Recall: Circuit Complexity

depth = parallel time

width = hardware

number of gates = computational work = sequential time

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

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

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

$\mathbf{NC}[t(n)]$ = ParallelTime$[t(n)]$ on real hardware

$\mathbf{AC}[t(n)] = \mathbf{CRAM}[t(n)]$

$\mathbf{ThC}[t(n)]$ = ParallelTime$[t(n)]$ on “wet-ware”

$\mathbf{sAC}^i = \mathbf{AC}^i$ but $\wedge$-gates bounded

Alternation/Circuit Theorem

For $i \geq 1$:

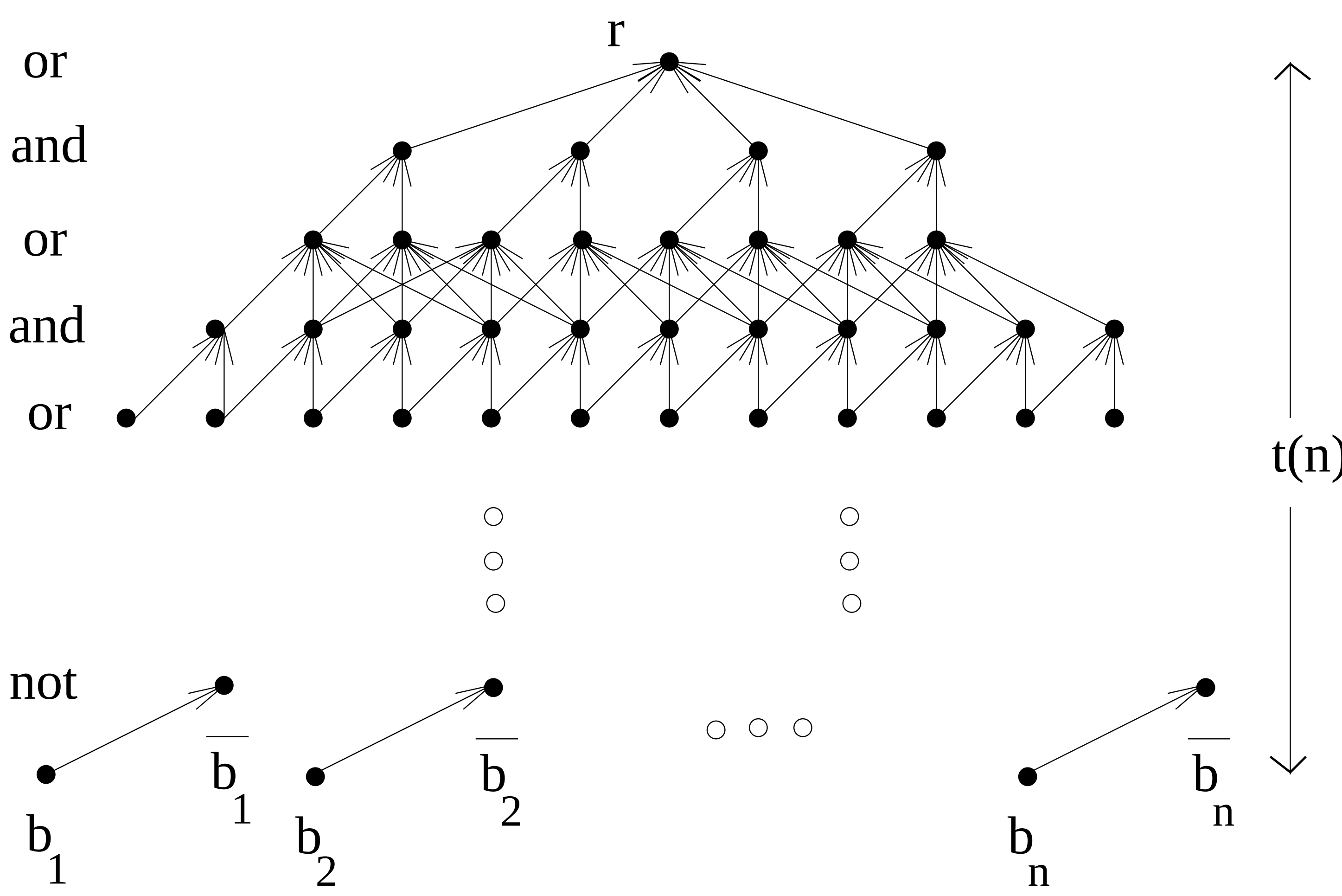
- **NC**^{i} equals ATM's with $O(\log n)$ space, $O(\log^i n)$ time
- **AC**^{i} equals ATM's with $O(\log n)$ space, $O(\log^i n)$ alternations

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

Note the **AC**^{0} case of the statement of this theorem is *false*. **AC**^{0} is *not* equal to ATM's with $O(\log n)$ space and $O(1)$ alternations: that's the *logspace* hierarchy, known by Immerman-Szelepcsenyi to be just **NL**.

What **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 24.1 sAC^1 $=$ **LOG(CFL)**$= \{A \mid \exists \text{CFL } L(A \leq L)\}$

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

Theorem 24.2 *(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 G is in Chomsky normal form. We have an input string w, and White claims there is a way to derive $S \rightarrow w$ 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 $S \rightarrow w_1 \dots w_i A w_{j+1} \dots w_n$ and $A \rightarrow w_{i+1} \dots w_j$. 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 $A \rightarrow w_i$, 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

  $$A \rightarrow w_{i_1} \ldots w_{i_2} B w_{i_3} \ldots w_{i_4} C w_{i_5} \ldots w_{i_k}.$$

  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 HW#8.)

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 HW#8.)

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. ♠

$O(1)$-Depth Threshold Circuits

What we call **ThC**^{i} here is often called **TC**^{i} elsewhere.

ThC^0 is the set of languages solvable by threshold cirucits of poly size and constant depth.

We proved **ThC**^{0} $\subseteq$ **NC**^{1} by showing how to add n n-bit numbers in **NC**^{1}, using *ambiguous 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 **NC**^{0}, so add n of them in **NC**^{1}, and finally convert the funny result to standard binary in **AC**^{0} $\subseteq$ **NC**^{1}.

I mangled that proof slightly last lecture, and will fix it now. Remember that we want to add two funny numbers, and get a funny result, without having to propagate carries.

First, in each column i we add two numbers from $\{0, 1, 2, 3\}$ to get a result r_i in the range from 0 through 6. We will carry a number c_i and keep $r_i - 2c_i$ for this columns result. But we have to choose c_i carefully to avoid carry propagation – our goal is that for each i, the final result $r_i - 2c_i + c_{i-1}$ will be in $\{0, 1, 2, 3\}$.

How does column i set c_i? The principle is that each column will carry as much as it knows that it can safely, up to 3. So the column looks at $r_{i-1}/2$, adds it to r_i, and sets c_i to be the minimum of 3 and that sum over 2. Since c_{i-1} will be at least what it expected, c_i won't be too large. And since c_{i-1} can be only two more than expected, c_i won't be too small.

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 $\mathbf{ThC}^0 - \mathbf{AC}^0$ (by ambiguous notation)
- Multiplication of two n-bit numbers is in $\mathbf{ThC}^0 - \mathbf{AC}^0$
- 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])

PSPACE

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

- **PSPACE** consists of what we could compute with a feasible amount of hardware, but with no time limit.
- **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.

PSPACE-Complete Problems

Recall Lecture 22,

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

Recall QSAT, the quantified satisfiability problem.

Proposition 24.3 QSAT *is* **PSPACE**-*complete.*

Proof: QSAT is in **ATIME**[n] (Lecture 22).

QSAT is hard for **ATIME**$[n^k]$:

Let M be an arbitrary **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 **DTIME**$[n^k]$ machine be D.

For all inputs w,

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

By the proof of Cook's Theorem there is a reduction f from $\mathcal{L}(D)$ to SAT,

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

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

M accepts w iff

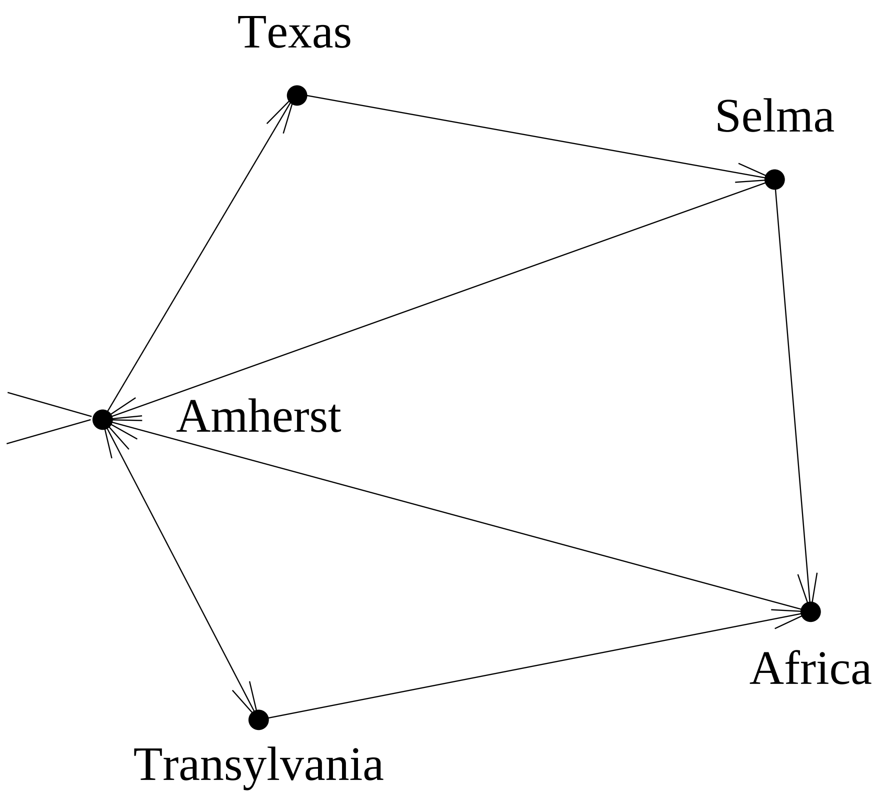
"$(\exists c_1)(\forall c_2) \cdots (Q_{n^k} c_{n^k})(\exists d_1 \ldots d_{t(n)}) f(\overline{c}, w)$" $\in$ QSAT

♠

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.



Proposition 24.4 *Figuring out which player has a winning strategy in a given position of* GEOGRAPHY *is*

PSPACE-*complete.*

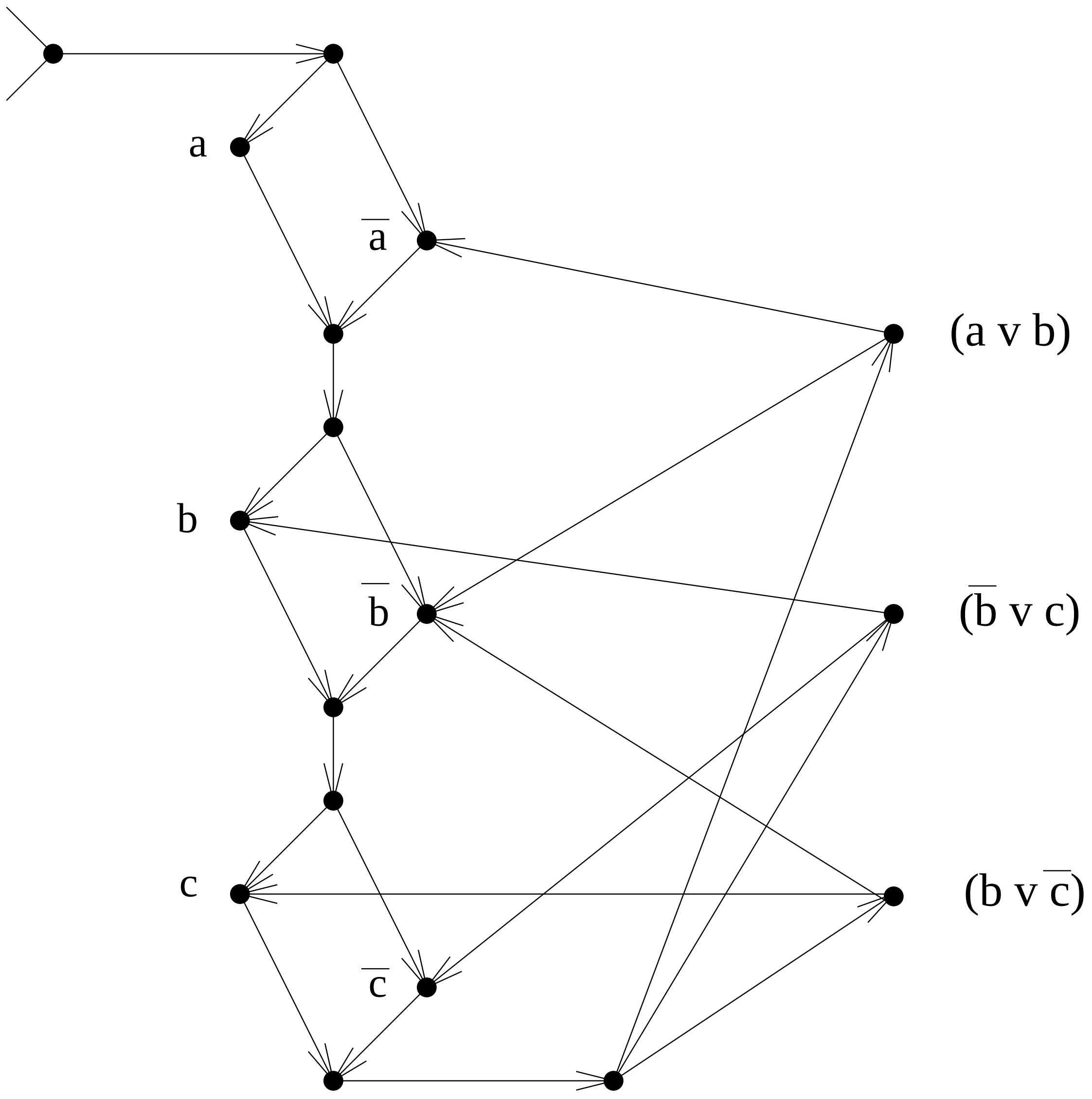
Proof:

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

Show: QSAT $\leq$ GEOGRAPHY

Given formula, φ, build graph G_φ s.t. E chooses existential variables; A chooses universal variables.

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



Definition 24.5 A *succinct* representation of a graph is $G(n, C) = (V, E, s, t)$ where C is a boolean circuit with $2n$ inputs and

$$V = \{w \mid w \in \{0,1\}^n\}$$

$$E = \{(w, w') \mid C(w, w') = 1\}$$

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

♠

Proposition 24.6 SUCCINCT REACH *is* **PSPACE**-*complete.*

Proof: This is assigned as a problem on Homework 8. ♠

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.

Barrington’s Theorem

A *permutation* of a finite set S is a one-to-one onto function from S to itself, The *composition* of two permutations is the permutation we get by performing first one and then the next. Composition is not commutative – the set of all permutations of five elements form the *non-commutative group* S_5 with composition as the operation.

The S_5 iterated multiplication problem (S_5-MULT) is to input n elements of S_5 (in order) and determine their composition. Clearly a DFA can do this, so S_5-MULT is a regular language.

Theorem 24.7 *S_5-MULT is complete for* **NC**^{1}. *Specifically, if C is an n-input circuit of depth d and fan-in two, we can take a string x of length n and construct a sequence of 4^d permutations that multiplies to a non-identity permutation iff $C(x) = 1$.*

Notation: A *five-cycle* is $(abcde)$ is the permutation that takes a to b, b to c, c to d, d to e, and e to a, where {a,b,c,d,e} = {1,2,3,4,5}.

Lemma: There exist five-cycles σ and tau such that $\sigma\tau\sigma^{-1}\tau^{-1}$ is a five-cycle. (This permutation is called the *commutator* of σ and τ.)

Proof: $(12345)(13542)(54321)(24531) = (13254)$.

Fact: (basic group theory) If α and β are both five-cycles, then $\alpha = \gamma\beta\gamma^1$ for some permutation γ

Proof: (of Barrington's Theorem) Use induction on the depth d of the circuit. For each gate g we'll construct a sequence $s(g)$ such that $s(g)$ evaluates to a five-cycle if g evaluates to 1 and $s(g)$ evaluates to the identity otherwise. By the Fact, if we can get one five-cycle we can get any other with a sequence of the same length.

Base Case: $d = 0$ and the gate is an input. Look up the literal and let $s(g)$ consist of one permutation, (12345) if the literal is true and the identity if it is false.

NOT Gates: If h is the NOT of g, compose $s(g)$ with (54321). This gives the identity if g is true and (54321) if g is false. Using the Fact, normalize to give (12345) if h is true and the identity if h is false.

AND Gates: Suppose h is the AND of g_1 and g_2 and each of g_1 and g_2 have depth d. Using $s(g_1)$ and $s(g_2)$, we construct four sequences of length 4^d each:

- a_1 yields (12345) if g_1 is true and the identity otherwise,
- a_2 yields (13542) if g_2 is true and the identity otherwise,
- b_1 yields (54321) if g_1 is true and the identity otherwise, and
- b_2 yields (24531) if g_2 is true and the identity otherwise.

Calculation: $a_a a_2 b_1 b_2$ yields (13254) if g_1 and g_2 are both true, and the identity otherwise.

Conclusion: If C is a depth $O(\log n)$ circuit, we get a sequence of length $4^{O(\log n)}$, which is polynomial. We have reduced the circuit evaluation problem to an S_5-MULT instance that is only polynomial size. ♠

Application to PSPACE

Fact: **PSPACE** is characterized by circuits of polynomial *depth*.

Corollary: Any **PSPACE** problem can be reduced to an instance of S_5-MULT of length $2^{O(\log n)}$.

Corollary: (Cai-Furst) Any **PSPACE** problem can be solved by a log-space Turing machine that:

- has access to a read-only clock
- wipes its memory every poly-many steps, except for *three safe bits*.