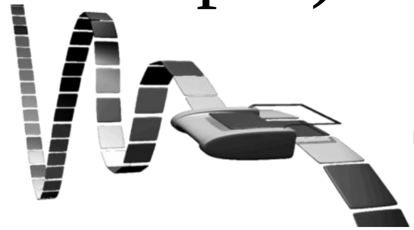
Space Complexity Hierarchy (proof)

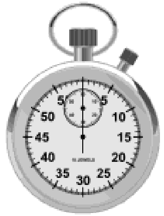
Define L={i | M_i does not accept i within t(i) space}

Note: L is decidable (by simulation; ∞-loops?)

Q: is L∈DSPACE(s(n)) ?







Assume (towards contradiction) L∈DSPACE(s(n))

i.e., ∃ a fixed K∈N such that Turing machine M_K decides L within space bound s(n)

i → If M_i accepts i within t(i) space then → Reject
else → Accept

$M_K \Rightarrow$ decides / accepts L

Space Complexity Hierarchy (proof)

$K \rightarrow$ If M_K accepts K within $s(K)$ space then $\rightarrow$ Reject

else $\rightarrow$ Accept

$M_K \Rightarrow$ decides / accepts L

Consider whether $K \in L$:

$K \in L \Rightarrow M_K$ must accept K within $s(K)$ space

$\Rightarrow M_K$ must reject $K \Rightarrow K \notin L$

$K \notin L \Rightarrow M_K$ must reject K within $s(K)$ space

$\Rightarrow M_K$ must accept $K \Rightarrow K \in L$

So $(K \in L) \Leftrightarrow (K \notin L)$, a contradiction!

$\Rightarrow$ Assumption is false $\Rightarrow L \notin \text{DSPACE}(s(n))$ ■

Space Hierarchy (another proof)

Consider all s(n)-space-bounded TM's on all inputs:

i =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
$w_i \in \Sigma^*$ =	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	bab	bba	bbb	aaaa	…
$M_1(i) \Rightarrow$	√	×	√	√	×	×	√	√	√	×	×	×	×	√	×	…
$M_2(i) \Rightarrow$	×	×	√	×	×	√	×	×	√	×	√	√	×	×	×	…
$M_3(i) \Rightarrow$	√	√	√	×	√	×	×	×	√	√	√	√	×	√	×	…
$M_4(i) \Rightarrow$	√	×	√	×	√	√	×	√	√	×	×	√	×	×	×	…
$M_5(i) \Rightarrow$	×	√	√	×	×	√	√	×	√	√	√	√	×	√	×	…
…																
$M'(i) \Rightarrow$	×	√	×	√	√	…										

… is s(n) space-bounded.

But M' computes a different function than any M_j

$\Rightarrow$ Contradiction!

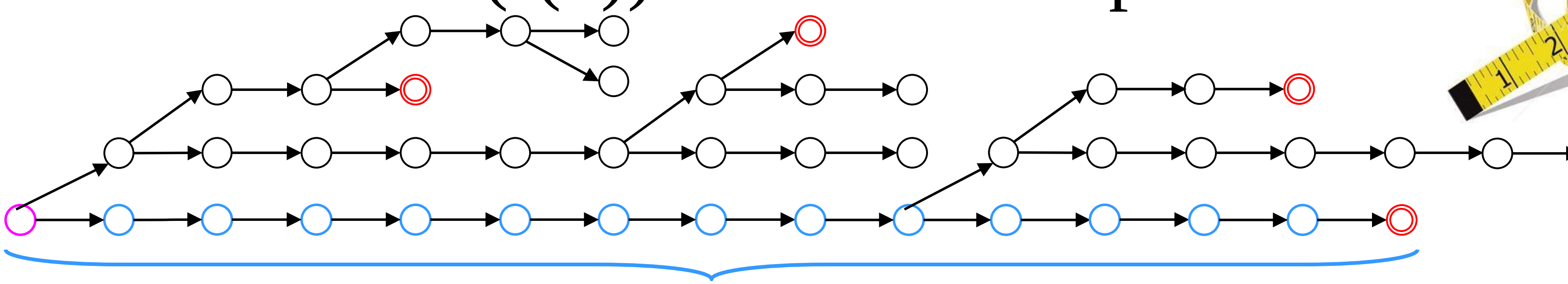
Savitch's Theorem

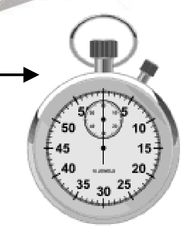

Walter Savitch

Theorem: $NSPACE(f(n)) \subseteq DSPACE(f^2(n))$

Proof: Simulation: idea is to aggressively conserve and reuse space while sacrificing (lots of) time.

Consider a sequence of TM states in one branch of an NSPACE(f(n))-bounded computation:





Computation time / length is bounded by $c^{f(n)}$ (why?)

We need to simulate this branch and all others too!

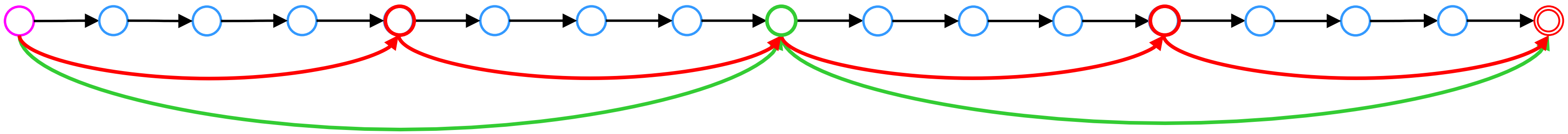
Q: How can we space-efficiently simulate these?

A: Use divide-and-conquer with heavy space-reuse!

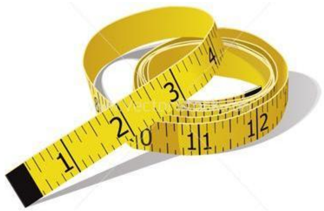
Savitch's Theorem

Walter Savitch

Pick a midpoint state along target path:



Verify it is a valid intermediate state
by recursively solving both subproblems.

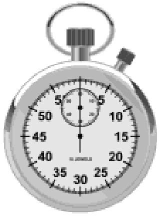


Iterate for all possible midpoint states!

The recursion stack depth is at most $\log(c^{f(n)}) = O(f(n))$

Each recursion stack frame size is $O(f(n))$.

$\Rightarrow$ total space needed is $O(f(n) * f(n)) = O(f^2(n))$



Note: total time is exponential (but that's OK).

$\Rightarrow$non-determinism can be eliminated by squaring the space: $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$ ■

Savitch's Theorem


Walter Savitch

Corollary: NPSPACE = PSPACE

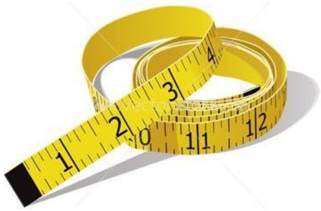
Proof:

$$\text{NPSPACE} = \bigcup_{k>1} \text{NSPACE}(n^k)$$

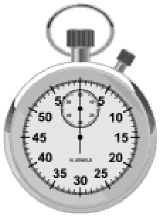
$$\subseteq \bigcup_{k>1} \text{DSPACE}(n^{2k})$$

$$= \bigcup_{k>1} \text{DSPACE}(n^k)$$

$$= \text{PSPACE}$$



i.e., polynomial space is invariant with respect to non-determinism!



Q: What about polynomial time?

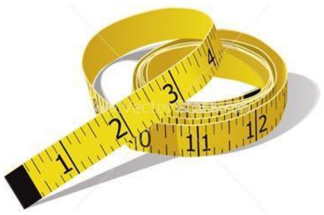
A: Still open! (P=NP)

Space & Complementation

Theorem: Deterministic space is closed under complementation, i.e.,

DSPACE(S(n)) = co-DSPACE(S(n))

$= \{\Sigma^*\text{-}L \mid L \in \text{DSPACE}(S(n))\ \}$



Proof: Simulate & negate.

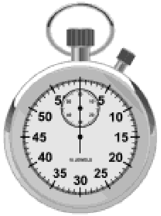
Theorem [Immerman, 1987]: Nondeterministic space is closed under complementation, i.e.

NSPACE(S(n)) = co-NSPACE(S(n))

Neil Immerman

Proof idea: Similar strategy to Savitch's theorem.

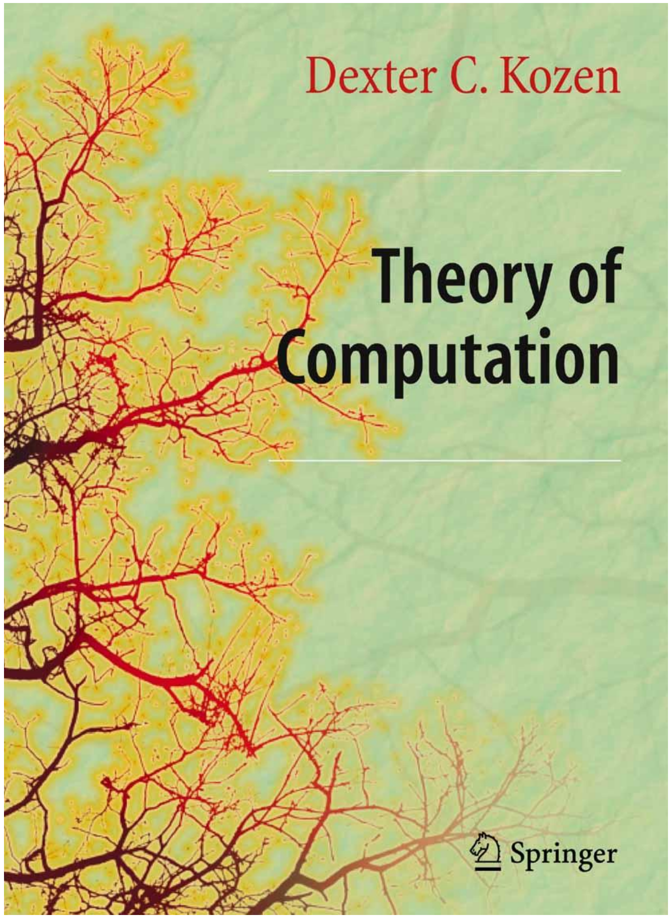
No similar result is known for any of the standard time complexity classes!



Q: Is NP = co-NP?

A: Still open!

From:



Lecture 4

The Immerman–Szelepcsényi Theorem

In 1987, Neil Immerman [65] and independently Róbert Szelepcsényi [119] showed that for space bounds $S(n) \geq \log n$, the nondeterministic space complexity class $NSPACE(S(n))$ is closed under complement. The case $S(n) = n$ gave an affirmative solution to a long-standing open problem of formal language theory: whether the complement of every context-sensitive language is context-sensitive.

Theorem 4.1 (Immerman–Szelepcsényi Theorem) *For $S(n) \geq \log n$, $NSPACE(S(n)) =$ co-$NSPACE(S(n))$.*

Proof. For simplicity we first prove the result for space-constructible $S(n)$. One can remove this condition in a way similar to the proof of Savitch's theorem (Theorem 2.7).

The proof is based on the following idea involving the concept of a *census function*. Suppose we have a finite set A of strings and a nondeterministic test for membership in A. Suppose further that we know in advance the size of the set A. Then there is a nondeterministic test for *nonmembership* in A: given y, successively guess $|A|$ distinct elements and verify that they are all in A and all different from y. If this test succeeds, then y cannot be in A.

Let M be a nondeterministic $S(n)$-space bounded Turing machine. We wish to build another such automaton N accepting the complement of

$L(M)$. Assume we have a standard encoding of configurations of M over a finite alphabet Δ, $|\Delta| = d$, such that every configuration on inputs of length n is represented as a string in $\Delta^{S(n)}$.

Assume without loss of generality that whenever M wishes to accept, it first erases its worktape, moves its heads all the way to the left, and enters a unique accept state. Thus there is a unique accept configuration $\mathtt{accept} \in \Delta^{S(n)}$ on inputs of length n. Let $\mathtt{start} \in \Delta^{S(n)}$ represent the start configuration on input x, $|x| = n$: in the start state, heads all the way to the left, worktape empty.

Because there are at most $d^{S(n)}$ configurations M can attain on input x, if x is accepted then there is an accepting computation path of length at most $d^{S(n)}$. Define A_m to be the set of configurations in $\Delta^{S(n)}$ that are reachable from the start configuration $\mathtt{start}$ in at most m steps; that is,

$$A_m = \{\alpha \in \Delta^{S(n)} \mid \mathtt{start} \xrightarrow{\leq m} \alpha\}.$$

Thus $A_0 = \{\mathtt{start}\}$ and

$$M \text{ accepts } x \quad \Leftrightarrow \quad \mathtt{accept} \in A_{d^{S(n)}}.$$

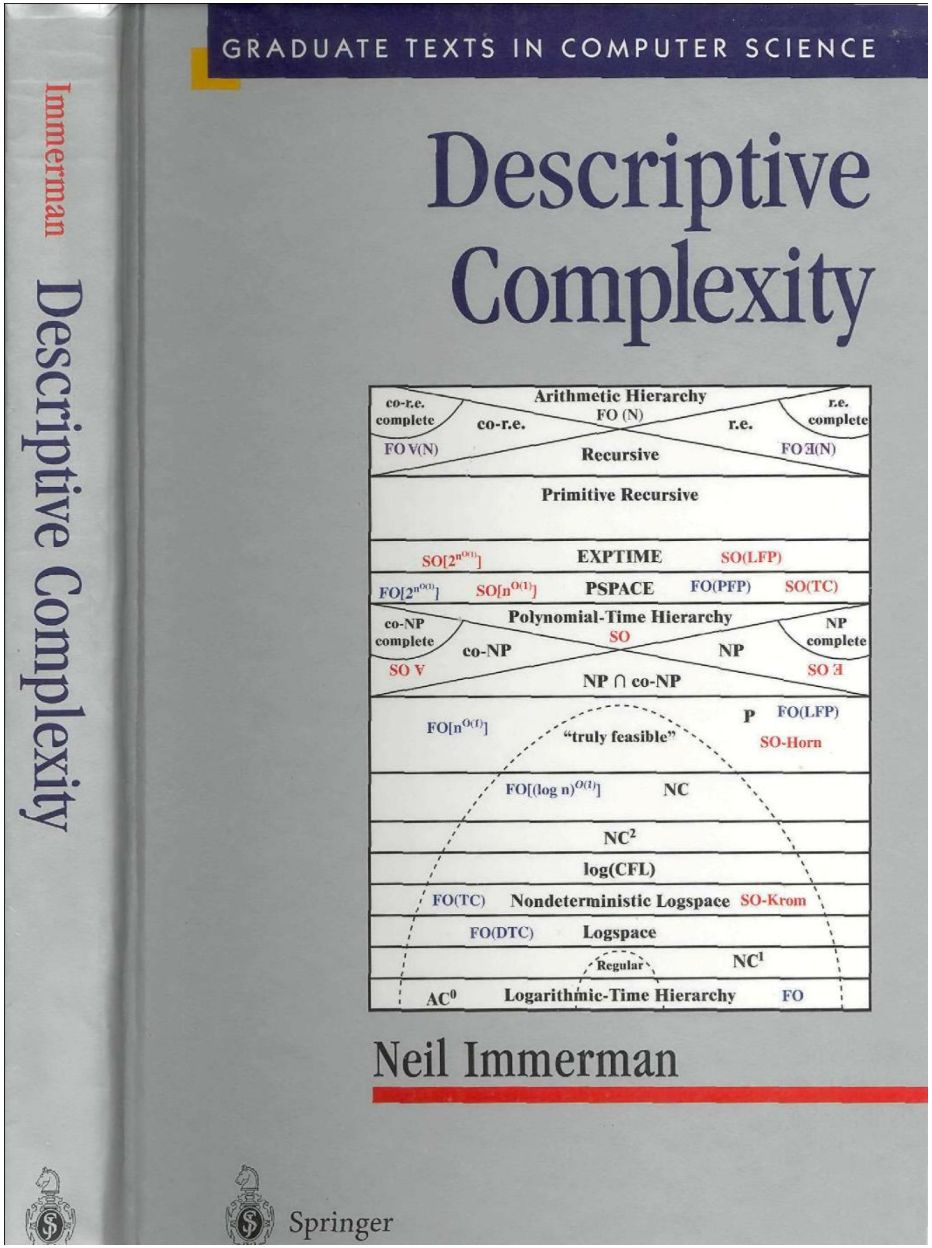
The machine N will start by laying off $S(n)$ space on its worktape. It will then proceed to compute the sizes $|A_0|, |A_1|, |A_2|, \ldots, |A_{d^{S(n)}}|$ inductively. First, $|A_0| = 1$. Now suppose $|A_m|$ has been computed and is written on a track of N's tape. Because $|A_m| \leq d^{S(n)}$, this takes up $S(n)$ space at most. To compute $|A_{m+1}|$, successively write down each $\beta \in \Delta^{S(n)}$ in lexicographical order; for each one, determine whether $\beta \in A_{m+1}$ (the algorithm for this is given below); if so, increment a counter by one. The final value of the counter is $|A_{m+1}|$. To test whether $\beta \in A_{m+1}$, nondeterministically guess the $|A_m|$ elements of A_m in lexicographic order, verify that each such α is in A_m by guessing the computation path $\mathtt{start} \xrightarrow{\leq m} \alpha$, and for each such α check whether $\alpha \xrightarrow{\leq 1} \beta$. If any such α yields β in one step, then $\beta \in A_{m+1}$; if no such α does, then $\beta \notin A_{m+1}$.

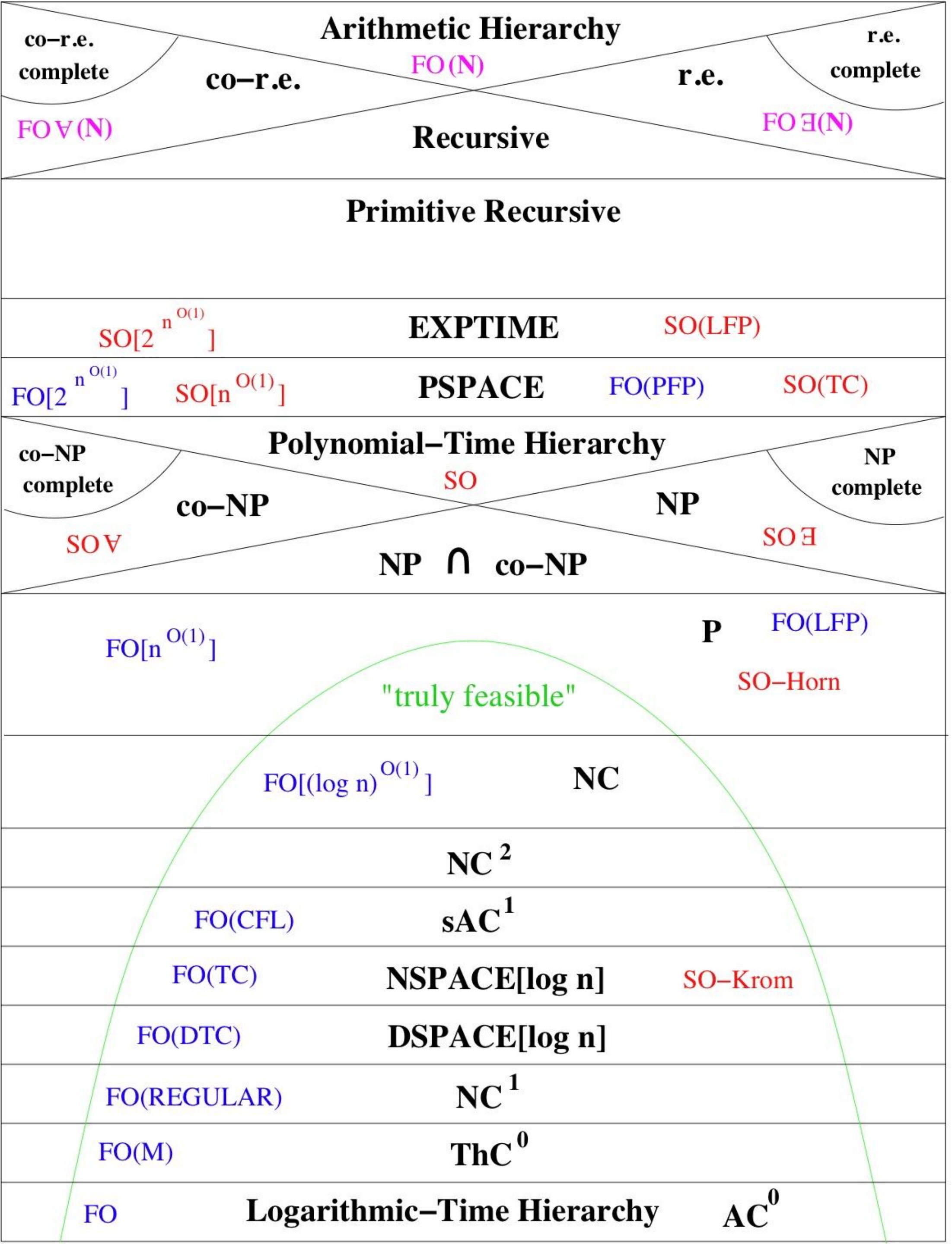
After $|A_{d^{S(n)}}|$ has been computed, in order to test $\mathtt{accept} \notin A_{d^{S(n)}}$ nondeterministically, guess the $|A_{d^{S(n)}}|$ elements of $A_{d^{S(n)}}$ in lexicographic order, verifying that each guessed α is in $A_{d^{S(n)}}$ by guessing the computation path $\mathtt{start} \xrightarrow{\leq d^{S(n)}} \alpha$, and verifying that each such α is different from $\mathtt{accept}$.

The nondeterministic machine N thus accepts the complement of $L(M)$ and can easily be programmed to run in space $S(n)$.

To remove the constructibility condition, we do the entire computation above for successive values $S = 1, 2, 3, \ldots$ approximating the true space bound $S(n)$. In the course of the computation for S, we eventually see all configurations of length S reachable from the start configuration, and can

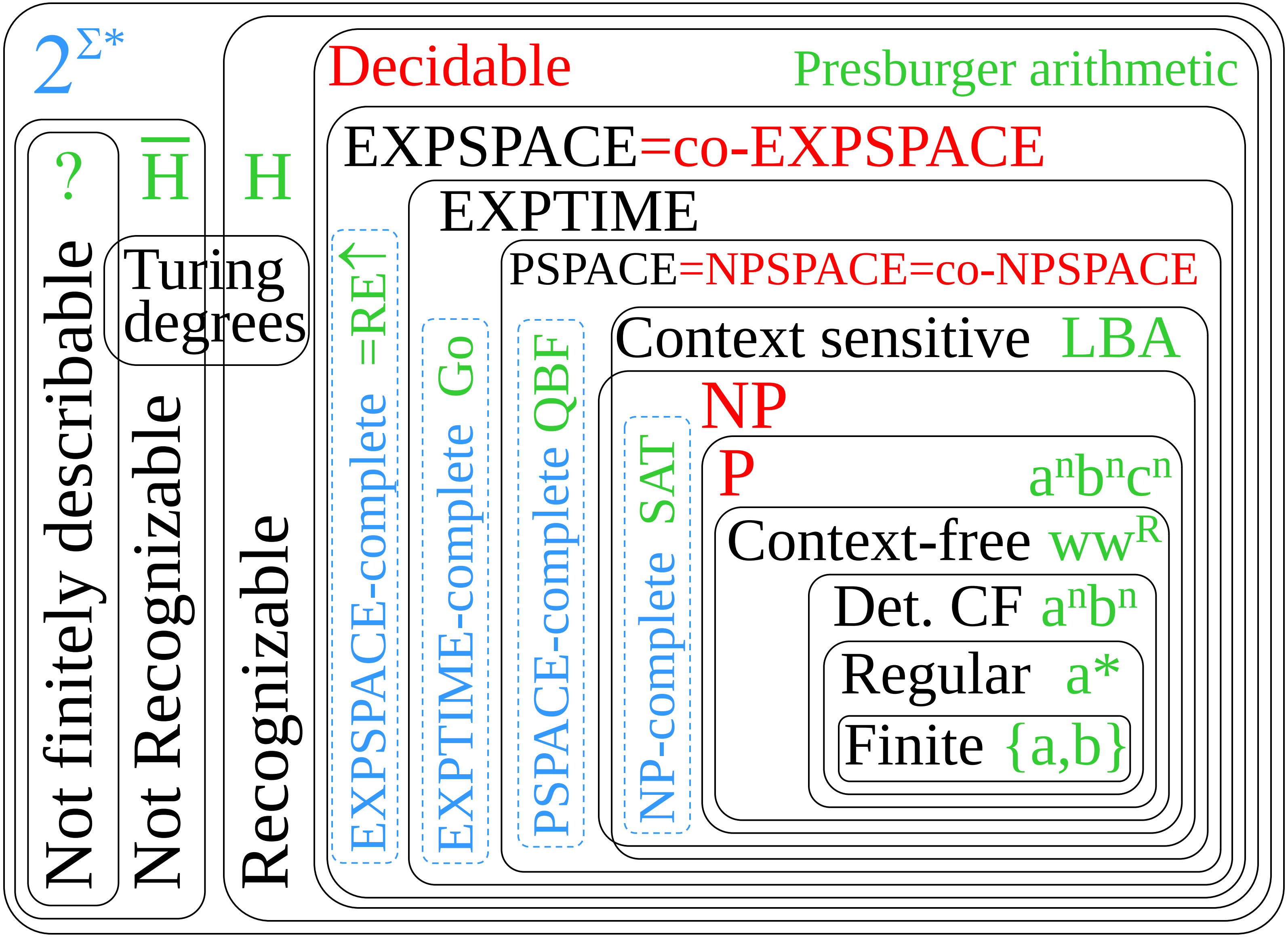
check whether M ever tries to use more than S space. If so, we know that S is too small and can restart the computation with $S + 1$. □





Neil Immerman

The Extended Chomsky Hierarchy



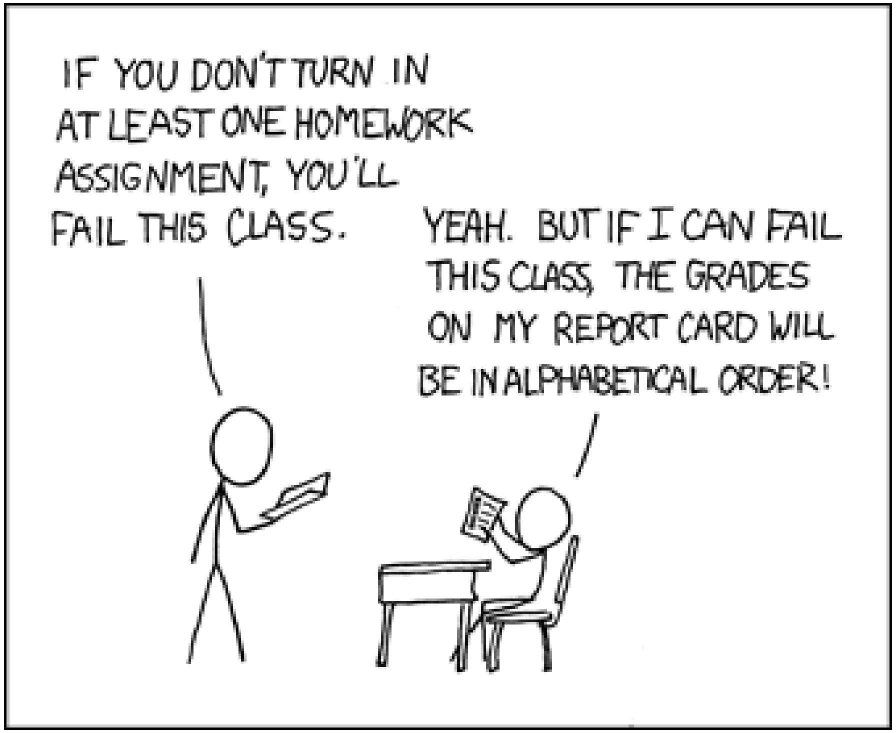
Enumeration of Resource-Bounded TMs

Q: Can we enumerate TM's for all languages in P?

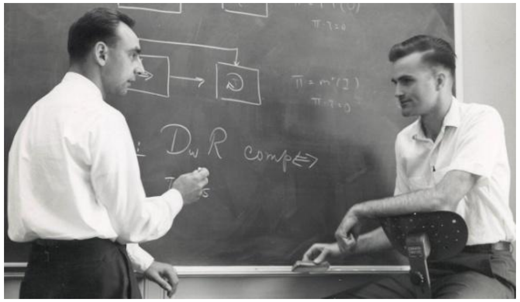
Q: Can we enumerate TM's for all languages in NP, PSPACE? EXPTIME? EXPSPACE?

Extra credit

Note: not necessarily in a lexicographic order.



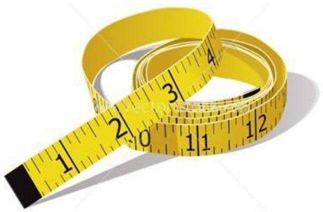
Denseness of Space Hierarchy



Juris Hartmanis   Richard Stearns

Q: How much additional space does it take to recognize more languages?

A: Very little more!



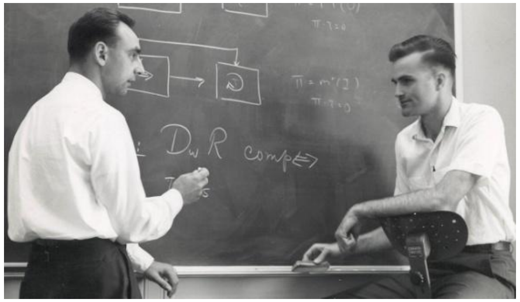
Theorem: Given two space bounds s_1 and s_2 such that $\text{Lim } s_1(n) / s_2(n)=0$ as $n\to\infty$, i.e., $s_1(n) = o(s_2(n))$, $\exists$ a decidable language L such that $L\in DSPACE(s_2(n))$ but $L\notin DSPACE(s_1(n))$.

Proof idea: Diagonalize efficiently.

Note: $s_2(n)$ must be computable within $s_2(n)$ space.

$\Rightarrow$ Space hierarchy is infinite and very dense!

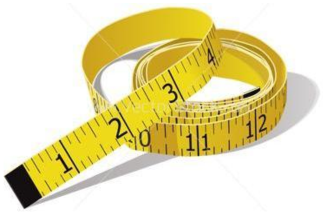
Denseness of Space Hierarchy



Juris Hartmanis   Richard Stearns

Space hierarchy is infinite

and very dense!



Examples:

$$\mathrm{DSPACE}(\log n) \subset \mathrm{DSPACE}(\log^2 n)$$

$$\mathrm{DSPACE}(n) \subset \mathrm{DSPACE}(n \log n)$$

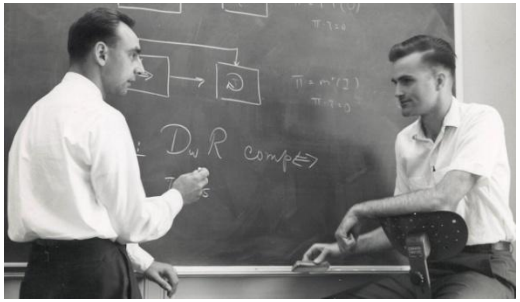
$$\mathrm{DSPACE}(n^2) \subset \mathrm{DSPACE}(n^{2.001})$$

$$\mathrm{DSPACE}(n^x) \subset \mathrm{DSPACE}(n^y)\ \forall\ 1<x<y$$

Corollary: LOGSPACE $\neq$ PSPACE

Corollary: PSPACE $\neq$ EXPSPACE

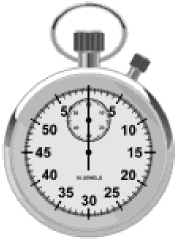
Denseness of Time Hierarchy



Juris Hartmanis   Richard Stearns

Q: How much additional time does it take to recognize more languages?

A: At most a logarithmic factor more!



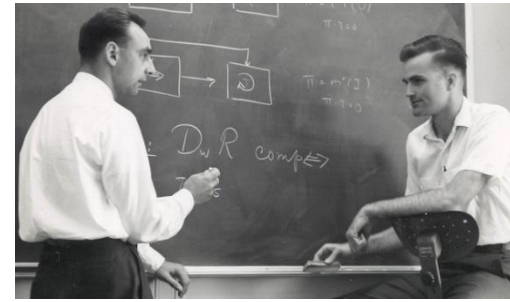
Theorem: Given two time bounds t_1 and t_2 such that $t_1(n) \cdot \log(t_1(n)) = o(t_2(n))$, $\exists$ a decidable language L such that $L \in DTIME(t_2(n))$ but $L \notin DTIME(t_1(n))$.

Proof idea: Diagonalize efficiently.

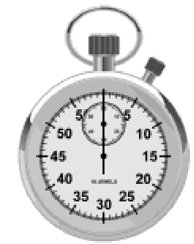
Note: $t_2(n)$ must be computable within $t_2(n)$ time.

$\Rightarrow$ Time hierarchy is infinite and pretty dense!

Denseness of Time Hierarchy



Juris Hartmanis Richard Stearns



Time hierarchy is infinite

and pretty dense!

Examples:

$$\mathrm{DTIME}(n) \subset \mathrm{DTIME}(n \log^2 n)$$

$$\mathrm{DTIME}(n^2) \subset \mathrm{DTIME}(n^{2.001})$$

$$\mathrm{DTIME}(2^n) \subset \mathrm{DTIME}(n^2 2^n)$$

$$\mathrm{DTIME}(n^x) \subset \mathrm{DTIME}(n^y)\ \forall\ 1<x<y$$

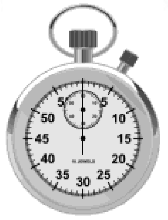
Corollary: LOGTIME $\neq$ P

Corollary: P $\neq$ EXPTIME

Complexity Classes Relationships

Theorems: LOGTIME $\subseteq$ L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ EXPSPACE $\subseteq$ . . .

Theorems: L $\neq$ PSPACE $\neq$ EXPSPACE $\neq$ . . .



Theorems: LOGTIME $\neq$ P $\neq$ EXPTIME $\neq$ . . .

Conjectures: L$\neq$NL, NL$\neq$P, P$\neq$NP, NP$\neq$PSPACE, PSPACE$\neq$EXPTIME, EXPTIME$\neq$NEXPTIME, NEXPTIME$\neq$EXPSPACE, . . .

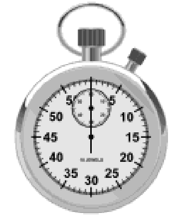
Theorem: At least two of the above conjectures are true!

Theorem: $P \neq SPACE(n)$

Open: $P \subset SPACE(n)$?

Open: $SPACE(n) \subset P$?

Open: $NSPACE(n) \neq DSPACE(n)$?



Theorem: At least two of the following conjectures are true:

$L \neq L$

$NL \neq P$

$P \neq NP$

$NP \neq PSPACE$

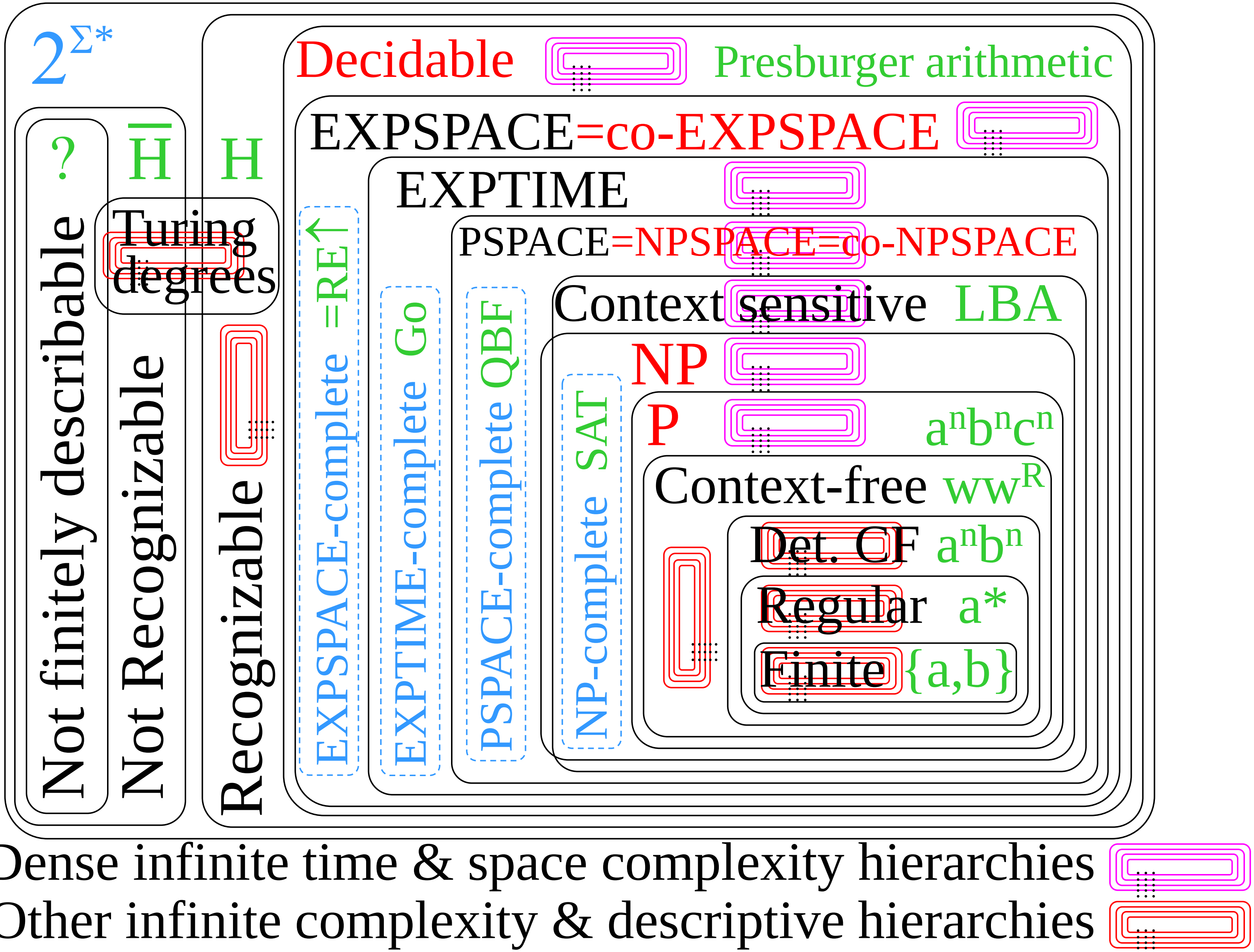
$PSPACE \neq EXPTIME$

$EXPTIME \neq NEXPTIME$

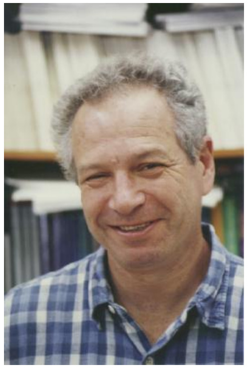
$NEXPTIME \neq EXPSPACE$

Open problems!

The Extended Chomsky Hierarchy Reloaded



Gap Theorems



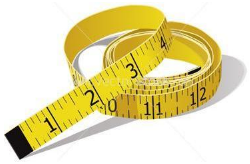
Allan Borodin

∃ arbitrarily large space & time complexity gaps!

Theorem [Borodin]: For any computable function g(n), ∃ t(n) such that DTIME(t(n)) = DTIME(g(t(n))).

Ex: DTIME(t(n)) = DTIME($2^{2^{t(n)}}$) for some t(n)

Theorem [Borodin]: For any computable function g(n), ∃ s(n) such that DSPACE(s(n)) = DSPACE(g(s(n))).



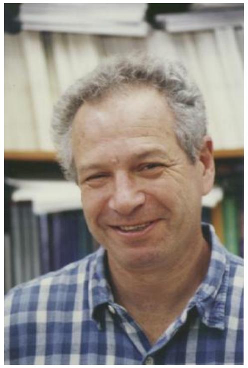
Ex: DSPACE(s(n)) = DSPACE($S(n)^{s(n)}$) for some s(n)

Proof idea: Diagonalize over TMs & construct a gap that avoids all TM complexities from falling into it.

Corollary: ∃ f(n) such that DTIME(f(n)) = DSPACE(f(n)).

Note: does not contradict the space and time hierarchy theorems, since t(n), s(n), f(n) may not be computable.

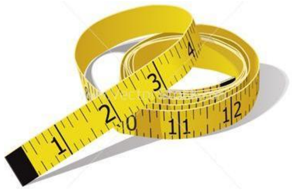
The First Complexity Gap


Allan Borodin

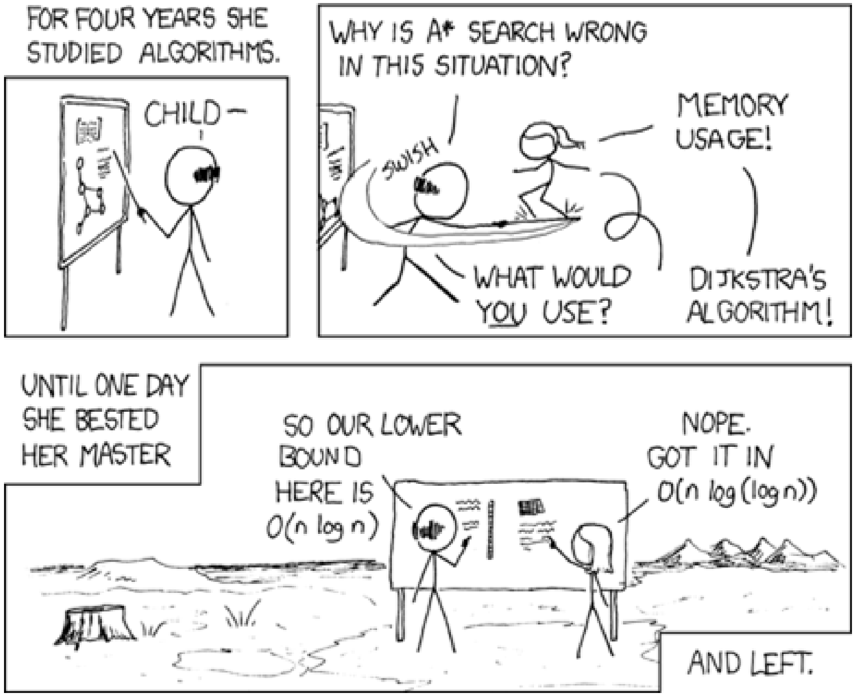
The first space "gap" is between O(1) and O(log log n)

Theorem: $L \in DSPACE(o(\log \log n)) \Rightarrow$

$L \in DSPACE(O(1)) \Rightarrow L$ is regular!



All space classes below O(log log n) collapes to O(1).

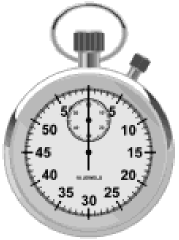


Speedup Theorem

Manuel Blum

There are languages for which there are no asymptotic space or time lower bounds for deciding them!

Theorem [Blum]: For any computable function g(n), ∃ a language L such that if TM M accepts L within t(n) time, ∃ another TM M' that accepts L within g(t(n)) time.

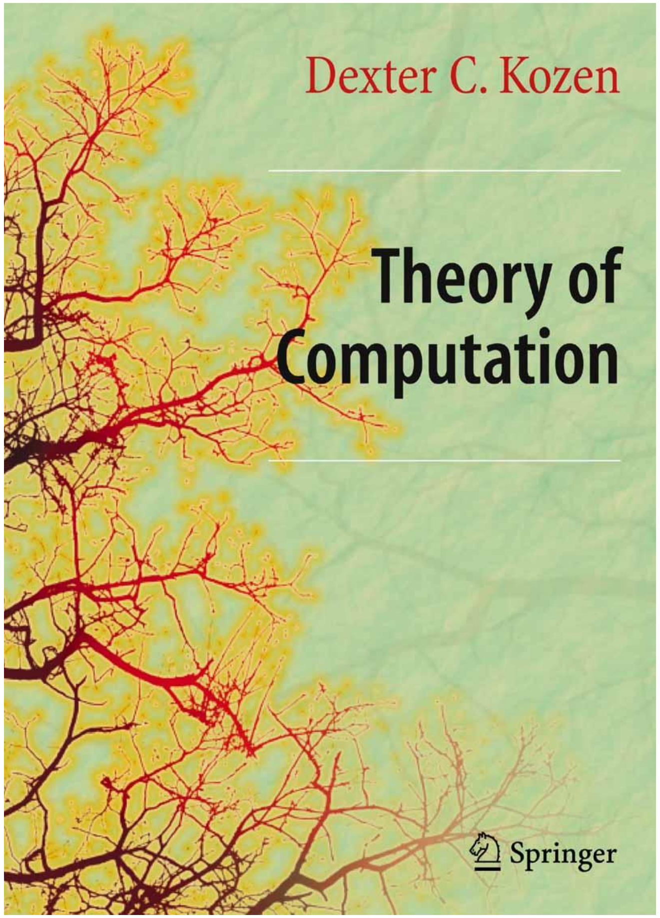


Corollary [Blum]: There is a problem such that if any algorithm solves it in time t(n), ∃ other algorithms that solve it, in times O(log t(n)), O(log(log t(n))), O(log(log(log t(n)))), ...

⇒Some problems don't have an "inherent" complexity!

Note: does not contradict the time hierarchy theorem!

From:



Lecture 32

The Gap Theorem and Other Pathology


Manuel Blum

One might get the impression from the structure of the complexity hierarchies we have studied that all problems have a natural inherent complexity, and that allowing slightly more time or space always allows more to be computed. Both these statements seem to be true for most natural problems and complexity bounds, but neither is true in general. One can construct pathological examples for which they provably fail.

For example, one can exhibit a computable function f with no asymptotically best algorithm, in the sense that for any algorithm for f running in time $T(n)$, there is another algorithm for f running in time $\log T(n)$. Thus f can be endlessly sped up. Also, there is nothing special about the log function—the result holds for any total recursive function.

For another example, one can show that there is a space bound $S(n)$ such that any function computable in space $S(n)$ is also computable in space $\log S(n)$. At first this might seem to contradict Theorem 3.1, but that theorem has a constructibility condition that is not satisfied by $S(n)$. Again, this holds for any recursive improvement, not just log.

Most of the examples of this lecture are constructed by intricate diagonalizations. They do not correspond to anything natural and would never arise in real applications. Nevertheless, they are worth studying as a way to better understand the power and limitations of complexity theory. We prove these results in terms of Turing machine time and space in this lecture; however, most of them are independent of the particular measure. A more abstract treatment is given in Supplementary Lecture J.

The first example we look at is the gap theorem, which states that there are arbitrarily large recursive gaps in the complexity hierarchy. This result is due independently to Borodin [21] and Trakhtenbrot [122].

Theorem 32.1 (Gap Theorem [21, 122]) *For any total recursive function $f : \omega \to \omega$ such that $f(x) \geq x$, there exists a time bound $T(n)$ such that $DTIME(f(T(n))) = DTIME(T(n))$; in other words, there is no set accepted by a deterministic TM in time $f(T(n))$ that is not accepted by a deterministic TM in time $T(n)$.*

Proof. Let $T_i(x)$ denote the running time of TM M_i on input x. For each n, define $T(n)$ to be the least m such that for all $i \leq n$, if $T_i(n) \leq f(m)$, then $T_i(n) \leq m$. To compute $T(n)$, start by setting $m := 0$. As long as there exists an $i \leq n$ such that $m < T_i(n) \leq f(m)$, set $m := T_i(n)$. This process must terminate, because there are only finitely many $i \leq n$. The value of $T(n)$ is the final value of m.

Now we claim that $T(n)$ satisfies the requirements of the theorem. Suppose M_i runs in time $f(T(n))$. Thus $T_i(n) \leq f(T(n))$ a.e.[1] By construction of T, for sufficiently large $n \geq i$, $T_i(n) \leq T(n)$. □

What we have actually proved is stronger than the statement of the theorem. The theorem states that for any deterministic TM M_i running in time $f(T(n))$, there is an equivalent deterministic TM M_j running in time $T(n)$. But what we have actually shown is that any deterministic TM running in time $f(T(n))$ also runs in time $T(n)$.

Of course, all these bounds hold a.e., but we can make them hold everywhere by encoding the values on small inputs in the finite control and computing them by table lookup.

The next example gives a set for which any algorithm can be sped up arbitrarily many times by an arbitrary preselected recursive amount. This result is due to Blum [17].

Theorem 32.2 (Speedup Theorem [17]) *Let $T_i(x)$ denote the running time of TM M_i on input x. Let $f : \omega \to \omega$ be any monotone total recursive function such that $f(n) \geq n^2$. There exists a recursive set A such that for any TM M_i accepting A, there is another TM M_j accepting A with $f(T_j(x)) < T_i(x)$ a.e.*

[1] "a.e." means "almost everywhere" or "for all but finitely many n". Also, "i.o." means, "infinitely often" = "for infinitely many n".

Proof. Let f^n denote the n-fold composition of f with itself:

$$f^n \stackrel{\text{def}}{=} \underbrace{f \circ f \circ \cdots \circ f}_{n}.$$

Thus f^0 is the identity function, $f^1 = f$, and $f^{m+n} = f^m \circ f^n$. For example, if $f(m) = m^2$, then $f^n(m) = m^{2^n}$, and if $f(m) = 2^m$, then $f^n(m)$ is an iterated exponential involving a stack of 2's of height n.

We construct by diagonalization a set $A \subseteq 0^*$ such that

(i) for any machine M_i accepting A, $T_i(0^n) > f^{n-i}(2)$ a.e.,[2] and

(ii) for all k, there exists a machine M_j accepting A such that $T_j(0^n) \leq f^{n-k}(2)$ a.e.

This achieves our goal, because for any machine M_i accepting A, (ii) guarantees the existence of a machine M_j accepting A such that $T_j(0^n) \leq f^{n-i-1}(2)$ a.e.; but then

$$\begin{array}{rcll} f(T_j(0^n)) & \leq & f(f^{n-i-1}(2)) \text{ a.e.} & \text{by monotonicity of } f \\ & = & f^{n-i}(2) & \\ & < & T_i(0^n) \text{ a.e.} & \text{by (i).} \end{array}$$

Now we turn to the construction of the set A. Let $M_0, M_1, \ldots$ be a list of all one-tape Turing machines with input alphabet $\{0\}$. Let N be an enumeration machine that carries out the following simulation. It maintains a finite *active list* of descriptions of machines currently being simulated. We assume that a description of M_i suitable for universal simulation is easily obtained from the index i.

The computation of N proceeds in stages. Initially, the active list is empty. At stage n, N puts the next machine M_n at the end of the active list. It then simulates the machines on the active list in order, smallest index first. For each such M_i, it simulates M_i on input 0^n for $f^{n-i}(2)$ steps. It picks the first one that halts within its allotted time and does the opposite: if M_i rejects 0^n, N declares $0^n \in A$, and if M_i accepts 0^n, N declares $0^n \notin A$. This ensures that $L(M_i) \neq A$. It then deletes M_i from the active list. If no machine on the active list halts within its allotted time, then N just declares $0^n \notin A$.

This construction ensures that any machine M_i that runs in time $f^{n-i}(2)$ i.o. does not accept A. The machine M_i is put on the active list at stage i. Thereafter, if M_i halts within time $f^{n-i}(2)$ on 0^n but is not

[2] We are regarding $f^{n-i}(2)$ as a function of n with i a fixed constant. Thus "i.o." and "a.e." in this context meant to be interpreted as "for infinitely many n" and "for all but finitely many n", respectively.

chosen for deletion, then some higher priority machine on the active list must have been chosen; but this can happen only finitely many times. So if M_i halts within time $f^{n-i}(2)$ on 0^n i.o., then eventually M_i will be the highest priority machine on the list and will be chosen for deletion, say at stage n. At that point, 0^n will be put into A iff $0^n \notin L(M_i)$, ensuring $L(M_i) \neq A$. This establishes condition (i) above.

For condition (ii), we need to show that for all k, A is accepted by a one-tape TM N_k running in time $f^{n-k}(2)$ a.e. The key idea is to hard-code the first m stages of the computation of N in the finite control of N_k for some sufficiently large m. Note that for each M_i, either

(A) $T_i(0^n) \leq f^{n-i}(2)$ i.o., in which case there is a stage $m(i)$ at which N deletes M_i from the active list; or

(B) $T_i(0^n) > f^{n-i}(2)$ a.e., in which case there is a stage $m(i)$ after which M_i always exceeds its allotted time.

Let $m = \max_{i \leq k} m(i)$. We cannot determine the $m(i)$ or m effectively (Miscellaneous Exercise 105), but we do know that they exist. The machine N_k has a list of elements $0^n \in A$ for $n \leq m$ hard-coded in its finite control. On such inputs, it simply does a table lookup to determine whether $0^n \in A$ and accepts or rejects accordingly. On inputs 0^n for $n > m$, it simulates the action of N on stages $m+1, m+2, \ldots, n$ starting with a certain active list, which it also has hard-coded in its finite control. The active list it starts with is N's active list at stage m with all machines M_i for $i \leq k$ deleted. This does not change the status of $0^n \in A$: for each M_i with $i \leq k$, in case A it has already been deleted from the active list by stage m, and in case B it will always exceed its allotted time after stage m, so it will never be a candidate for deletion. The simulation will therefore behave exactly as N would at stage m and beyond. The machine N_k can thus determine whether $0^n \in A$ and accept or reject accordingly.

It remains to estimate the running time of N_k on input 0^n. If $n \leq m$, N_k takes linear time, enough time to read the input and do the table lookup. If $n > m$, N_k must simulate at most $n-k$ machines on the active list on $n-m$ inputs, each for at most $f^{n-k-1}(2)$ steps. Under mild assumptions on the encoding scheme, interpreting the binary representation of the index i as a description of M_i, M_i has at most $\log i$ states, at most $\log i$ tape symbols, and at most $\log i$ transitions in its finite control, and one step of M_i can be simulated in roughly $c(\log i)^2$ steps of N_k. Thus the total time needed for all the simulations is at most $cn^2(\log n)^2 f^{n-k-1}(2)$. But

$$\begin{aligned} cn^2(\log n)^2 &\leq 2^{2^{n-k-1}} \text{ a.e.} \\ &\leq f^{n-k-1}(2) \quad \text{because } f(m) \geq m^2, \end{aligned}$$

therefore

$$\begin{aligned} cn^2(\log n)^2 f^{n-k-1}(2) &\leq (f^{n-k-1}(2))^2 \text{ a.e.} \\ &\leq f(f^{n-k-1}(2)) \\ &= f^{n-k}(2). \end{aligned}$$

□

There are a few interesting observations we can make about the proof of Theorem 32.2.

First, the "mild assumptions" on the encoding scheme are inconsequential. If they are not satisfied, the condition $f(m) \geq m^2$ can be strengthened accordingly. We only need to know that the overhead for universal simulation of Turing machines is bounded by a total recursive function.

The value $m = \max_{i \leq k} m(i)$ in the proof of Theorem 32.2 cannot be obtained effectively. We know that for each M_i there exists such an m, but it is undecidable whether M_i falls in case A or case B, so we do not know whether to delete M_i from the active list. Indeed, it is impossible to obtain a machine for A running in time $f^{n-k}(2)$ effectively from k (Miscellaneous Exercise 105).

Abstract Complexity Theory


Manuel Blum

Complexity theory can be machine-independent!

Instead of referring to TM's, we state simple axioms that any complexity measure Φ must satisfy.

Example: the Blum axioms:

1) $\Phi(M,w)$ is finite iff $M(w)$ halts; and

2) The predicate "$\Phi(M,w)=n$" is decidable.

Theorem [Blum]: Any complexity measure satisfying these axioms gives rise to hierarchy, gap, & speedup theorems.

Corollary: Space & time measures satisfy these axioms.

AKA "Axiomatic complexity theory [Blum, 1967]

Alternation

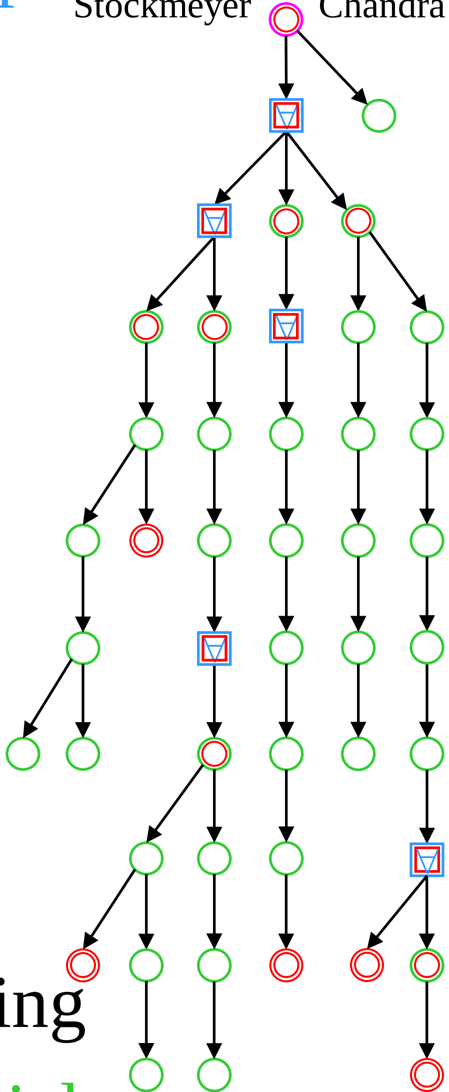

Stockmeyer Chandra

Alternation: generalizes non-determinism, where each state is either "existential" or "universal"

Old: existential states ∃

New: universal states ∀

- Existential state is accepting iff any of its child states is accepting (OR)
- Universal state is accepting iff all of its child states are accepting (AND)
- Alternating computation is a "tree".
- Final states are accepting
- Non-final states are rejecting
- Computation accepts iff initial state is accepting



Note: in non-determinism, all states are existential

Alternation

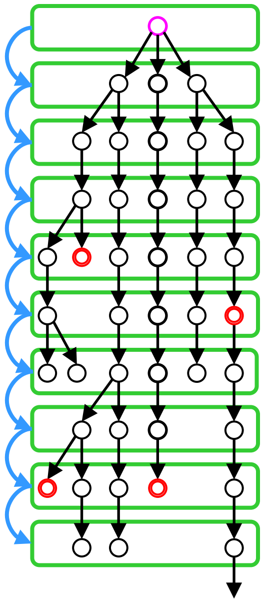
Stockmeyer Chandra

Theorem: a k-state alternating finite automaton can be converted into an equivalent 2^k-state non-deterministic FA.

Proof idea: a generalized powerset construction.

Theorem: a k-state alternating finite automaton can be converted into an equivalent 2^{2^k}-state deterministic FA.

Proof: two composed powerset constructions.



Def: alternating Turing machine is an alternating FA with an unbounded read/write tape.

Theorem: alternation does not increase the language recognition power of Turing machine.

Proof: by simulation.

Alternating Complexity Classes


Stockmeyer Chandra

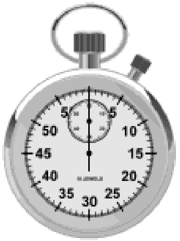
Properties!

Def: ATIME(t(n))={L | L is decidable in time O(t(n)) by some alternating TM}

Def: ASPACE(s(n))={L | L decidable in space O(s(n)) by some alternating TM}

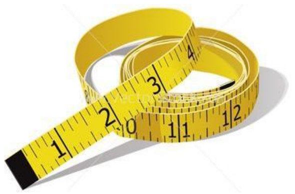
Def: $AP = \bigcup_{\forall k>1} ATIME(n^k)$

AP ≡ alternating polynomial time



Def: $APSPACE = \bigcup_{\forall k>1} ASPACE(n^k)$

APSPACE ≡ alternating polynomial space



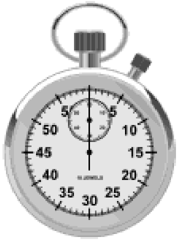
Alternating Complexity Classes


Stockmeyer


Chandra

Def: $\text{AEXPTIME} = \bigcup_{\forall k>1} \text{ATIME}(2^{n^k})$

AEXPTIME ≡ alternating exponential time

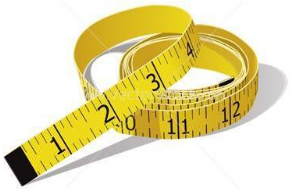


Def: $\text{AEXPSPACE} = \bigcup_{\forall k>1} \text{ASPACE}(2^{n^k})$

AEXPSPACE ≡ alternating exponential space

Def: AL = ALOGSPACE = ASPACE(log n)

AL ≡ alternating logarithmic space



Note: AP, ASPACE, AL are model-independent

Alternating Space/Time Relations

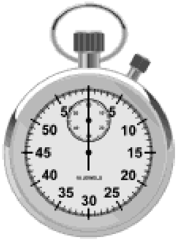

Stockmeyer

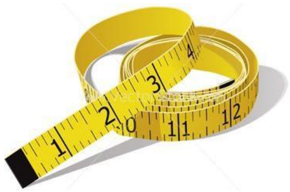

Chandra

Theorem: $P \subseteq NP \subseteq AP$

Open: $NP = AP$?

Open: $P = AP$?





Corollary: $P=AP \Rightarrow P=NP$

Theorem: $ATIME(f(n)) \subseteq DSPACE(f(n)) \subseteq ATIME(f^2(n))$

Theorem: $PSPACE = NPSPACE \subseteq APSPACE$

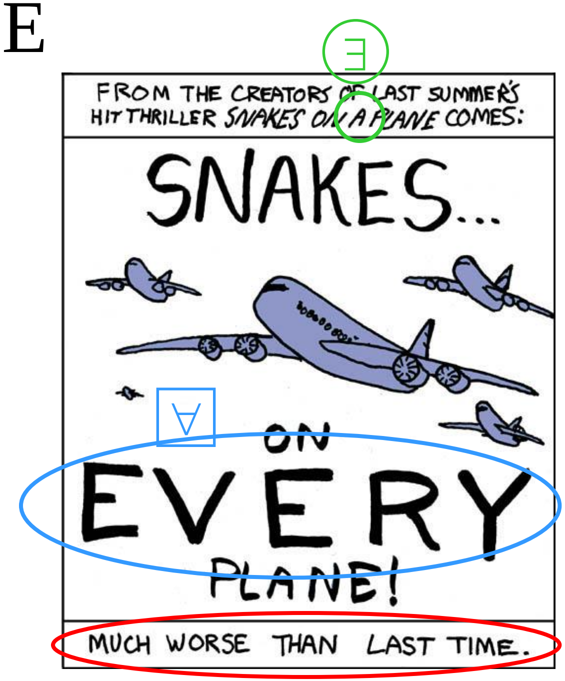
Theorem: $ASPACE(f(n)) \subseteq DTIME(c^{f(n)})$

Theorem: $AL = P$

Theorem: $AP = PSPACE$

Theorem: $APSPACE = EXPTIME$

Theorem: $AEXPTIME = EXPSPACE$



Quantified Boolean Formula Problem

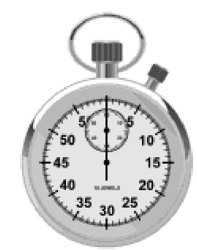
Def: Given a fully quantified Boolean formula, where each variable is quantified existentially or universally, does it evaluate to "true"?

Example: Is "$\forall$ x $\exists$ y $\exists$ z $(x \wedge z) \vee y$" true?

- Also known as quantified satisfiability (QSAT)
- Satisfiability (one $\exists$ only) is a special case of QBF

Theorem: QBF is PSPACE-complete.
Proof idea: combination of [Cook] and [Savitch].

Theorem: QBF $\in$ TIME(2^n)
Proof: recursively evaluate all possibilities.



Theorem: QBF $\in$ DSPACE(n)
Proof: reuse space during exhaustive evaluations.

Theorem: QBF $\in$ ATIME(n)
Proof: use alternation to guess and verify formula.

QBF and Two-Player Games

- SAT solutions can be succinctly (polynomially) specified.
- It is not known how to succinctly specify QBF solutions.
- QBF naturally models winning strategies for two-player games:

∃ a move for player A

  ∀ moves for player B

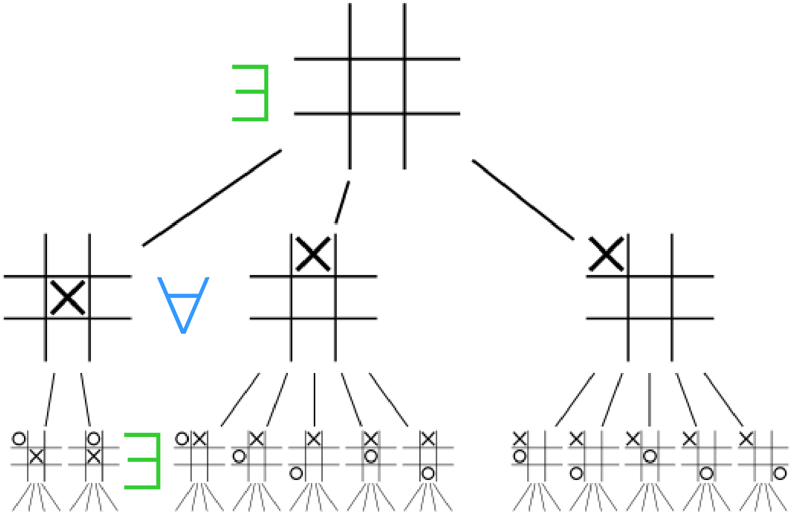
    ∃ a move for player A

      ∀ moves for player B

        ∃ a move for player A

          ⋮

            player A has a winning move!



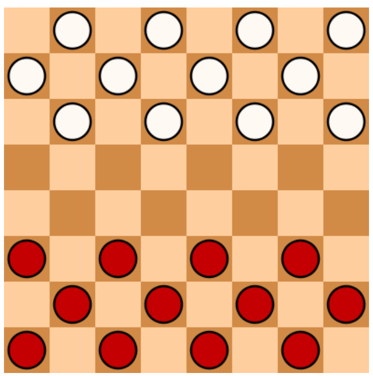
QBF and Two-Player Games

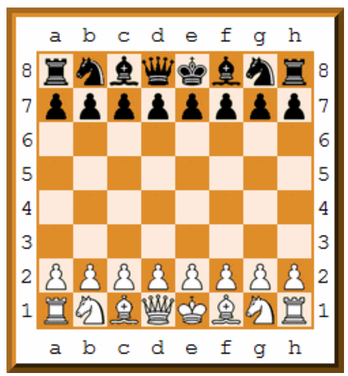
Theorem: Generalized Checkers is EXPTIME-complete

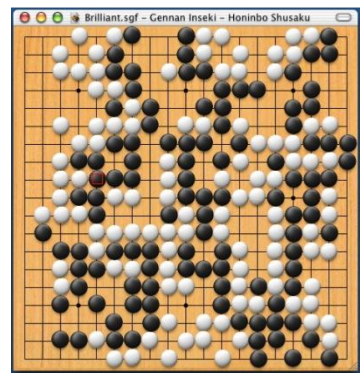
Theorem: Generalized Chess is EXPTIME-complete.

Theorem: Generalized Go is EXPTIME-complete.

Theorem: Generalized Othello is PSPACE-complete.







The Polynomial Hierarchy

Meyer Stockmeyer

Idea: bound # of "existential" / "universal" states

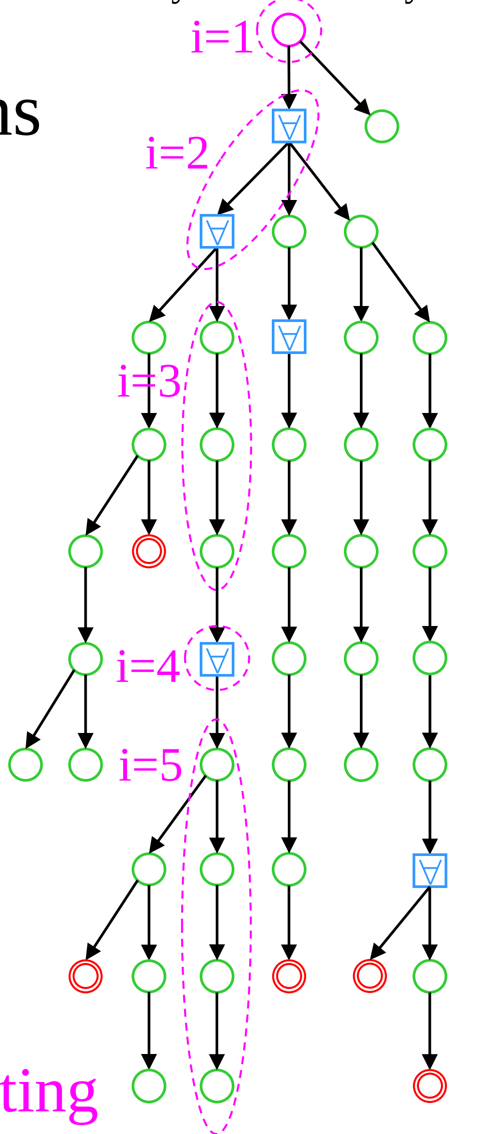
Old: unbounded existential / universal states

New: at most i existential / universal alternations

Def: a Σ_i-alternating TM has at most i runs of quantified steps, starting with existential ∃

Def: a Π_i-alternating TM has at most i runs of quantified steps, starting with universal ∀

Note: Π_i- and Σ_i- alternation-bounded TMs are similar to unbounded alternating TMs



Σ_5-alternating

The Polynomial Hierarchy

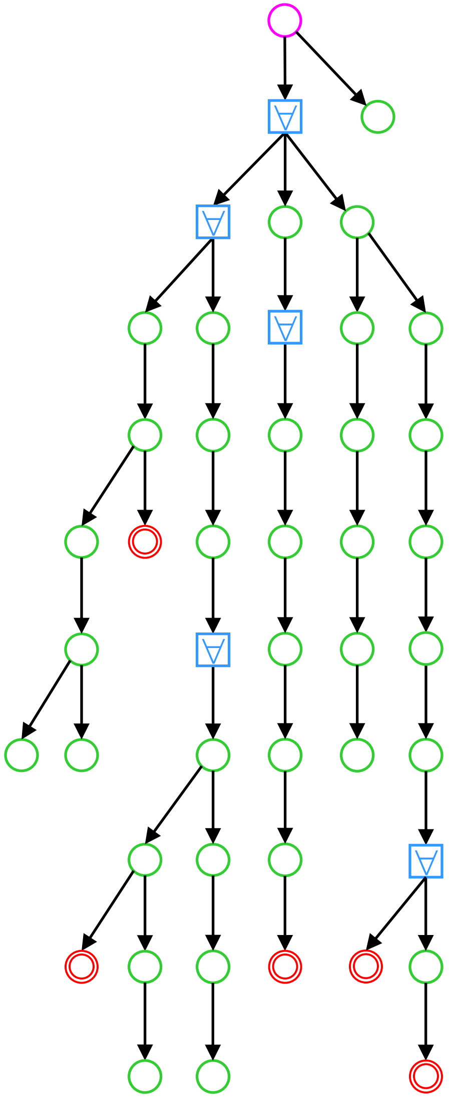
Meyer Stockmeyer

Def: Σ_iTIME(t(n))={L | L is decidable within time O(t(n)) by some Σ_i-alternating TM}

Def: Σ_iSPACE(s(n))={L | L is decidable within space O(s(n)) by some Σ_i-alternating TM}

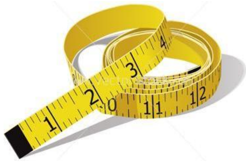
Def: Π_iTIME(t(n))={L | L is decidable within time O(t(n)) by some Π_i-alternating TM}

Def: Π_iSPACE(s(n))={L | L is decidable within space O(s(n)) by some Π_i-alternating TM}



Def: $$\Sigma_i P = \bigcup_{\forall k>1} \Sigma_i \mathrm{TIME}(n^k)$$

Def: $$\Pi_i P = \bigcup_{\forall k>1} \Pi_i \mathrm{TIME}(n^k)$$

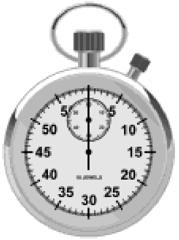


The Polynomial Hierarchy

Meyer Stockmeyer

Def: $\Sigma PH = \bigcup_{\forall i>1} \Sigma_i P$

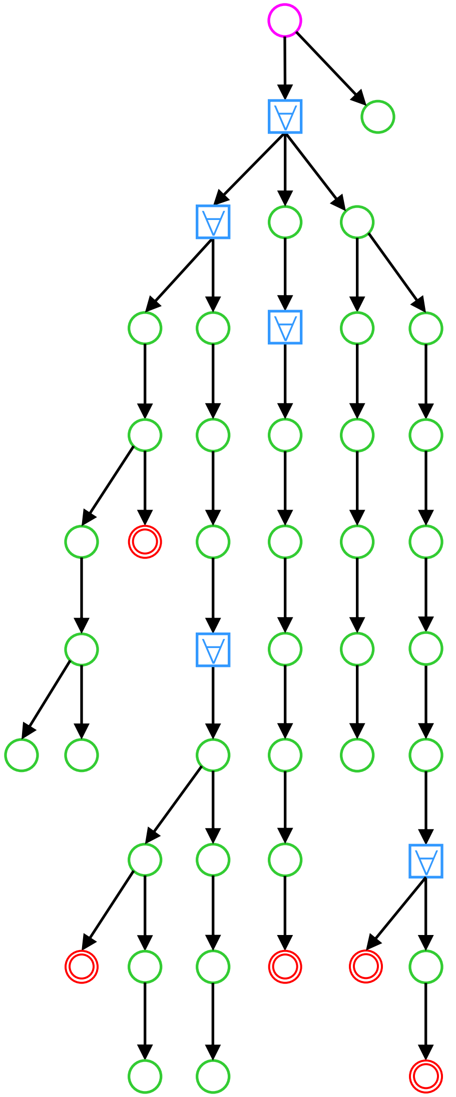
Def: $\Pi PH = \bigcup_{\forall i>1} \Pi_i P$



Theorem: $\Sigma PH = \Pi PH$

Def: The Polynomial Hierarchy $PH = \Sigma PH$

$\Rightarrow$ Languages accepted by polynomial time, unbounded-alternations TMs



Theorem: $\Sigma_0 P = \Pi_0 P = P$

Theorem: $\Sigma_1 P = NP$, $\Pi_1 P =$ co-NP

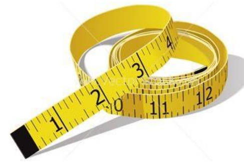
Theorem: $\Sigma_i P \subseteq \Sigma_{i+1} P$, $\Pi_i P \subseteq \Pi_{i+1} P$

Theorem: $\Sigma_i P \subseteq \Pi_{i+1} P$, $\Pi_i P \subseteq \Sigma_{i+1} P$

The Polynomial Hierarchy

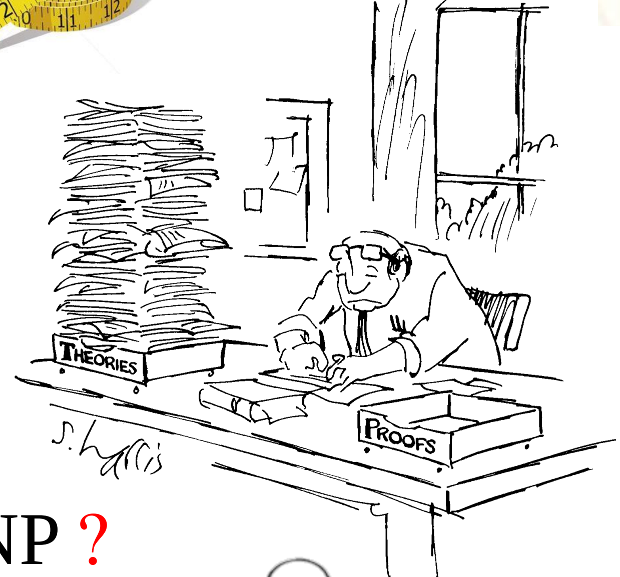

Meyer Stockmeyer

Theorem: $\Sigma_i P \subseteq PSPACE$



Theorem: $\Pi_i P \subseteq PSPACE$

Theorem: $PH \subseteq PSPACE$



Open: $PH = PSPACE$?

Open: $\Sigma_0 P = \Sigma_1 P$? $\Leftrightarrow$ $P=NP$?

Open: $\Pi_0 P = \Pi_1 P$? $\Leftrightarrow$ $P=co\text{-}NP$?

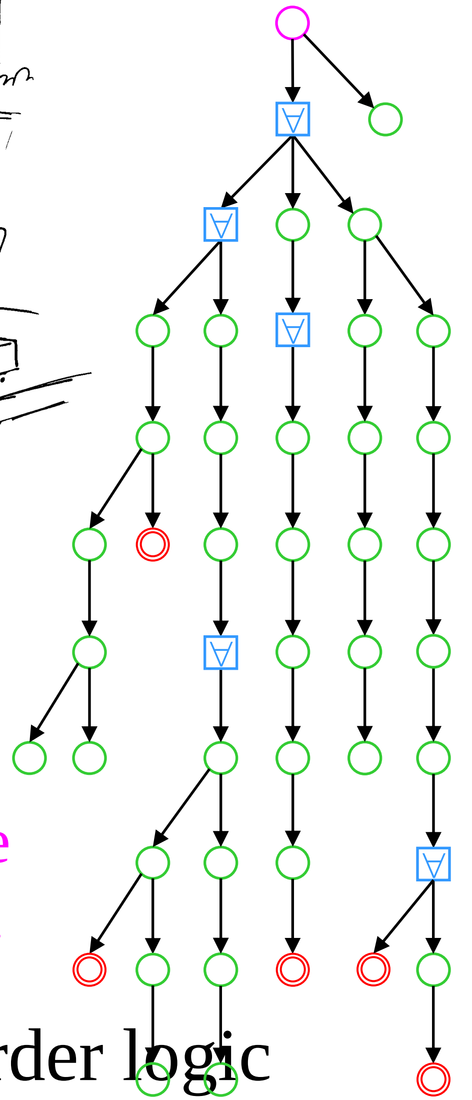
Open: $\Sigma_1 P = \Pi_1 P$? $\Leftrightarrow$ $NP=co\text{-}NP$?

Open: $\Sigma_k P = \Sigma_{k+1} P$ for any k ?

Open: $\Pi_k P = \Pi_{k+1} P$ for any k ?

Open: $\Sigma_k P = \Pi_k P$ for any k ?

Infinite number of "P=NP"–type open problems!



Theorem: PH = languages expressible by 2nd-order logic

The Polynomial Hierarchy


Meyer Stockmeyer

Open: Is the polynomial hierarchy infinite ?

Theorem: If any two successive levels conicide ($\Sigma_k P = \Sigma_{k+1} P$ or $\Sigma_k P = \Pi_k P$ for some k) then the entire polynomial hierarchy collapses to that level (i.e., $PH = \Sigma_k P = \Pi_k P$).

Corollary: If P = NP then the entire polynomial hierarchy collapses completely (i.e., PH = P = NP).

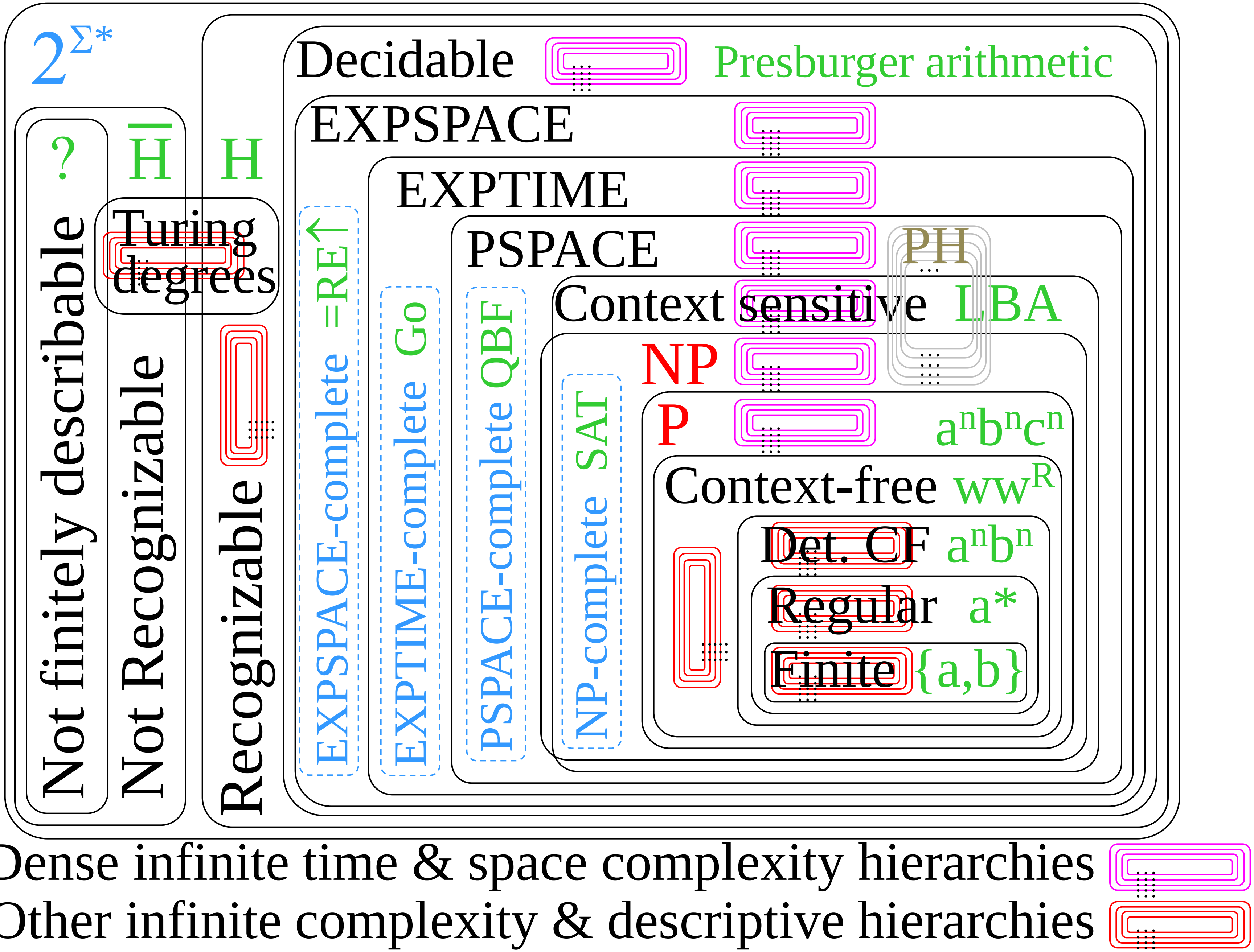
Theorem: $P=NP \Leftrightarrow P=PH$

Corollary: To show $P \neq NP$, it suffices to show $P \neq PH$.

Theorem: There exist oracles that separate $\Sigma_k P \neq \Sigma_{k+1} P$.

Theorem: PH contains almost all well-known complexity classes in PSPACE, including P, NP, co-NP, BPP, RP, etc.

The Extended Chomsky Hierarchy Reloaded



Dense infinite time & space complexity hierarchies

Other infinite complexity & descriptive hierarchies

Probabilistic Turing Machines

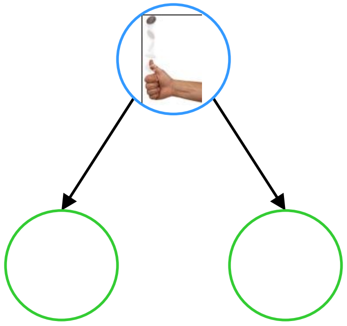
Idea: allow randomness / coin-flips during computation

Old: nondeterministic states

New: random states changes via coin-flips

- Each coin-flip state has two successor states

Def: Probability of branch B is $\Pr[B] = 2^{-k}$ where k is the # of coin-flips along B.



Def: Probability that M accepts w is sum of the probabilities of all accepting branches.

Def: Probability that M rejects w is 1 – (probability that M accepts w).

Def: Probability that M accepts L with probability ε if:

$w \in L \Rightarrow$ probability(M accepts w) $\geq 1-\varepsilon$

$w \notin L \Rightarrow$ probability(M rejects w) $\geq 1-\varepsilon$

Probabilistic Turing Machines

Def: BPP is the class of languages accepted by probabilistic polynomial time TMs with error $\varepsilon = 1/3$.

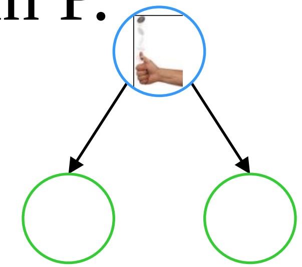
Note: BPP Bounded-error Probabilistic Polynomial time

Theorem: any error threshold $0<\varepsilon<1/2$ can be substituted.

Proof idea: run the probabilistic TM multiple times and take the majority of the outputs.

Theorem [Rabin, 1980]: Primality testing is in BPP.

Theorem [Agrawal et al., 2002]: Primality testing is in P.



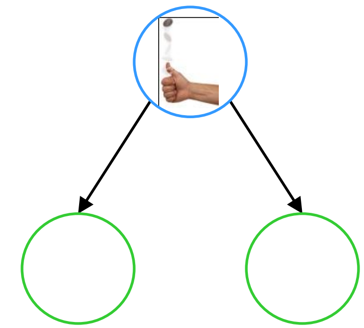
Note: BPP is one of the largest practical classes of problems that can be solved effectively.

Theorem: BPP is closed under complement (BPP=co-BPP).

Open: BPP $\subseteq$ NP ?

Open: NP $\subseteq$ BPP ?

Probabilistic Turing Machines



Theorem: BPP $\subseteq$ PH

Theorem: P=NP $\Rightarrow$ BPP=P

Theorem: NP $\subseteq$ BPP $\Rightarrow$ PH $\subseteq$ BPP

Note: the former is unlikely, since this would imply efficient randomized algorithms for many NP-hard problems.

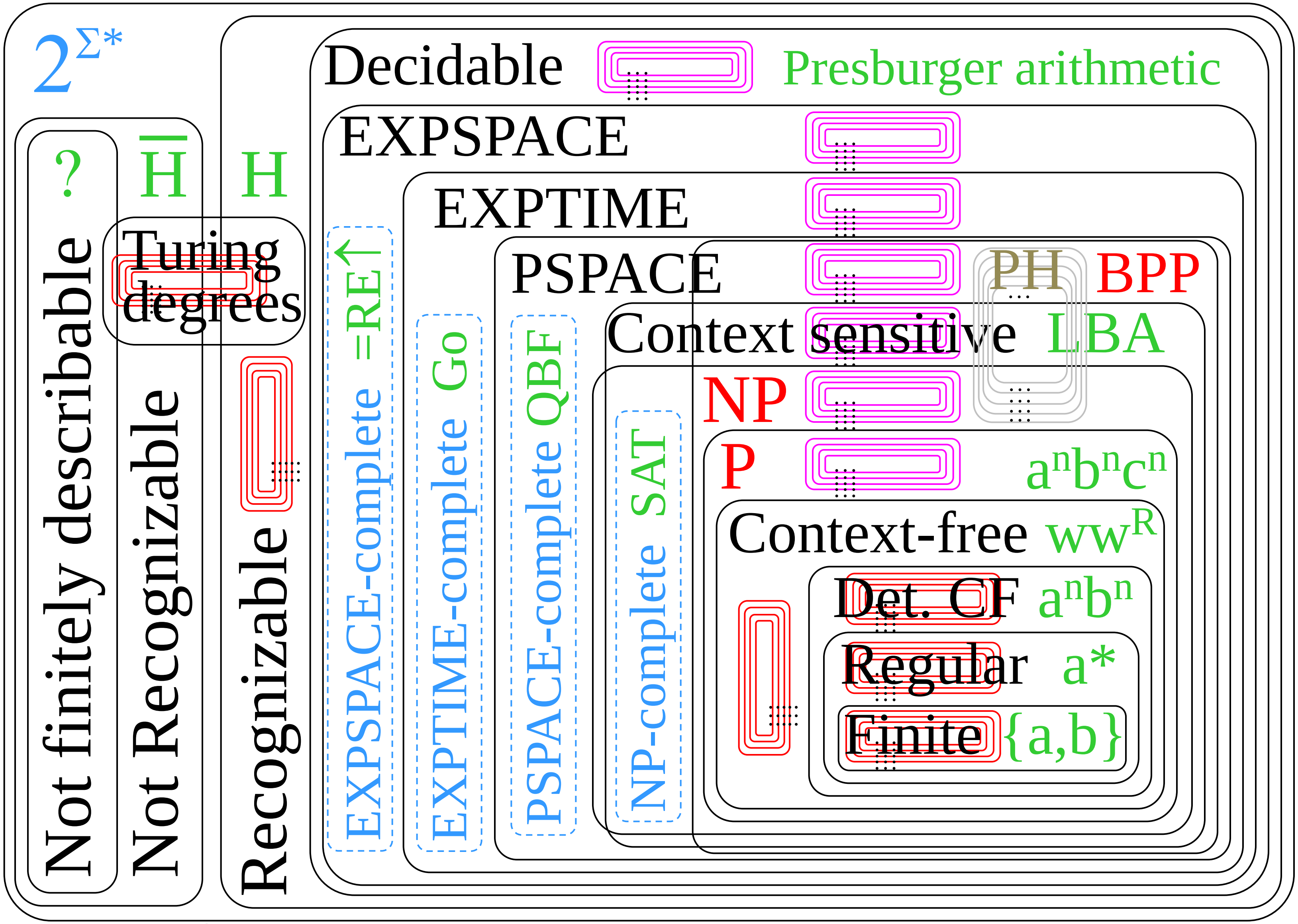
Def: A pseudorandom number generator (PRNG) is an algorithm for generating number sequences that approximates the properties of random numbers.

Theorem: The existance of strong PRNGs implies that P=BPP.

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

The Extended Chomsky Hierarchy Reloaded



Dense infinite time & space complexity hierarchies

Other infinite complexity & descriptive hierarchies

Complexity Zoo - Qwiki - Mozilla Firefox

http://qwiki.stanford.edu/wiki/Complexity_Zoo

Log in / create account

page | discussion | view source | history

Complexity Zoo

navigation

- Main Page
- Browse Categories
- Recent Changes
- Random Page
- Community Portal
- Help!
- FAQ
- Sign up!

search

Go | Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

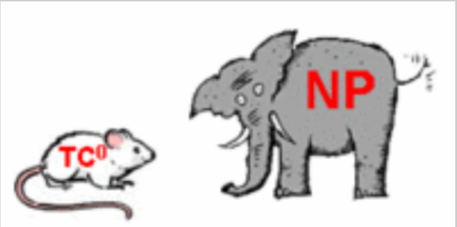
Introduction

Welcome to the **Complexity Zoo**... There are now 489 classes and counting!

This information was originally moved from http://www.complexityzoo.com/ in August 2005, and is currently under the watchful eyes of its original creators:

Zookeeper: Scott Aaronson
Veterinarian: Greg Kuperberg

Tour Guide: Christopher Granade



what's your problem?

Errors? Omissions? Misattributions? Your favorite class not here? Then please contribute to the zoo as you see fit by signing up and clicking on the edit links. Please include references, or better yet links to papers if available.

To create a new class, click on the edit link of the class before or after the one that you want to add and copy the format of that class. (The classes are alphabetized by their tag names.) Then add the class to the table of contents and increment the total number of classes. After this, you can use the side edit links to edit the individual sections. For more on using the wiki language, see our simple wiki help page.

If you would like to contribute but feel unable to make the updates yourself, email the zookeeper at scott at scottaaronson.com.

Done

All Classes

Complexity classes by letter: Symbols - A - B - C - D - E - F - G - H - I - J - K - L - M - N - O - P - Q - R - S - T - U - V - W - X - Y - Z

Lists of related classes: Communication Complexity - Hierarchies - Nonuniform

Symbols

0-1-NP_C - 1NAuxPDA^p - 3SUM-hard - #AC^0 - #L - #L/poly - #GA - #P - #W[t] - ⊕EXP - ⊕L - ⊕L/poly - ⊕P - ⊕SAC^0 - ⊕SAC^1

A

A_0PP - AC - AC^0 - $AC^0[m]$ - AC^1 - ACC^0 - AH - AL - ALL - ALOGTIME - AlgP/poly - Almost-NP - Almost-P - Almost-PSPACE - AM - AM_{EXP} - AM ∩ coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - APP - APX - ATIME - AUC-SPACE(f(n)) - AuxPDA - AVBPP - AvgE - AvgP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - AxP - AxPP

B

βP - BH - $BP_d(P)$ - BPE - BPEE - $BP_HSPACE(f(n))$ - BPL - BP•NP - BPP - BPP^{cc} - BPP_k^{cc} - BPP^{KT} - BPP/log - BPP/mlog - BPP//log - BPP/rlog - BPP-OBDD - BPP_{path} - BPQP - BPSPACE(f(n)) - BPTIME(f(n)) - BQNC - BQNP - BQP - BQP/log - BQP/poly - BQP/mlog - BQP/mpoly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQPSPACE - BQP_{CTC} - BQP_{tt}/poly - BQTIME(f(n)) - k-BWBP

C

$C_=AC^0$ - $C_=L$ - $C_=P$ - CC - CC^0 - CFL - CLOG - CH - Check - CL#P - C_kP - CNP - coAM - $coC_=P$ - cofrIP - Coh - coMA - $coMod_kP$ - compIP - compNP - coNE - coNEXP - coNL - coNP - $coNP^{cc}$ - coNP/poly - coNQP - coRE - coRNC - coRP - coSL - coSPARSE - coUCC - coUP - CP - CSIZE(f(n)) - CSL - CSP - CZK

D

D#P - DCFL - Δ_2P - δ-BPP - δ-RP - DET - DiffAC^0 - DisNP - DistNP - DP - DQP - DSPACE(f(n)) - DTIME(f(n)) - DTISP(t(n),s(n)) - Dyn-FO - Dyn-ThC^0

E

E - EE - EEE - EESPACE - EEXP - EH - ELEMENTARY - EL$_k$P - EP - EPTAS - k-EQBP - EQP - EQP$_K$ - EQTIME(f(n)) - ESPACE - ∃BPP - ∃NISZK - EXP - EXP/poly - EXPSPACE

F

FBQP - Few - FewEXP - FewP - FH - FIXP - FNL - FNL/poly - FNP - FO(t(n)) - FOLL - FP - FP$^{NP[log]}$ - FPR - FPRAS - FPT - FPT$_{nu}$ - FPT$_{su}$ - FPTAS - FQMA - frIP - F-TAPE(f(n)) - F-TIME(f(n))

G

GA - GAN-SPACE(f(n)) - GapAC^0 - GapL - GapP - GC(s(n),C) - GCSL - GI - GLO - GPCD(r(n),q(n)) - G[t]

H

HalfP - HeurBPP - HeurBPTIME(f(n)) - HeurDTIME$_\delta$(f(n)) - HeurP - HeurPP - HeurNTIME$_\delta$(f(n)) - H$_k$P - HVSZK

I

IC[log,poly] - IP - IPP - IP[polylog]

L

L - LIN - L$_k$P - LOGCFL - LogFew - LogFewNL - LOGNP - LOGSNP - L/poly - LWPP

M

MA - MA' - MAC^0 - MA$_E$ - MA$_{EXP}$ - mAL - MA$_{POLYLOG}$ - MaxNP - MaxPB - MaxSNP - MaxSNP$_0$ - mcoNL - MinPB - MIP -

M

MA - MA' - MAC^0 - MA_E - MA_{EXP} - mAL - $MA_{POLYLOG}$ - MaxNP - MaxPB - MaxSNP - $MaxSNP_0$ - mcoNL - MinPB - MIP - MIP*[2,1] - MIP_{EXP} - $(M_k)P$ - mL - MM - MMSNP - mNC^1 - mNL - mNP - Mod_kL - ModL - Mod_kP - ModP - $ModZ_kL$ - mP - MP - MPC - mP/poly - mTC^0

N

$NAuxPDA^p$ - NC - NC^0 - NC^1 - NC^2 - NE - NE/poly - Nearly-P - NEE - NEEE - NEEXP - NEXP - NEXP/poly - NIPZK - NIQSZK - NISZK - $NISZK_h$ - NL - NL/poly - NLIN - NLOG - NONE - NP - NPC - NP_C - NP^{cc} - $NP_{\mathbb{R}}^{cc}$ - NPI - NP ∩ coNP - (NP ∩ coNP)/poly - NP/log - NPMV - NPMV-sel - $NPMV_t$ - $NPMV_t$-sel - NPO - NPOPB - NP/poly - (NP,P-samplable) - NP_R - NPSPACE - NPSV - NPSV-sel - $NPSV_t$ - $NPSV_t$-sel - NQP - NSPACE(f(n)) - NT - NT* - NTIME(f(n))

O

OCQ - OptP

P

P - P/log - P/poly - $P^{\#P}$ - $P^{\#P[1]}$ - P_{CTC} - PAC^0 - PBP - k-PBP - P_C - P^{cc} - $P_{\mathbb{R}}^{cc}$ - PCD(r(n),q(n)) - P-Close - PCP(r(n),q(n)) - PermUP - PEXP - PF - PFCHK(t(n)) - PH - PH^{cc} - Φ_2P - PhP - Π_2P - PINC - PIO - P^K - PKC - PL - PL_1 - PL_∞ - PLF - PLL - PLS - P^{NP} - $P^{\|NP}$ - $P^{NP[k]}$ - $P^{NP[log]}$ - $P^{NP[log^2]}$ - P-OBDD - PODN - polyL - PostBQP - PP - PP^{cc} - PP/poly - PPA - PPAD - PPADS - P^{PP} - PPP - PPSPACE - PQUERY - PR - P_R - $Pr_HSPACE(f(n))$ - PromiseBPP - PromiseBQP - PromiseP - PromiseRP - PrSPACE(f(n)) - P-Sel - PSK - PSPACE - PSPACE/poly - PT_1 - PTAPE - PTAS - PT/WK(f(n),g(n)) - PZK

Q

Q - QAC^0 - $QAC^0[m]$ - $QACC^0$ - QAC_f^0 - QAM - QCFL - QCMA - QH - QIP - QIP[2] - QMA - QMA-plus - QMA(2) - QMA_1 - QMA_{log} - QMAM - QMA/qpoly - QMIP - $QMIP_{le}$ - $QMIP_{ne}$ - QNC - QNC^0 - QNC_f^0 - QNC^1 - QP - QPLIN - QPSPACE - QRG - QS_2P - QSZK

R - RBQP - RE - REG - RevSPACE(f(n)) - RG - RG[1] - R_HL - R_HSPACE(f(n)) - RL - RNC - RP - RP_k^{cc} - RPP - RQP - RSPACE(f(n))

S

S_2P - S_2-EXP•P^{NP} - SAC - SAC^0 - SAC^1 - SAPTIME - SBP - SC - SE - SEH - SelfNP - SF_k - Σ_2P - SKC - SL - SLICEWISE PSPACE - SNP - SO-E - SP - span-P - SPARSE - SPL - SPP - SQG - SUBEXP - symP - SZK - SZK_h

T

TALLY - TC^0 - TFNP - Θ_2P - TreeBQP - TREE-REGULAR

U

UAP - UCC - UE - UL - UL/poly - UP - UPP^{cc} - US

V

VC_k - VC_{OR} - VNC_k - VNP_k - VP_k - VPL - VQP_k

W

W[1] - WAPP - W[P] - WPP - W[SAT] - W[*] - W[t] - W^*[t]

X

XOR-MIP*[2,1] - XP - $XP_{uniform}$

Y

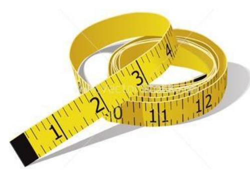
YACC - YP - YPP - YQP

Z

ZBQP - ZPE - ZPP - ZPTIME(f(n)) - ZQP

The "Complexity Zoo" Class inclusion diagram

- Currently 493 named classes!
- Interactive, clickable map
- Shows class subset relations



Legend:

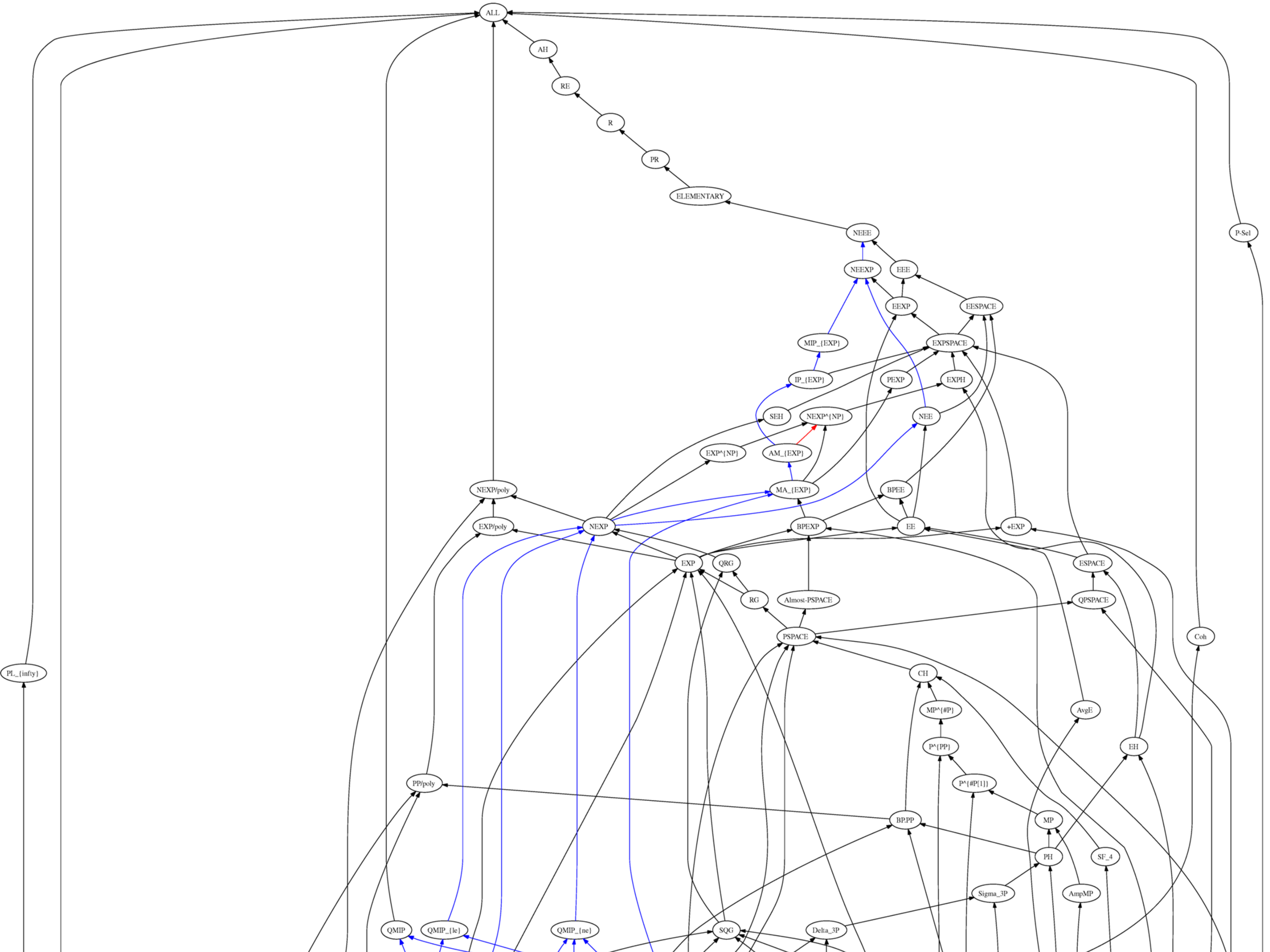
Arrows: ⟶ $\forall X: A^X \subseteq B^X$ and $co.A^X \subseteq B^X$ ⟶ $\forall X: A^X \subseteq B^X$ ⟶ $\forall X: co.A^X \subseteq B^X$ ⟶ $\forall X: cocap.A^X \subseteq B^X$

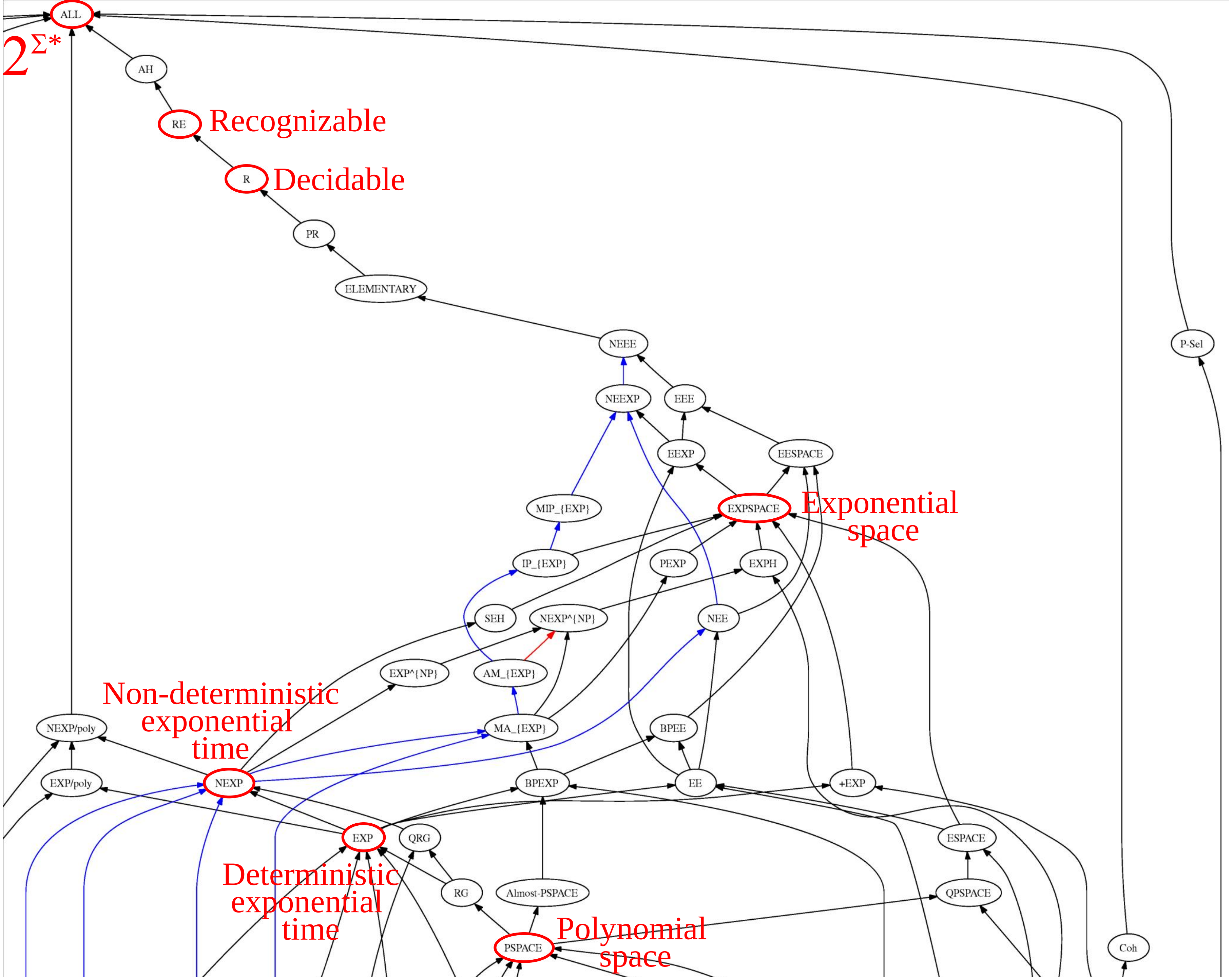
Node colors: $\forall X$: $A^X \subseteq^? B^X$ proven disproven open unknown to us

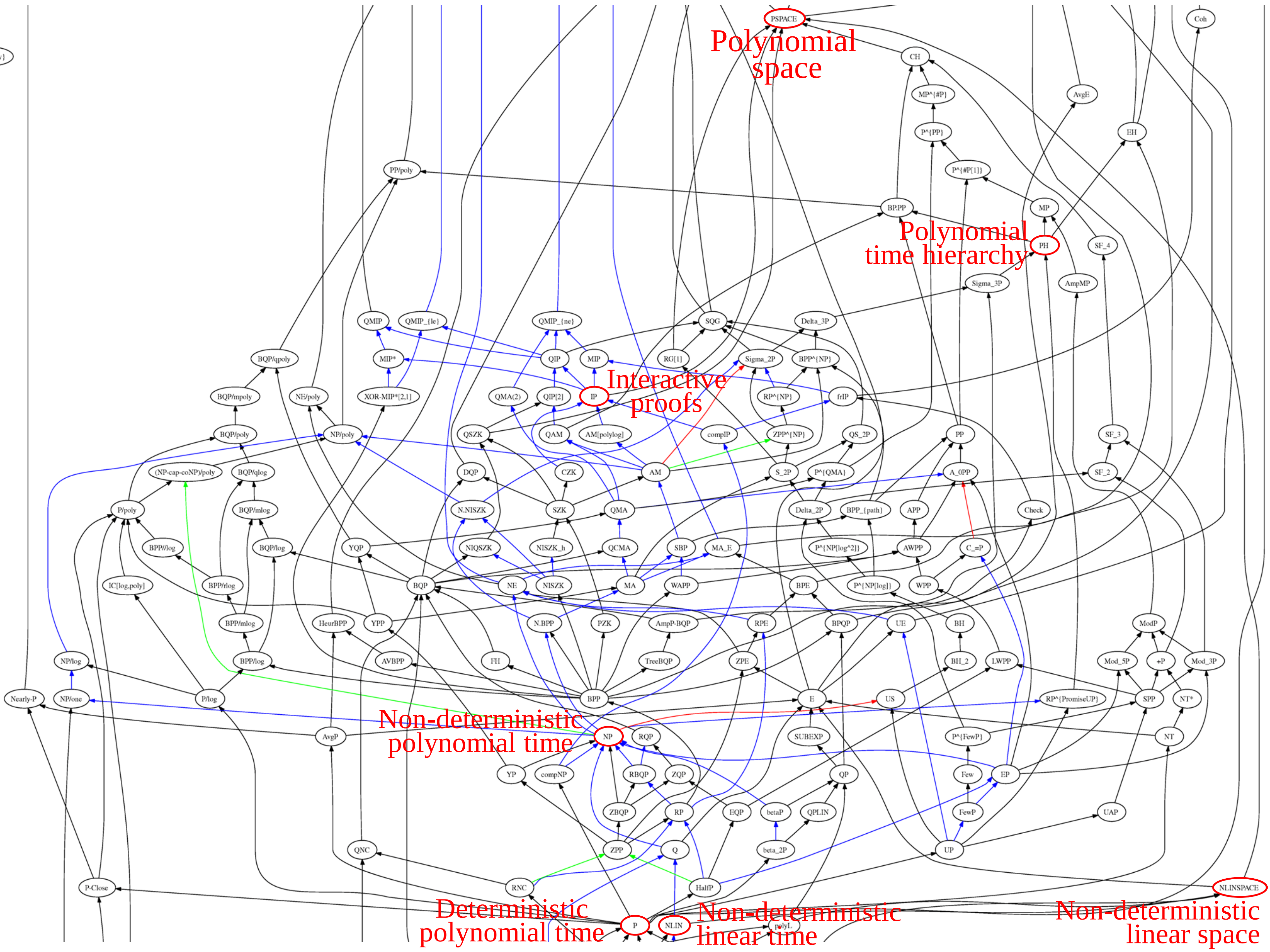
If a cell has more than one color: left - regular inclusion; right - twisted inclusion; middle - weak inclusion.
Click on a node to select it as the "A" class. Press backslash (\) to switch between subset and superset status.
Shift-click to open to the "Class Relations" entry in a separate page.
See also: Complexity Zoology Introduction, Static Inclusion Diagram, Complexity Class Relations

http://www.math.ucdavis.edu/~greg/zoology/diagram.xml

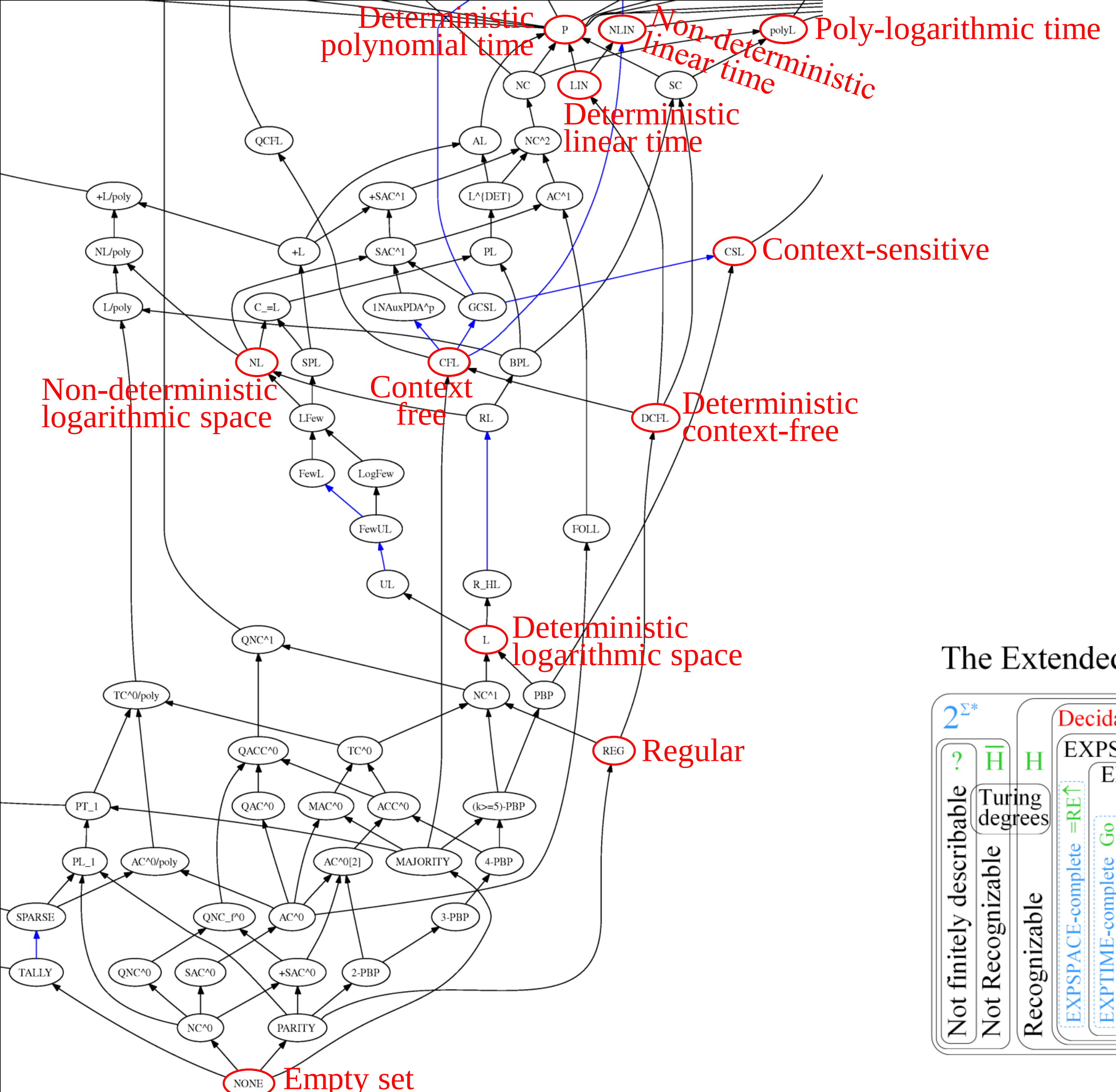
Scott Aaronson


ALL
AH
RE
R
PR
ELEMENTARY
NEEE
P-Sel
NEEXP
EEE
EEXP
EESPACE
MIP_{EXP}
EXPSPACE
IP_{EXP}
PEXP
EXPH
SEH
NEXP^{NP}
NEE
EXP^{NP}
AM_{EXP}
NEXP/poly
MA_{EXP}
BPEE
EXP/poly
NEXP
BPEXP
EE
+EXP
EXP
QRG
ESPACE
RG
Almost-PSPACE
QPSPACE
PSPACE
Coh
PL_{infty}
CH
MP^{#P}
AvgE
P^{PP}
EH
PP/poly
P^{#P[1]}
BP.PP
MP
PH
SF_4
Sigma_3P
AmpMP
QMIP
QMIP_{le}
QMIP_{ne}
SQG
Delta_3P

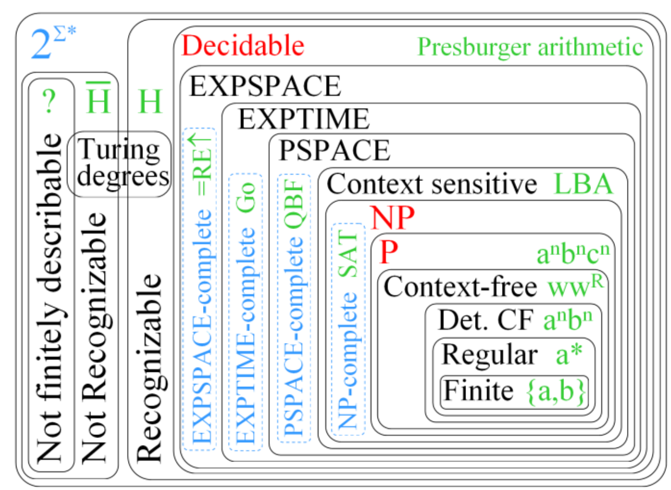

ALL
2^Σ*
AH
RE
Recognizable
R
Decidable
PR
ELEMENTARY
NEEE
P-Sel
NEEXP
EEE
EEXP
EESPACE
MIP_{EXP}
EXPSPACE
Exponential space
IP_{EXP}
PEXP
EXPH
SEH
NEXP^{NP}
NEE
EXP^{NP}
AM_{EXP}
Non-deterministic exponential time
NEXP/poly
MA_{EXP}
BPEE
EXP/poly
NEXP
BPEXP
EE
+EXP
EXP
QRG
ESPACE
Deterministic exponential time
RG
Almost-PSPACE
QPSPACE
PSPACE
Polynomial space
Coh



Polynomial space
Polynomial time hierarchy
Interactive proofs
Non-deterministic polynomial time
Deterministic polynomial time
Non-deterministic linear time
Non-deterministic linear space
PSPACE
Coh
CH
MP^{#P}
AvgE
P^{PP}
EH
PP/poly
P^{#P[1]}
BP.PP
MP
PH
SF_4
Sigma_3P
AmpMP
QMIP
QMIP_{le}
QMIP_{ne}
SQG
Delta_3P
BQP/qpoly
MIP*
QIP
MIP
RG[1]
Sigma_2P
BPP^{NP}
BQP/mpoly
NE/poly
XOR-MIP*[2,1]
QMA(2)
QIP[2]
IP
RP^{NP}
frIP
BQP/poly
NP/poly
QSZK
QAM
AM[polylog]
compIP
ZPP^{NP}
QS_2P
PP
SF_3
(NP-cap-coNP)/poly
BQP/qlog
DQP
CZK
AM
S_2P
P^{QMA}
A_0PP
SF_2
P/poly
BQP/mlog
N.NISZK
SZK
QMA
Delta_2P
BPP_{path}
APP
Check
BPP//log
BQP/log
YQP
NIQSZK
NISZK_h
QCMA
SBP
MA_E
P^{NP[log^2]}
AWPP
C_=P
IC[log,poly]
BPP/rlog
BQP
NE
NISZK
MA
WAPP
BPE
P^{NP[log]}
WPP
BPP/mlog
HeurBPP
YPP
N.BPP
PZK
AmpP-BQP
RPE
BPQP
UE
BH
ModP
NP/log
AVBPP
FH
BPP/log
TreeBQP
ZPE
BH_2
LWPP
Mod_5P
+P
Mod_3P
Nearly-P
NP/one
P/log
BPP
E
US
RP^{PromiseUP}
SPP
NT*
AvgP
NP
RQP
SUBEXP
P^{FewP}
NT
YP
compNP
RBQP
ZQP
QP
Few
EP
ZBQP
RP
EQP
betaP
QPLIN
FewP
UAP
QNC
ZPP
Q
beta_2P
UP
P-Close
RNC
HalfP
NLINSPACE
P
NLIN
polyL



The Extended Chomsky Hierarchy



The Extended Chomsky Hierarchy Reloaded