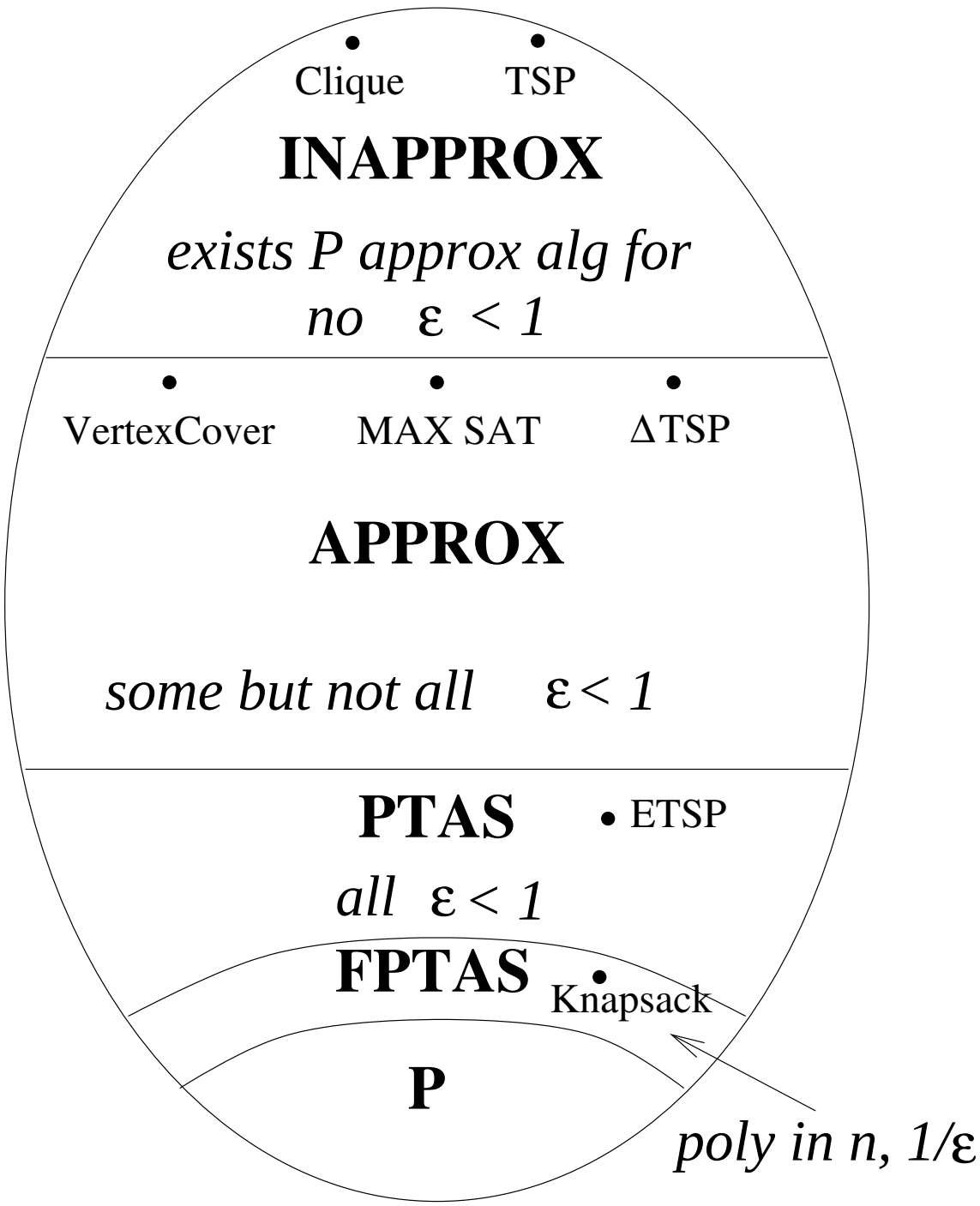
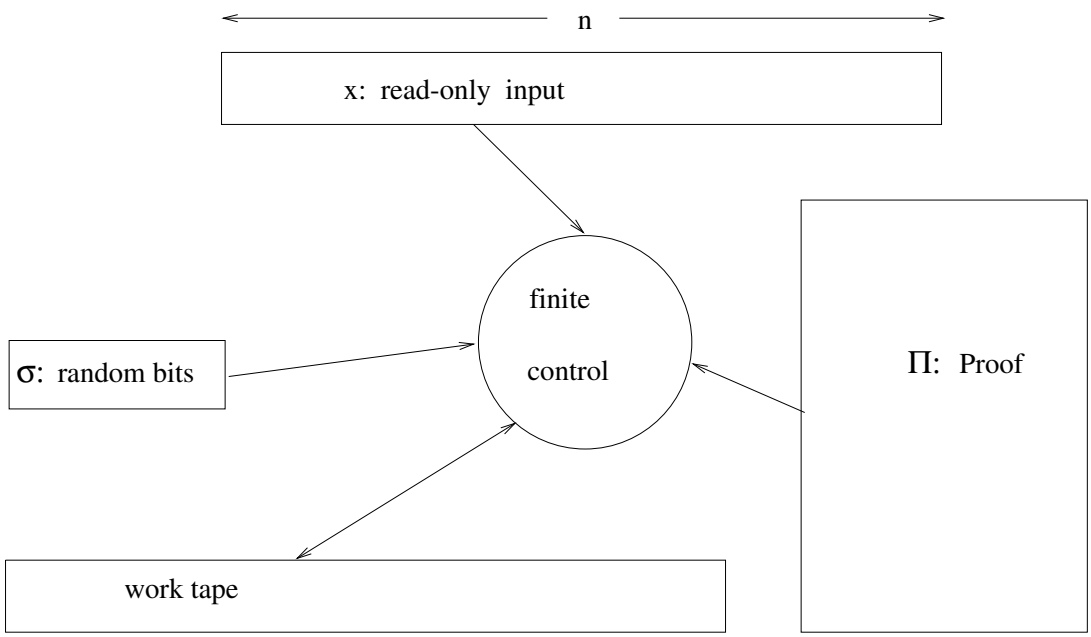

Clique
TSP
INAPPROX
exists P approx alg for
no ε < 1
VertexCover
MAX SAT
ΔTSP
APPROX
some but not all ε < 1
PTAS
ETSP
all ε < 1
FPTAS
Knapsack
P
poly in n, 1/ε



Merlin-Arthur games (MA) [Babai]

Decision problem: D; input string: x

Merlin — Prover — chooses the polynomial-length string Π that **Maximizes** the chances of convincing Arthur that x is an element of D.

Arthur — Verifier — "computes" the **Average** value of his possible computations on Π, x. Arthur is a polynomial-time, probabilistic Turing machine.

Definition 22.1 We say that $\mathbf{A}$ *accepts* D iff the following conditions hold:

1. If $x \in D$, there exists a proof Π_x, such that $\mathbf{A}$ accepts for every random string σ,

$$Pr_\sigma \left[\mathbf{A}^{\Pi_x}(x, \sigma) = Accept\right] = 1$$

2. If $x \notin D$, for every proof Π, $\mathbf{A}$ rejects for most of the random strings σ,

$$Pr_\sigma \left[\mathbf{A}^{\Pi}(x, \sigma) = Accept\right] < \frac{1}{2}$$

♠

Any decision problem $D \in$ **NP** has a deterministic, polynomial-time verifier satisfying Definition 22.1.

By adding randomness to the verifier, we can greatly restrict its computational power and the number of bits of Π that it needs to look at, while still enabling it to accept all of **NP**.

We say that a verifier **A** is $(r(n), q(n))$-*restricted* iff for all inputs of size n, and all proofs Π, **A** uses at most $O(r(n))$ random bits and examines at most $O(q(n))$ bits of its proof, Π.

Let PCP$(r(n), q(n))$ be the set of boolean queries that are accepted by $(r(n), q(n))$-restricted verifiers.

Fact 22.2 (PCP Theorem) **NP** $=$ PCP$[\log n, 1]$

The proof of this theorem is pretty messy, certainly more than we can deal with here. But we can look at the applications of the PCP Theorem to approximation problems.

MAX-3-SAT: given a 3CNF formula, find a truth assignment that maximizes the number of true clauses.

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$
$$\wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5)$$

Proposition 22.3 MAX-3-SAT *has a polynomial-time* $\epsilon = \frac{1}{2}$ *approximation algorithm.*

Proof: Be greedy, set each variable in turn to the better value. ♠

You can do better – a random assignment gets 7/8 of the clauses.

Open for Years: Assuming **NP** $\neq$ **P** is there some ϵ, $0 < \epsilon < 1$ s.t. MAX-3-SAT has no PTIME ϵ-approximation algorithm?

Theorem 22.4 *The PCP theorem (***NP** $=$ PCP$[\log n, 1]$*)*
is equivalent to the fact that
If **P** $\neq$ **NP**, *then*
For some ϵ, $1 > \epsilon > 0$,
$\mathrm{MAX\text{-}3\text{-}SAT}$ *has no polynomial-time,* ϵ*-approximation algorithm.*

Fact 22.5 $\mathrm{MAX\text{-}3\text{-}SAT}$ *has a PTIME approximation algorithm with* $\epsilon = \frac{1}{8}$ *and no better ratio can be achieved unless* **P** $=$ **NP**.

References:

- *Approximation Algorithms for NP Hard Problems*, Dorit Hochbaum, ed., PWS, 1997.
- Juraj Hromkovic, *Algorithmics for Hard Problems*, Springer, 2001.
- Sanjeev Arora, "The Approximability of NP-hard Problems", STOC 98, www.cs.princeton.edu/∼arora.

The concept of a nondeterministic acceptor of a boolean query has a long and rich history, going back to various kinds of nondeterministic automata.

It is important to remember that these are fictitious machines: we suspect that they cannot be built.

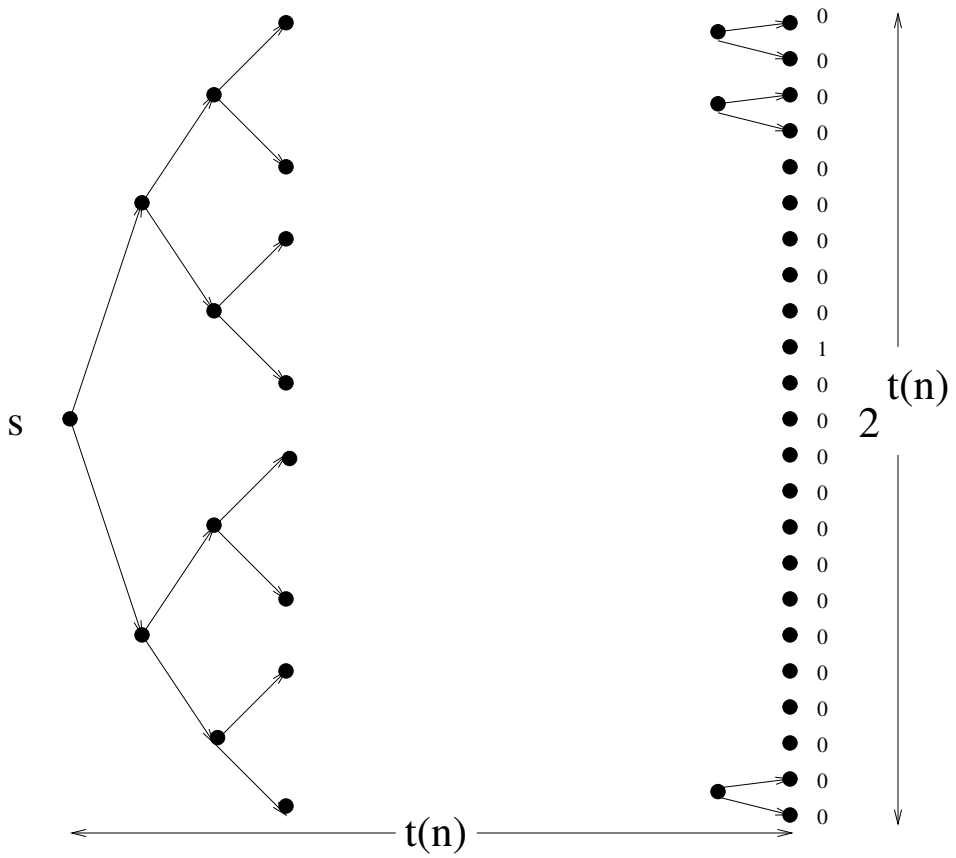
For $A \subseteq \text{STRUC}_{\text{fin}}[\tau]$ a boolean query, define its complement $\overline{A} = \text{STRUC}_{\text{fin}}[\tau] - A$. Given a complexity class, $\mathcal{C}$, define the complementary class,

$$\text{co-}\mathcal{C} \quad = \quad \{\overline{A} \mid A \in \mathcal{C}\}$$

Example: $\text{SAT} \in$ **NP**; $\overline{\text{SAT}} = \text{UNSAT} \in$ co-**NP**

Open question: **NP** $?=$ co-**NP**

If one could really build an **NP** machine, then one could with a single gate to invert its answer also build a co-**NP** machine. From a very practical point of view, the complexity of a problem A and its complement, $\overline{A}$ are identical.



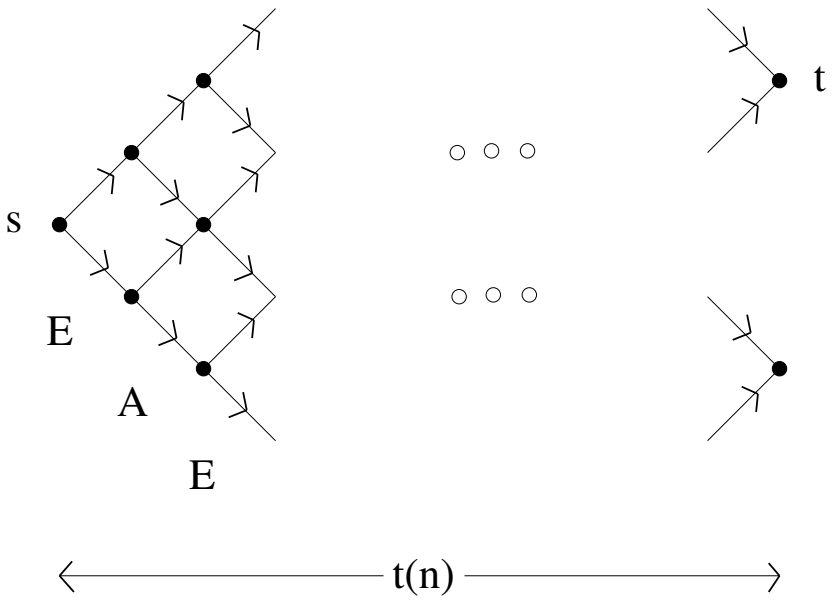
$$\text{Value}(\text{ID}) \quad \equiv \quad \text{Value(LeftChild(ID))} \vee \text{Value(Right(Child(ID)}$$

weak communication pattern;
wasteful use of all these processors

Definition 22.6 Define an *alternating Turing machine* to be a Turing machine whose states are divided into two groups: the *existential* states and the *universal* states.

Recall that an *instantaneous description* (ID) consists of the machine's state, work-tape contents, and head positions. The notion of when such a machine accepts an input is defined by induction: The alternating Turing machine in ID_0 *accepts* iff

1. ID_0 is in a final accepting state, or
2. ID_0 is in an existential state and there exists a next ID that accepts, or
3. ID_0 is in a universal state, there is at least one next ID, and all next ID's accept.



What does it mean for a string to be in the language of an alternating TM? We sometimes misleadingly say "the ATM accepts x" just as we might say "the NDTM accepts x". But this isn't really sensible, because we should not talk about what an NDTM or ATM *will* do, only what it might do.

Language of an NDTM: $x \in \mathcal{L}(N)$ iff N might accept x.

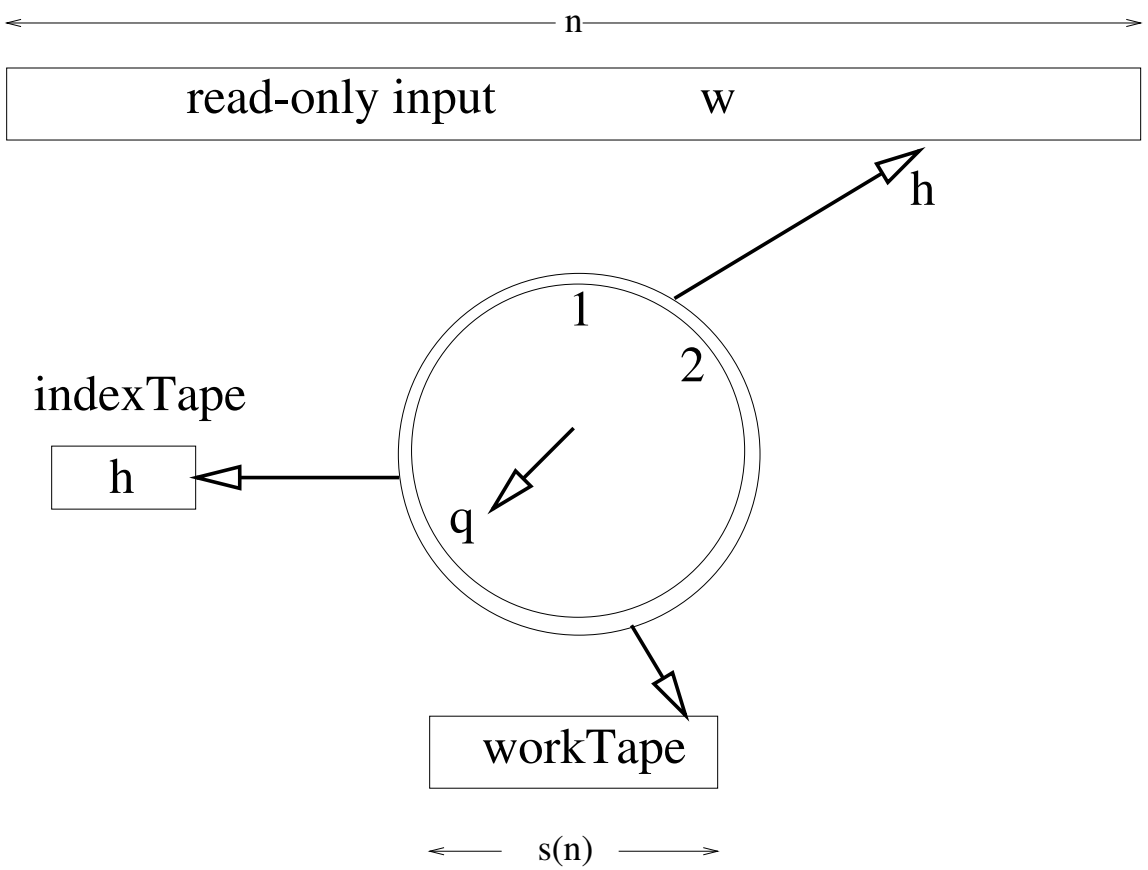
Equivalently: $x \in \mathcal{L}(N)$ if a smart player, who controls N's choices and wants N to accept, can make N accept.

Game Semantics of an ATM: There are two players, White who controls M in the existential states and wants M to accept, and Black who controls M in the universal states and wants M to reject. $x \in \mathcal{L}(M)$ iff White can force M to accept x given optimal play by each player.

This is *equivalent* to the definition of "acceptance" given on the last slide, but I find it much easier to understand.

Index Tapes

From now on assume that our Turing machines have a *random access* read-only input. There is an *index tape* which can be written on and read like other tapes. Whenever the value h, written in binary, appears on the index tape, the read head will automatically scan bit h bit of the input.



We can now define alternating complexity classes as follows.

ASPACE$[s(n)]$ is the set of boolean queries (languages) accepted by alternating Turing machines that use $O(s(n))$ tape cells on any computation path when the input size is n.

Similarly **ATIME**$[t(n)]$ is the set of boolean queries (languages) accepted by alternating TM's that take $O(t(n))$ time steps on any computation path when the input size is n.

Theorem 22.7 *For $s(n) \geq \log n$, and for $t(n) \geq n$,*

$$\bigcup_{k=1}^{\infty} \textbf{ATIME}[(t(n))^k] \;=\; \bigcup_{k=1}^{\infty} \textbf{DSPACE}[(t(n))^k]$$

$$\textbf{ASPACE}[s(n)] \;=\; \bigcup_{k=1}^{\infty} \textbf{DTIME}[k^{s(n)}]$$

Corollary 22.8 *In particular,*

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

The Alternation Theorem has four parts.

We must simulate:

1. A time-bounded DTM with a space-bounded ATM, using *the circuit game*,
2. A space-bounded DTM with a time-bounded ATM, using *the Savitch game*,
3. A space-bounded ATM with a time-bounded DTM, using the *marking algorithm*, and
4. A time-bounded ATM with a space-bounded DTM, using *exhaustive search of the game tree*.

First, though, let's see some examples of alternating machines. I'll include both "classical" and game-semantics descriptions of each machine.

Define the monotone, circuit value problem (MCVP) to be the subset of CVP in which no negation gates occur. It turns out that MCVP is still P-complete.

Proposition 22.9 MCVP *is recognizable in* **ASPACE**$[\log n]$.

Proof: Let G be a monotone boolean circuit. For $a \in V^G$, define "EVAL(a)",

1. **if** (InputOn(a)) **then accept**
2. **if** (InputOff(a)) **then reject**
3. **if** ($G_\wedge(a)$) **then** universally choose child b of a
4. **if** ($G_\vee(a)$) **then** existentially choose child b of a
5. Return(EVAL(b))

M simply calls EVAL(r). EVAL(a) returns "**accept** " iff gate a evaluates to one.

Space used for naming vertices a, b: $O(\log n)$. ♠

Game Semantics for this Proof:

In the *Circuit Game*, we start with a marker at the output gate of a monotone Boolean circuit G. On each move, the marker moves from the current gate to one of its children (one of its inputs). For AND gates Black chooses which child to move to, and for OR gates White chooses. When we reach an input gate marked with a literal, White wins the game iff this literal evaluates to true.

White wants to stay on gates that evaluate to true, Black on gates that evaluate to false. Whoever is happy with the output gate can always move the marker to stay happy, until an input is reached.

Definition 22.10 The *quantified satisfiability problem* (QSAT) is the set of true formulas of the following form:

$$\Psi \quad = \quad (Q_1 x_1)(Q_2 x_2) \cdots (Q_r x_r)\varphi$$

♠

For any boolean formula φ on variables $\overline{x}$,

$$\varphi \in \text{SAT} \quad \Leftrightarrow \; (\exists \overline{x})\varphi \in \text{QSAT}$$

$$\varphi \notin \text{SAT} \quad \Leftrightarrow \; (\forall \overline{x})\neg\varphi \in \text{QSAT}$$

Thus QSAT logically contains SAT and $\overline{\text{SAT}}$.

Proposition 22.11 QSAT *is recognizable in* **ATIME***[n]*.

Proof: Construct an alternating machine A as follows. To evaluate the sentence,

$$\Phi \equiv (\exists x_1)(\forall x_2) \cdots (Q_r x_r)\alpha(\overline{x}) \,,$$

in an existential state, A writes down a boolean value for x_1, in a universal state it writes a bit for x_2, and so on.

Next A must evaluate the quantifier-free boolean formula α on these chosen values. For each "$\wedge$" in α, A universally chooses which side to evaluate; for each "$\vee$", A existentially chooses.

Thus A only has to read one of the chosen bits, x_i, and accept iff it is true and occurs positively, or false and occurs negatively.

A runs in linear time.

A accepts Φ iff Φ is true. ♠

Game Semantics: White picks values for the $\exists$ variables, Black for the $\forall$ variables. White wins iff the quantifier-free part evaluates to true.

Theorem 22.12 *For any* $s(n) \geq \log n$*,*

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

Proof: NSPACE$[s(n)] \subseteq$ **ATIME**$[s(n)^2]$:

Let N be an **NSPACE**$[s(n)]$ Turing machine.

Let w be an input to N, $n = |w|$.

$$w \in \mathcal{L}(N) \qquad \Leftrightarrow \qquad \text{CompGraph}(N, w) \in \text{REACH}$$

$P(d, x, y)$ accepts iff there is a path in CompGraph(N, w) of length at most 2^d from x to y.

$$P(d, x, y) \;\equiv\; (\exists z)(P(d-1, x, z) \;\wedge\; P(d-1, z, y))$$

1. Existentially: choose middle ID z.
2. Universally: $(x, y) := (x, z)$ **AND** (z, y)
3. Return($P(d-1, x, y)$)

$$\begin{aligned} T(d) &= O(s(n)) + T(d-1) \;=\; O(d \cdot s(n)) \\ d &= O(s(n)) \\ T(d) &= O((s(n))^2) \end{aligned}$$

Game Semantics for this version of Savitch's Theorem:

White originally *claims* that the path from s to t, of length at most $2^{s(n)}$, exists in CompGraph(N, w). White *advances* this claim by naming the alleged middle vertex of the alleged path. Black picks either the *first half* or the *second half* of the path to *challenge*. The path whose existence is disputed is then half the length. They continue, halving the length of the disputed path each time, until they dispute the existence of an *edge*. At this point the player with the correct claim about this edge wins.

If White's path actually exists, White has a winning strategy that consists of always telling the truth. If White is wrong, however, either the first or second half of the alleged path must fail to exist, and Black has a winning move.

ATIME$[t(n)] \subseteq$ **DSPACE**$[t(n)]$:

Let A be an **ATIME**$[t(n)]$ machine, input w, $n = |w|$.

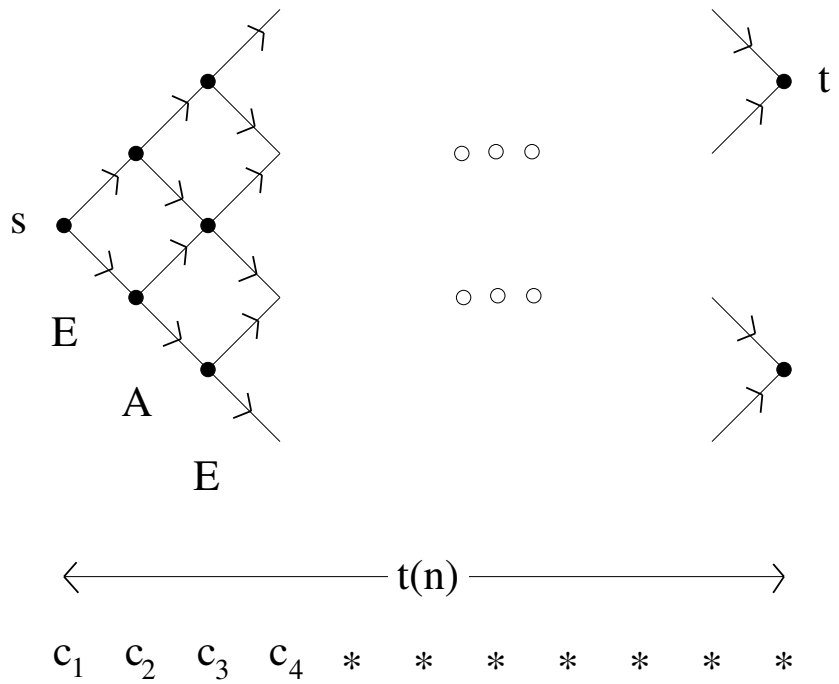
CompGraph(A, w) has depth $O(t(n))$ and size $2^{O(t(n))}$.

A deterministic Turing machine can systematically search this entire and-or graph using space $O(t(n))$. This is done, by keeping a string of length $O(t(n))$: $c_1 c_2 \ldots c_r \star \ldots \star$ denoting that we are currently simulating step r of A's computation having made choices $c_1 \ldots c_r$ on all of the existential and universal branches up until now. The rest of the simulation will report an **answer** as to whether choices $c_1 \ldots c_r$ will lead to acceptance. This is done as follows:

If one of the following conditions holds:

1. $c_r = 1$, or
2. **answer** = "yes" and step r was existential, or
3. **answer** = "no" and step r was universal,

then let $c_r = \star$ and report **answer** back to the $r - 1^{\text{st}}$ step. Otherwise, set $c_r = 1$ and continue. Note, that we do not have to store intermediate IDs of the simulation because the sequence $c_1 c_2 \ldots c_r \star \ldots \star$ uniquely determines which ID of A we go to next. ♠



$$\textbf{ATIME}[t(n)] \quad \subseteq \quad \textbf{DSPACE}[t(n)]$$

We evaluate the computation graph of **ATIME**$[t(n)]$ machine using $t(n)$ space to cycle through all possible computations of A on input w.

The **game tree** for the machine represents all possible configurations and moves of the players. In effect our algorithm exhaustively searches every possible path through this tree, using space $t(n)$ to keep track of where in the search it might be. (One could do this with a very time-inefficient recursive algorithm.)

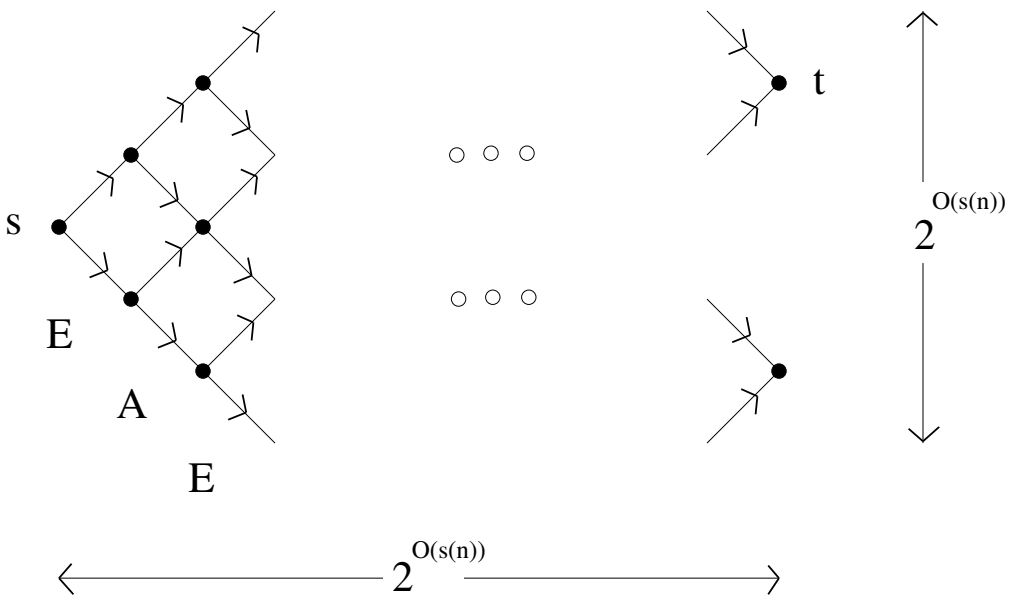
Theorem 22.13 **ASPACE**$[s(n)]$ $=$ **DTIME**$[2^{O(s(n))}]$

Proof: ASPACE$[s(n)] \subseteq$ **DTIME**$[2^{O(s(n))}]$:

Let A be an **ASPACE**$[s(n)]$ machine. Let w be an input, $n = |w|$.

CompGraph$(A(w))$ has size $\leq 2^{O(s(n))}$

The *marking algorithm* evaluates this in **DTIME**$[2^{O(s(n))}]$.



We mark the winner at each point, starting from the end configurations and working backward.

DTIME$[2^{O(s(n))}] \subseteq$ **ASPACE**$[s(n)]$:

Let M be a **DTIME**$[2^{k(s(n))}]$ TM.

w an input, $n = |w|$.

Define an alternating procedure $C(t, p, a)$ which accepts iff the contents of cell p at time t in M's computation on input w is symbol a.

Inductively, $C(t{+}1, p, b)$ holds iff the three symbols a_{-1}, a_0, a_1 in tape positions $p-1, p, p+1$ lead to a "b" in position p in one step of M's computation.

$$(a_{-1}, a_0, a_1) \stackrel{M}{\rightarrow} b$$

$$C(t+1, p, b) \;\equiv\; (\exists a_{-1}, a_0, a_1)(\; (a_{-1}, a_0, a_1) \stackrel{M}{\rightarrow} b \quad \wedge$$

$$(\forall i \in \{-1, 0, 1\})(C(t, p+i, a_i)))$$

Space needed is $O(\log 2^{k(s(n))})$ = $O(s(n))$.

Note that M accepts w iff $C(2^{k(s(n))}, 1, \langle q_f, 1\rangle)$

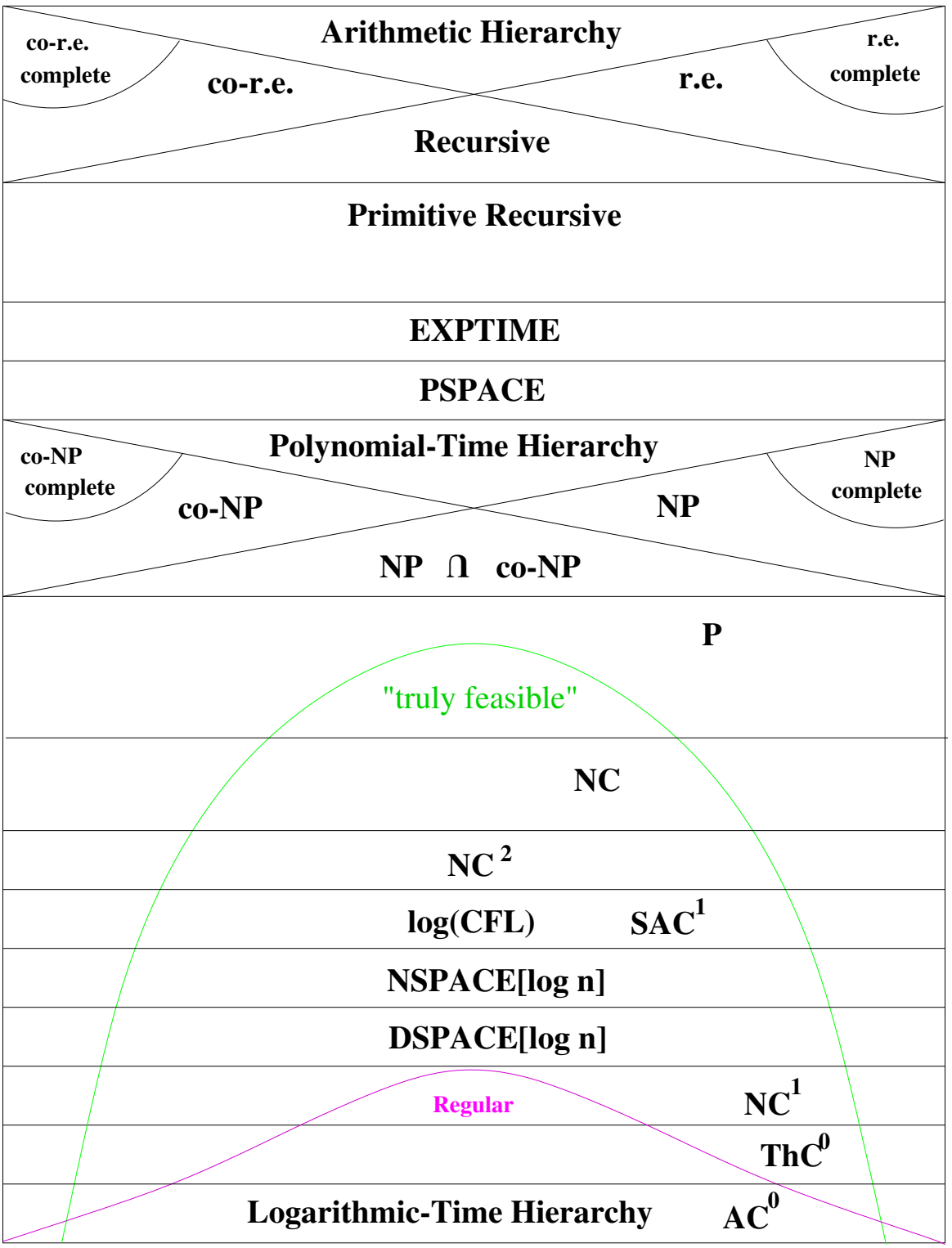
	Space						
	1	2	p	n			$T(n)$
Time 0	$\langle q_0, w_1 \rangle$	w_2	$\cdots$	w_n	$\sqcup$	$\cdots$	$\sqcup$
1	w_1	$\langle q_1, w_2 \rangle$	$\cdots$	w_n	$\sqcup$	$\cdots$	$\sqcup$
	$\vdots$	$\vdots$	$\vdots$			$\vdots$	
t			a_{-1} a_0 a_1				
$t+1$			b				
	$\vdots$	$\vdots$	$\vdots$			$\vdots$	
$T(n)$	$\langle q_f, 1 \rangle$	$\cdots$	$\cdots$			$\cdots$	

$$C(t+1,p,b) \equiv (\exists a_{-1}, a_0, a_1)(\ (a_{-1}, a_0, a_1) \xrightarrow{M} b \quad \wedge$$

$$(\forall i \in \{-1,0,1\})(C(t, p+i, a_i)))$$

Game Semantics: At each point in the game the contents of a single cell of the tableau are in dispute. Originally this is the bottom-left cell, which White claims is accepting. At each move White names the three cells above the current one, which must imply the claimed contents of the current one. Black then picks one of these three to dispute. Note the similarity to the Circuit Game.

This completes the proof of Theorem 22.7. ♠


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