$$\text{depth} \quad = \quad \text{parallel time}$$

$$\text{width} \; = \; \text{hardware}$$

$$\text{number of gates} \; = \; \text{computational work} \; = \; \text{sequential time}$$

Theorem: For all i, $\quad \mathbf{CRAM}[(\log n)^i] \quad = \quad \mathbf{AC}^i$

$$\begin{array}{ccccccccccccc} \mathbf{AC}^0 & \subseteq & \mathbf{ThC}^0 & \subseteq & \mathbf{NC}^1 & \subseteq & \mathbf{L} & \subseteq & \mathbf{NL} & \subseteq & \mathbf{sAC}^1 & & \subseteq \\ \mathbf{AC}^1 & \subseteq & \mathbf{ThC}^1 & \subseteq & \mathbf{NC}^2 & & & & & \subseteq & \mathbf{sAC}^2 & \subseteq & \cdots \end{array}$$

$$\bigcup_{i=0}^{\infty} \mathbf{NC}^i \; = \; \bigcup_{i=0}^{\infty} \mathbf{AC}^i \; = \; \bigcup_{i=0}^{\infty} \mathbf{ThC}^i \; = \; \mathbf{NC} \; \subseteq \; \mathbf{P}$$

$$\begin{aligned} \mathbf{NC}[t(n)] \; &= \; \text{ParallelTime}[t(n)] \text{ on real hardware} \\ \mathbf{AC}[t(n)] \; &= \; \mathbf{CRAM}[t(n)] \\ \mathbf{ThC}[t(n)] \; &= \; \text{ParallelTime}[t(n)] \text{ on ``wet-ware''} \\ \mathbf{sAC}^i \; &= \; \mathbf{AC}^i \text{ but } \wedge\text{-gates bounded} \end{aligned}$$

Alternation/Circuit Theorem

For $i \geq 1$:

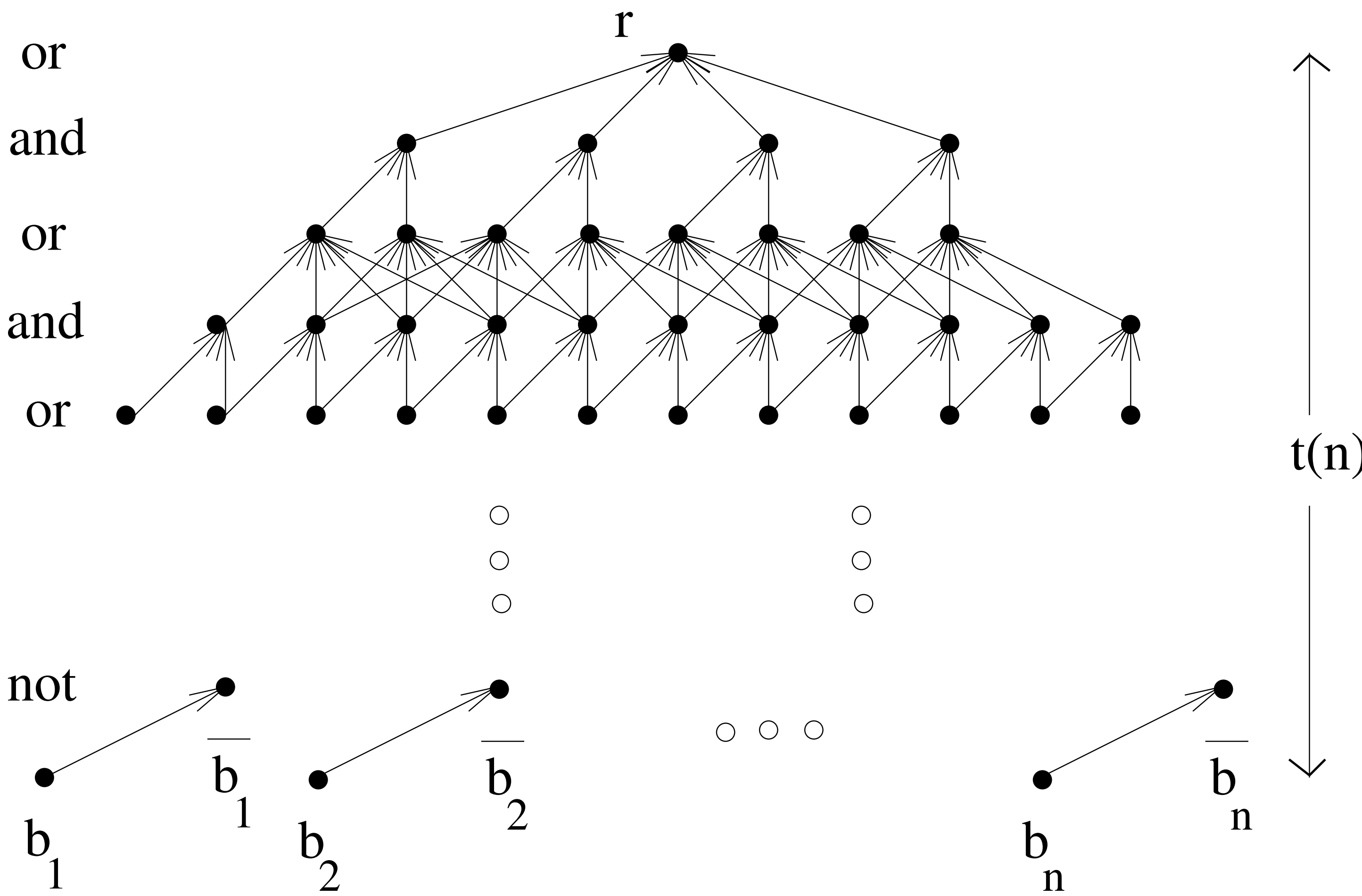
- $\mathbf{NC}^i$ equals ATM's with $O(\log n)$ space, $O(\log^i n)$ time
- $\mathbf{AC}^i$ equals ATM's with $O(\log n)$ space, $O(\log^i n)$ alternations

Proof Outline: Simulate ATM's by circuits by making a node for each configuration. Simulate circuits by ATM's using the Circuit Game.

Note that the $\mathbf{AC}^0$ case of the statement of this theorem is *false*. $\mathbf{AC}^0$ is *not* equal to ATM's with $O(\log n)$ space and $O(1)$ alternations: that's the *logspace* hierarchy. On HW#7 you used Immerman-Szelepcsenyi to prove that this hierarchy is just **NL**.

What $\mathbf{AC}^0$ actually equals is the *log-time hierarchy*, ATM's with $O(\log n)$ *time* and $O(1)$ alternations. This is also equal to FO given the suitable precise definitions.

sAC$[t(n)]$ Circuit



unbounded "or"s; bounded "and"s

Fact 25.1 sAC^1 $=$ LOG(CFL)$=$

$\{A \mid \exists\}$a CFL $B : A \leq B)$

We'll conclude our discussion of parallel complexity by showing where another one of our existing classes, the context-free languages, fits into the **NC** hierarchy.

Theorem 25.2 *(Ruzzo) If G is any context-free grammar, $\mathcal{L}(G) \in$* **sAC**^{1}.

Proof: Using the Alternation/Circuit theorem, we'll prove this by designing an ATM game for $\mathcal{L}(G)$ that has the following properties:

- White wins the game on input w iff $w \in \mathcal{L}(G)$,
- the game uses $O(\log n)$ space,
- the number of alternations is $O(\log n)$, and
- all Black's alternation phases consist of a single bit move.

When we covert this game to a circuit, the last clause ensures that all the AND gates have fan-in two, so we are in **sAC**^{1}. (Though our best upper bound for REACH is also **sAC**^{1}, it is believed that REACH is not complete for **sAC**^{1} while there are CFL's that are complete for it.)

Let's assume that G is in Chomsky normal form (only rules of the form $A \to BC$, $A \to a$, or $S \to \epsilon$). We have an input string w, and White claims there is a way to derive $S \to w$ using the rules of G. Black, as usual, disputes this.

White advances her claim by naming a node in the middle of the parse tree and saying what it does. Specifically, for some i, j, and A she says $S \to w_1 \ldots w_i A w_{j+1} \ldots w_n$ and $A \to w_{i+1} \ldots w_j$. Black picks one of these two claims to challenge.

If White is telling the truth about the orginal claim, she can get two true claims by telling the truth. But if she is lying, one of her two subsidiary claims must be a lie. We continue the process until we have a claim about a single input letter, such as $A \to w_i$, which can be verified by looking up the input letter and checking the rules of G.

This is a valid ATM game that decides whether $w \in \mathcal{L}(G)$, but it does not yet meet our specification. There are two problems:

- The game could last as long as $n - 1$ moves, rather than the $O(\log n)$ we need, and
- The subclaim under dispute might not be specifiable in space $O(\log n)$, as it has the form

  $$A \to w_{i_1} \dots w_{i_2} B w_{i_3} \dots w_{i_4} C w_{i_5} \dots w_{i_k}.$$

  We need $O(\log n)$ bits to record each "scar" in the string.

We solve the first problem by setting a fair time limit on White. If she has not reduced the claim to one letter in $O(\log n)$ moves, she loses. But why is this fair? On her move, she is dividing the *parse tree* of w into two pieces by cutting an edge.

Lemma: (Lipton-Tarjan) Any binary tree can be cut on some edge into two pieces, each at most 2/3 the original size. (Proof on Spring 2003 HW#8, solutions on my web site.)

So since White is so smart, she can choose her division to leave smaller subtrees, and after $O(\log n)$ moves she can reduce the subtree to one node.

To solve the second problem, we force White to make sure that the current claim is about a tree with at most three scars, giving her $O(\log n)$ more moves to spend on this goal.

Lemma: Let T be any rooted binary tree and let a, b, and c be any three nodes none of which is an ancestor of another. Then there exists a node d that is an ancestor of exactly two of a, b, and c. (Proof on Spring 2003 HW#8, with posted solutions.)

Now if White is faced with a tree with scars at a, b, and c, we force her to find some d and divide the tree there. This may not shrink the tree under dispute very much, but it makes sure that on the *next* move, the two subclaims have only two scars each.

White still wins the revised game iff she should, and the revised game now fits all the specifications. ♠

What we call $\mathbf{ThC}^i$ here is often called $\mathbf{TC}^i$ elsewhere.

$\mathbf{ThC}^0$ is the set of languages solvable by threshold cirucits of poly size and constant depth.

We proved $\mathbf{ThC}^0 \subseteq \mathbf{NC}^1$ by showing how to add n n-bit numbers in $\mathbf{NC}^1$, using *redundant binary notation*, base two with digits $\{0, 1, 2, 3\}$.

This has the effect that there are now many different "funny" ways to write the same number. The idea is that we can add two funny numbers in $\mathbf{NC}^0$, so we can add n of them in $\mathbf{NC}^1$ and then finally convert the funny result to standard binary in $\mathbf{AC}^0 \subseteq \mathbf{NC}^1$.

On HW#8 you'll provide some of the details of the construction to add two funny numbers in $\mathbf{NC}^0$.

Some problems in $\mathbf{TC}^0$:

- Addition of two n-bit numbers is in FO $=$ $\mathbf{AC}^0$ (the carry look-ahead adder)

- Addition of n n-bit numbers is in $\mathbf{ThC}^0$ but not in $\mathbf{AC}^0$ (by redundant notation)

- Multiplication of two n-bit numbers is in $\mathbf{ThC}^0$ but not in $\mathbf{AC}^0$.

- Sorting of n n-bit numbers is in $\mathbf{ThC}^0$. (Compare each of the n^2 pairs in parallel, then count up the wins for each item to get its place.)

- Division (and iterated multiplication) of two n-bit numbers (to n bits of accuracy) is in polynomial-time uniform $\mathbf{ThC}^0$. ([BCH86], [Reif87], using Chinese remainder notation)

- Division is in (first-order uniform) $\mathbf{ThC}^0$. ([Bill Hesse01], [HAB02])

Lower Bounds Against $\mathbf{AC}^0$:

We just asserted that the iterated addition and multiplication problems are not in $\mathbf{AC}^0$. How could one prove such a thing?

The argument is called a *size lower bound* for constant-depth, unbounded fan-in circuits. Lower bounds often call for detailed combinatorial arguments. In this case Furst-Saxe-Siper (yes, Sipser the author and Sipser my Ph.D. advisor) and Ajtai proved in the early 1980's that for any d, depth-d unbounded fan-in circuits need super-polynomial size to decide the language

PARITY $= \{w : w$ has an odd number of ones$\}$.

It is not hard to show that PARITY *circuit-reduces* to iterated addition and to multiplication, as defined in HW#7. By the downward closure of $\mathbf{AC}^0$, then, neither of these problems can be in $\mathbf{AC}^0$.

The key to the lower bound proof, which we won't cover in this course, is *randomized restriction*. They show that by setting all but $\sqrt{n}$ bits of the input randomly to 0 or 1, you can turn a depth-d circuit computing PARITY of n variables to a slightly larger depth-$(d-1)$ circuit computing PARITY of the remaining variables. Repeating this process leads to a contradiction unless the original circuit was superpolynomial size.

Later Yao and Håstad showed that a depth-d PARITY circuit must have exponential size. In 1986 Smolensky considered circuits that along with AND and OR gates also have gates counting modulo some constant m. He showed that with a *prime* modulus p you cannot count the ones in the input modulo any number that is not a power of p.

Embarassingly, it remains open to show any meaningful size lower bounds on constant-depth circuits of AND, OR, and mod-m gates where m is 6, or any product of two or more primes. Such circuits might, for all we can prove, be able to solve **NP**-complete problems.

$$\textbf{PSPACE} \quad = \quad \textbf{DSPACE}[n^{O(1)}] \quad = \quad \textbf{NSPACE}[n^{O(1)}]$$

- **PSPACE** consists the problems we could solve with a feasible amount of hardware, but with limit on computation time.
- **PSPACE** is a large and very robust complexity class.
- With polynomially many bits of memory, we can search any implicitly-defined graph of exponential size. This leads to complete problems such as reachability on exponentially-large graphs.
- We can search the game tree of any board game whose configurations are describable with polynomially-many bits. This leads to complete problems concerning winning strategies.

Recall that part of Lecture 23's Alternation Theorem says:

$$\textbf{PSPACE} \quad = \quad \textbf{ATIME}[n^{O(1)}]$$

Recall QSAT, the quantified satisfiability problem, which is the set of *true quantified boolean formulas*.

Proposition 25.3 QSAT *is* **PSPACE**-*complete.*

Proof: In Lecture 23 we proved that QSAT is in **ATIME**[n] and hence in **ATIME**$[n^{O(1)}]$. This is because the two players can guess the quantified boolean variables, and the referee can then evaluate the formula with these values.

It remains to show that QSAT is hard for $\mathbf{ATIME}[n^k]$:

Let M be an arbitrary $\mathbf{ATIME}[n^k]$ machine.

Let M write down its n^k alternating choices, $c_1 c_2 \ldots c_{n^k}$, and then deterministically evaluate its input, using the choice vector $\overline{c}$.

Let the corresponding $\mathbf{DTIME}[n^k]$ machine be D.

For all inputs w,

$$M(w) = 1 \quad \Leftrightarrow \quad (\exists c_1)(\forall c_2)\cdots(Q_{n^k} c_{n^k})(D(\overline{c}, w) = 1)$$

Applying the tableau construction from the proof of the Fagin or Cook-Levin theorems to D, we see that there is a reduction f from $\mathcal{L}(D)$ to SAT, so that:

$$D(\overline{c}, w) = 1 \quad \Leftrightarrow \quad f(\overline{c}, w) \in \text{SAT}$$

Let the new boolean variables in $f(\overline{c}, w)$ be $d_1 \ldots d_{t(n)}$.

M accepts w iff

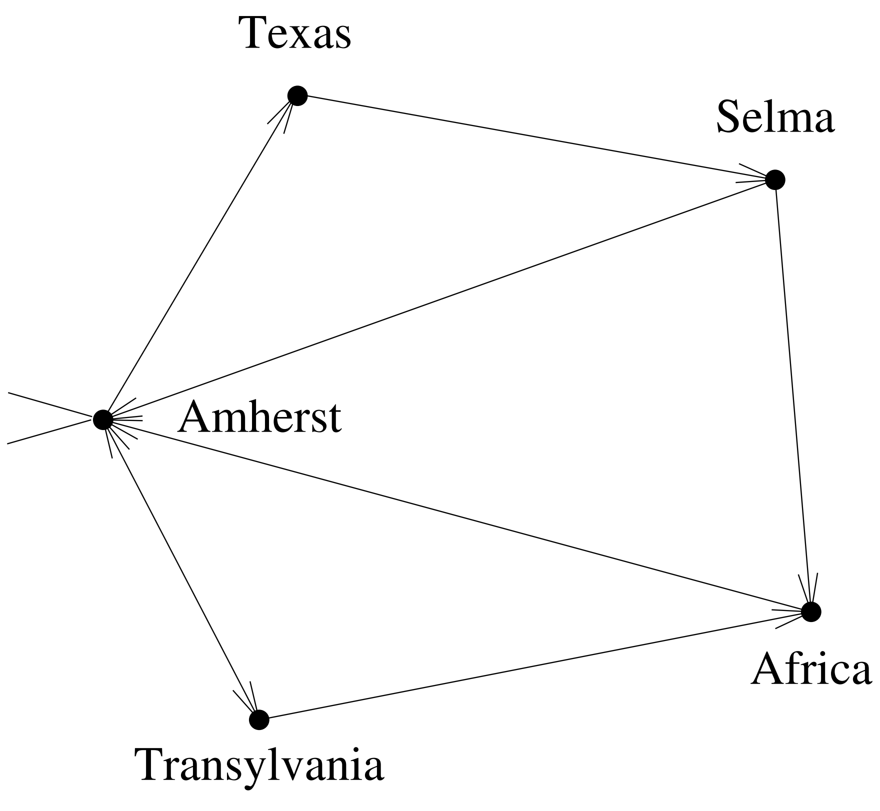
"$(\exists c_1)(\forall c_2)\cdots(Q_{n^k} c_{n^k})(\exists d_1 \ldots d_{t(n)}) f(\overline{c}, w)$" $\in$ QSAT

♠

GEOGRAPHY is a game with players E and A, played on a directed graph with start node s.

1. E chooses a vertex v_1 with an edge from s.
2. A chooses v_2, having an edge from v_1
3. E chooses v_3, have an edge from v_2

And so on. No vertex may be chosen twice. Whoever moves last wins. (In the original version of the game, the vertices consist of all known place names, and there is an edge from u to v iff the last letter of u's name is the first letter of v's, as in this graph:



Proposition 25.4 *Figuring out which player has a winning strategy in a given position of* GEOGRAPHY *is* **PSPACE**-*complete.*

Proof:

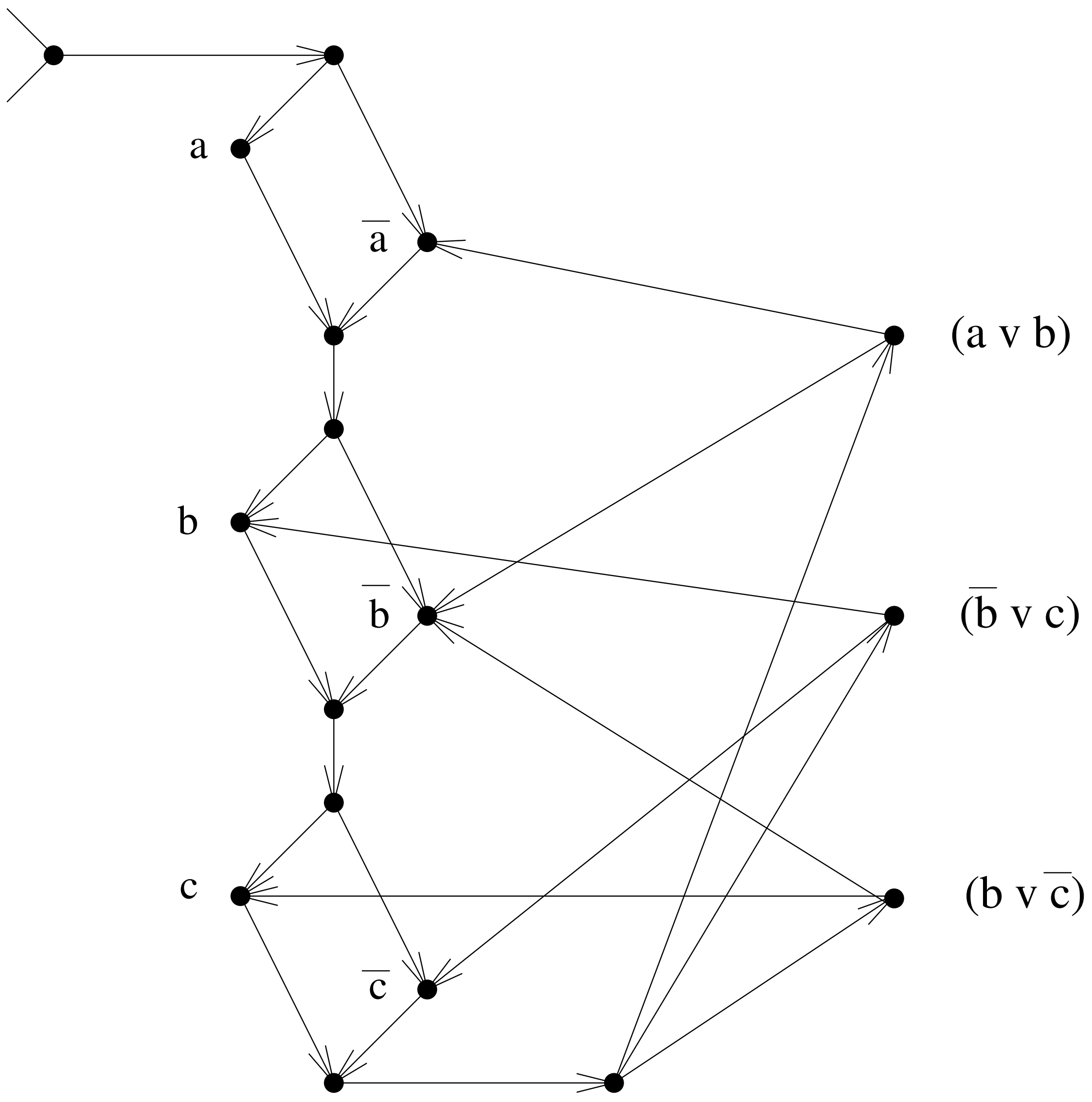
GEOGRAPHY $\in$ **PSPACE:** search the polynomial-depth game tree.

Other Half: QSAT $\leq$ GEOGRAPHY

Given a quantified boolean formula, φ, there is a general construction to build a graph G_φ such that player E wins the game on G_φ iff the formula is true. We first set up vertices so that player E chooses a value for each existential variable and player A choose one for each universal variables. Then we make vertices for each graph so that A's last move picks a clause, and E can then move iff that clause is satisfied by some literal set to true by one of the players.

Here is the graph G_φ where

$$\varphi \equiv \exists a \forall b \exists c[(a \vee b) \wedge (\bar{b} \vee c) \wedge (b \vee \bar{c})]$$



Definition 25.5 A *succinct* representation of a graph is $G(n, C) = (V, E, s, t)$ where C is a boolean circuit with $2n$ inputs and

$$\begin{aligned} V &= \{w \mid w \in \{0,1\}^n\} \\ E &= \{(w, w') \mid C(w, w') = 1\} \end{aligned}$$

$$\text{SUCCINCT REACH} = \{(n, C, s, t) \mid G(n, C) \in \text{REACH}\}$$

♠

Proposition 25.6 SUCCINCT REACH *is* **PSPACE**-*complete.*

Proof: This was assigned as a problem on Spring 2003's Homework 8, and is solved on the web pages for that version of the course. ♠

Suitably generalized to n by n versions, Go and Chess are also both **PSPACE**-complete. In general, an n by n game where the playing board can *change* is going to be **PSPACE**-complete because it can simulate alternating poly-time. A game where $O(1)$ pieces move around on a board is going to be **P**-complete because it can simulate alternating log space.