# Complexity in Computer Science

## Solutions to Exercises

Thomas Watson

February 2, 2026

# Contents

## Chapter 14



## Chapter 15



## Chapter 16

**Exercise 1.1:**

- Reg[0] is $N$.
- Reg[1] is the loop counter and an address into the input segment.
- Reg[2] is the current number from the input.
- Reg[3] is the minimum of the input numbers seen so far.
- Reg[4] is the maximum of the input numbers seen so far.

The word size is $\lceil \log(N+1) \rceil$.

```
 0. read Reg[3] ← In[0]
 1. let Reg[4] ← Reg[3]
 2. let Reg[1] ← Reg[1] + 1
 3. if Reg[1] ≥ Reg[0] then goto line 11
 4. read Reg[2] ← In[Reg[1]]
 5. if Reg[2] ≥ Reg[3] then goto line 8
 6. let Reg[3] ← Reg[2]
 7. goto line 2
 8. if Reg[2] ≤ Reg[4] then goto line 2
 9. let Reg[4] ← Reg[2]
10. goto line 2
11. write Reg[3]
12. write Reg[4]
13. halt
```

**Exercise 1.2:** The program uses register aliases $N$, $n$, $y$, $lo$, $hi$, $mid$, and $xval$. The loop invariant is that the lowest address $i \in \{0, 1, \ldots, n-1\}$ such that $x_{i+1} = y$, if it exists, must be between $lo$ and $hi$, inclusive. No register ever exceeds $N$, except that $mid$ might be almost as large as $2N$ because of line 5. Thus word size $\lceil \log(2N) \rceil = O(\log N)$ suffices. The time efficiency is $O(\log N)$ since the loop has $O(\log N)$ iterations (as $hi - lo$ goes down by a factor of approximately 2 in each iteration), and $O(1)$ instructions are executed per iteration. The space efficiency is $O(\log N)$ since the work segment is not used.

```
 0. let n ← N − 1
 1. read y ← In[n]
 2. let lo ← 0
 3. let hi ← n − 1
 4. if lo = hi then goto line 13
 5.     let mid ← lo + hi
 6.     let mid ← ⌊mid/2⌋
 7.     read xval ← In[mid]
 8.     if y > xval then goto line 11
 9.     let hi ← mid
10.     goto line 4
11.     let lo ← mid + 1
12.     goto line 4
13. read xval ← In[lo]
14. if xval = y then goto line 16
15. reject
16. accept
```

**Exercise 1.3:** The program loops through the bit positions, from most significant to least significant, to find the first position on which $x$ and $y$ differ. It uses register aliases $N$, *xptr*, *yptr*, *xbit*, and *ybit*. The word size is $\lceil \log(N+1) \rceil = O(\log N)$. The time efficiency is $O(N)$ since the loop has at most $N/2$ iterations, and $O(1)$ instructions are executed per iteration. The space efficiency is $O(\log N)$ since the work segment is not used.

```
 0.  let xptr ← ⌊N/2⌋
 1.  let yptr ← N
 2.  if xptr = 0 then goto line 10
 3.      let xptr ← xptr − 1
 4.      let yptr ← yptr − 1
 5.      read xbit ← In[xptr]
 6.      read ybit ← In[yptr]
 7.      if xbit < ybit then goto line 10
 8.      if xbit > ybit then goto line 11
 9.      goto line 2
10.  reject
11.  accept
```

**Exercise 1.4:** For ELEMENT DISTINCTNESS $\in$ TIME[$N$], the idea is to take one pass through the input list, using the work segment to remember which values have been seen already. For any nonnegative integer $y \leq N^2$, Mem[$y$] will be 1 if $y$ has been encountered in the input so far ("discovered"), and 0 otherwise. The program rejects if it sees an input value that was previously seen, and accepts if it finishes the input without encountering any duplicates. It uses register aliases $N$, $i$, $xi$, and $disc$. For convenience we think of the indices as $0, \ldots, N-1$, so register $xi$ actually gets $x_{i+1}$ from the input. No register ever exceeds $N^2$, so word size $\lceil \log(N^2+1) \rceil = O(\log N)$ suffices. The time efficiency is $O(N)$ since the loop has only $N$ iterations, and $O(1)$ instructions are executed per iteration. This implementation's space efficiency is $O(N^2 \log N)$, but it doesn't matter.

```
0.  if i ≥ N then goto line 8
1.      read xi ← In[i]
2.      load disc ← Mem[xi]
3.      if disc = 0 then goto line 5
4.          reject
5.      store Mem[xi] ← 1
6.      let i ← i + 1
7.      goto line 0
8.  accept
```

For ELEMENT DISTINCTNESS $\in$ SPACE[$\log N$], the idea is to compare each pair of input elements and reject iff a duplicate is encountered. The program only needs to remember the current pair of indices $0 \leq i < j < N$. It uses register aliases $N$, $i$, $j$, $xi$, and $xj$. No register ever exceeds $N^2$, so word size $\lceil \log(N^2+1) \rceil = O(\log N)$ suffices. The space efficiency is $O(\log N)$ since the work segment is not used. The time efficiency is $O(N^2)$, but it doesn't matter.

```
0.  if i ≥ N then goto line 11
1.      read xi ← In[i]
2.      let j ← i + 1
3.      if j ≥ N then goto line 9
4.          read xj ← In[j]
5.          if xi ≠ xj then goto line 7
6.              reject
7.          let j ← j + 1
8.          goto line 3
9.      let i ← i + 1
10.     goto line 0
11. accept
```

**Exercise 1.5:** For each $u \in \{1, 2, \ldots, n\}$, Mem[$u$] will be 0 if node $u$ is undiscovered, and will be $u$'s predecessor otherwise. To mark $s$ as discovered, we let $s$ be its own predecessor (though this doesn't matter, as long as Mem[$s$] $\neq 0$). The second phase of the algorithm (line 29 through line 34) outputs the path backward from $t$ to $s$. The same word size as plain BFS works.

```
 0. let ptr ← N − 2
 1. read s ← In[ptr]
 2. let ptr ← N − 1
 3. read t ← In[ptr]
 4. if s ≠ t then goto line 7
 5.     write t
 6.     halt
 7. store Mem[s] ← s
 8. read head ← In[0]
 9. let head ← head + 1
10. store Mem[head] ← s
11. let tail ← head + 1
12. if head = tail then goto line 28
13.     load u ← Mem[head]
14.     let head ← head + 1
15.     read ptr ← In[u]
16.     read outdeg ← In[ptr]
17.     let end ← ptr + outdeg
18.     if ptr = end then goto line 12
19.         let ptr ← ptr + 1
20.         read v ← In[ptr]
21.         load pred ← Mem[v]
22.         if pred ≠ 0 then goto line 18
23.         store Mem[v] ← u
24.         if v = t then goto line 29
25.         store Mem[tail] ← v
26.         let tail ← tail + 1
27.         goto line 18
28. reject
29. write v
30. if v = s then goto line 34
31.     load v ← Mem[v]
32.     write v
33.     goto line 30
34. halt
```

**Exercise 1.6:** The first loop (line 1 through line 12) tabulates the indegrees of all nodes. The second loop (line 16 through line 22) initializes the queue to contain all nodes with indegree 0 in $G$. The third loop (line 23 through line 40) "deletes" each node of indegree 0. The *count* register records how many nodes have been deleted and written to the output segment. If not all nodes get deleted, the graph must not be a dag. The program's word size is $\lceil \log(N+1) \rceil$ since no register ever exceeds $N$.

```
 0. read n ← In[0]
 1. if u ≥ n then goto line 13
 2.     let u ← u + 1
 3.     read ptr ← In[u]
 4.     read outdeg ← In[ptr]
 5.     let end ← ptr + outdeg
 6.     if ptr = end then goto line 1
 7.         let ptr ← ptr + 1
 8.         read v ← In[ptr]
 9.         load indeg ← Mem[v]
10.         let indeg ← indeg + 1
11.         store Mem[v] ← indeg
12.         goto line 6
13. let head ← n + 1
14. let tail ← head
15. let u ← 0
16. if u ≥ n then goto line 23
17.     let u ← u + 1
18.     load indeg ← Mem[u]
19.     if indeg > 0 then goto line 16
20.     store Mem[tail] ← u
21.     let tail ← tail + 1
22.     goto line 16
23. if head = tail then goto line 41
24.     load u ← Mem[head]
25.     let head ← head + 1
26.     write u
27.     let count ← count + 1
28.     read ptr ← In[u]
29.     read outdeg ← In[ptr]
30.     let end ← ptr + outdeg
31.     if ptr = end then goto line 23
32.         let ptr ← ptr + 1
33.         read v ← In[ptr]
34.         load indeg ← Mem[v]
35.         let indeg ← indeg − 1
36.         store Mem[v] ← indeg
37.         if indeg > 0 then goto line 31
38.         store Mem[tail] ← v
39.         let tail ← tail + 1
40.         goto line 31
41. if count < n then goto line 43
42. halt
43. reject
```

**Exercise 1.7:** If $G$ has $n$ nodes then the input size is $N = n^2$, since the adjacency matrix has one bit for each ordered pair of nodes (indicating whether $G$ has an edge between those nodes).

```
compute n ← √N
for u ← 1,...,n−2:
    for v ← u+1,...,n−1:
        for w ← v+1,...,n:
            if each of {u,v}, {v,w}, {w,u} is an edge in G: accept
reject
```

The value of $n$ can be computed in $O(\log N)$ time with binary search, as mentioned in § 1.5.4. The algorithm's time efficiency is $O(n^3) = O(N^{1.5})$ since each of the nested loops has at most $n$ iterations, and each iteration contributes $O(1)$ to the running time (since each of the three edge look-ups $\{u,v\}$, $\{v,w\}$, $\{w,u\}$ takes $O(1)$ instructions). The space efficiency is $O(\log n) = O(\log N)$ since $n$, $u$, $v$, $w$ each take at most $\lceil \log(\sqrt{N}+1) \rceil$ bits, so an implementation would be register-only with word size $O(\log N)$.

**Exercise 1.8.a:** Let $x_{i,n-1} \cdots x_{i,0}$ be the binary representation of $x_i$, so $x_i = \sum_{j=0}^{n-1} x_{i,j} \cdot 2^j$. Our algorithm determines whether there are more 0s or 1s among the most significant bits of all the $x_i$ numbers; the variable $c$ counts how many 0s.

compute $n \leftarrow \sqrt{N}$
initialize $c \leftarrow 0$
for $i \leftarrow 1, \ldots, n$:
    if $x_{i,n-1} = 0$: increment $c$
if $c \geq \lceil n/2 \rceil$: reject
else: accept

The time efficiency is $O(n) = O(\sqrt{N})$ since $n$ can be computed in $O(\log N)$ time (as mentioned in §1.5.4) and the loop has $n$ iterations and $O(1)$ instructions per iteration. Word size $O(\log N)$ suffices, so the space efficiency is $O(\log N)$ since an implementation would be register-only (using registers for the variables $n$, $c$, $i$, and intermediate calculations).

**Exercise 1.8.b:** Our algorithms identify the exact median (after which, outputting its least significant bit is trivial). For Least Significant Bit of Median $\in$ TIME$[N]$, the following algorithm maintains these outer loop invariants:

- The overall median is the $k^{\text{th}}$ smallest among all $x_i$ such that $i$ remains "in contention."
- $x_{i,n-1} \cdots x_{i,j+1}$ is the same for all $i$ in contention.

```
compute n ← √N
initialize k ← ⌈n/2⌉
mark each i ∈ {1,...,n} as "in contention"
for j ← n−1,...,0:
    initialize c ← 0
    for i ← 1,...,n:
        if i is in contention and x_{i,j} = 0: increment c
    if c ≥ k:
        for i ← 1,...,n:
            if i is in contention and x_{i,j} = 1: mark i "not in contention"
    else if c < k:
        for i ← 1,...,n:
            if i is in contention and x_{i,j} = 0: mark i "not in contention"
        update k ← k−c
pick any i that remains in contention
output x_{i,0}
```

The invariants hold at the beginning (with $j = n-1$) and are preserved by each outer iteration: The variable $c$ counts how many numbers in contention have 0 in position $j$. If $c \geq k$ then the median has 0 in position $j$, and if $c < k$ then the median has 1 in position $j$. The input numbers with the wrong bit in position $j$ are culled from contention (maintaining the second invariant), and $k$ is updated accordingly (maintaining the first invariant). Correctness of the output is implied by the invariants holding at termination (with $j = -1$) since all $x_i$ numbers in contention are identical and the median is among them.

The time efficiency is $O(n^2) = O(N)$ since the part before the outer loop contributes $O(n)$, the outer loop has $n$ iterations, each inner loop has $n$ iterations, and each iteration contributes $O(1)$ instructions. An implementation would use a list of $n$ bits in the work memory to keep track of which indices $i$ are in contention, so that $O(1)$ time suffices to look up (and change) whether $i$ is contention.

For Least Significant Bit of Median $\in$ SPACE$[\log N]$, the idea is to try each $x_i$ and check whether it's the median, by comparing it to all input numbers. Note that $x_i$ is the median iff there are fewer than $\lceil n/2 \rceil$ indices $j$ with $x_j < x_i$ and fewer than $\lceil n/2 \rceil$ indices $j$ with $x_j > x_i$; the variables $a$ and $b$ count these two things respectively.

compute $n \leftarrow \sqrt{N}$
for $i \leftarrow 1, \ldots, n$:
    initialize $a \leftarrow 0$ and $b \leftarrow 0$
    for $j \leftarrow 1, \ldots, n$:
        compute whether $x_j$ is $<$, $=$, or $>$ $x_i$ as in Exercise 1.3
        if $x_j < x_i$: increment $a$
        else if $x_j > x_i$: increment $b$
    if $a < \lceil n/2 \rceil$ and $b < \lceil n/2 \rceil$: terminate and output $x_{i,0}$

The algorithm only needs to remember $n$, $i$, $a$, $b$, $j$, and an index for walking through $x_i$ and $x_j$ to see which is greater by comparing bits in most-to-least-significant order. None of these variables ever exceeds $n$, so they can be kept in registers with word size $O(\log N)$. Thus the space efficiency is $O(\log N)$.

**Exercise 1.9:** Suppose $x_i$ has bit-length $m_i$, and denote the binary representation of $x_i$ by $x_{i,m_i-1} \cdots x_{i,0}$. Define $m = \lceil \log n \rceil + \max_{i=1}^{n} m_i$, and note that $\sum_{i=1}^{n} x_i < n \cdot \max_{i=1}^{n} 2^{m_i} \leq 2^m$, so $m$ bits are enough for the output. For any given $i$ and $j$, we can look up the bit $x_{i,j}$ in $O(1)$ time.

For each $i$, a "pointer" variable $p_i$ holds the index of the next input number $x_{p_i}$ that has remaining bits (as we sweep from least to most significant). So $p_0$ is the lowest index of an input number with remaining bits, and if $p_i > n$ then no input numbers with index $> i$ have remaining bits. If $x_i$ has no remaining bits, then $p_i$ is junk.

```
initialize c ← 0
for i ← 0, 1, ..., n: initialize p_i ← i + 1
for j ← 0, 1, ..., m − 1:
    initialize i ← 0
    while p_i ≤ n:
        if j < m_{p_i}:
            update i ← p_i
            update c ← c + x_{i,j}
        else: update p_i ← p_{p_i}
    let z_i ← c mod 2
    update c ← ⌊c/2⌋
output the binary number z_{m−1} ··· z_1 z_0
```

This algorithm is correct because it produces the same output as the standard carry algorithm; it just saves time by skipping over input numbers that have run out of bits. Initializing the $p_i$s takes $O(n) = O(N)$ time. The rest of the time is $O(\text{number of iterations of the inner "while" loop})$. Let's "charge" an inner iteration to the input number $x_{p_i}$, for the value of $p_i$ at the beginning of the iteration. Each $x_k$ only gets charged for $m_k + 1$ many iterations, namely $m_k$ many iterations where the "if" condition is true ($j = 0, \ldots, m_k - 1$), and one iteration where it's false ($j = m_k$, so the "else" executes). This is because after the "else" executes, the pointer chain will skip over $x_k$ for the rest of the algorithm. Thus the time is $O\left(\sum_{k=1}^{n}(m_k + 1)\right) = O(N)$. The carry $c$ always fits in $O(\log N)$ bits (just like in Claim 1.1), as do all pointers and indices, so $O(\log N)$ word size suffices.

**Exercise 1.10:** Let $\deg(v)$ denote the degree of a node $v$. For $i \in \{0, 1, \ldots, \deg(v)-1\}$, let $v[i]$ denote $v$'s $i^{\text{th}}$ neighbor according to $v$'s adjacency list in the input. For any integer $i$, let $v[i]$ stand for $v[i \bmod \deg(v)]$. View an undirected edge $\{u, v\}$ as a pair of directed edges $(u, v)$ and $(v, u)$, and note that $u = v[i]$ and $v = u[j]$ for some $i$ and $j$ that are unrelated to each other.

```
if s = t: accept
if deg(s) = 0: reject
initialize (u, v) ← (s, s[0])
repeat until (u, v) = (s[−1], s):
    if v = t: accept
    find i such that u = v[i]
    update (u, v) ← (v, v[i + 1])
reject
```

This algorithm only needs registers for $s$, $t$, $u$, $v$, $i$, $\deg(s)$, $\deg(v)$, and some pointers into the input. None of these ever exceeds $N$, so we can keep them in registers with word size $O(\log N)$. Thus the space efficiency is $O(\log N)$.

We prove that this algorithm is correct. It is correct if $s = t$ or $\deg(s) = 0$, so assume $s \neq t$ and $\deg(s) > 0$.

We first argue that the algorithm always terminates. It traverses a walk from $s$, with $(u, v)$ as the current directed edge. Supposing for contradiction that it runs forever, it must traverse some directed edge at least twice, by the pigeonhole principle. Consider the first edge $(u, v)$ traversed at least twice. The edge preceding $(u, v)$ on the walk is uniquely determined as $(u[j-1], u)$ where $j$ is such that $v = u[j]$. If $(u, v) \neq (s, s[0])$ then the first two traversals of $(u, v)$ would both be preceded by the same edge, contradicting the assumption that $(u, v)$ is the first edge traversed at least twice. If $(u, v) = (s, s[0])$ then the second traversal of $(u, v)$ would be preceded by $(s[-1], s)$, so the algorithm actually would have terminated and rejected.

Now, we argue that the algorithm accepts iff $t$ is reachable from $s$:

$\Rightarrow$: Suppose the algorithm accepts. Since it traverses a walk from $s$ and only accepts when it reaches $t$, this means there exists a walk (and hence a path) from $s$ to $t$.

$\Leftarrow$: Suppose there exists a path $w_0 \to w_1 \to \cdots \to w_d$ from $s = w_0$ to $t = w_d$. We claim that the algorithm traverses each directed edge on this path, and thus reaches $t$ and accepts. Suppose for contradiction it doesn't. Consider the least $\ell$ such that the algorithm doesn't traverse $(w_\ell, w_{\ell+1})$.

First, suppose $\ell > 0$. Let $i$ be such that $w_{\ell-1} = w_\ell[i]$, and consider the least $k \in \{1, 2, \ldots, \deg(w_\ell) - 1\}$ such that the algorithm doesn't traverse $(w_\ell, w_\ell[i + k])$. Such a $k$ exists since $w_{\ell+1} \neq w_{\ell-1}$ and the algorithm doesn't traverse $(w_\ell, w_{\ell+1})$. If $k = 1$ then the algorithm would traverse $(w_\ell, w_\ell[i + k])$ immediately after traversing $(w_{\ell-1}, w_\ell)$, so we must have $k > 1$. Since we considered the least such $k$, the algorithm must traverse $(w_\ell, w_\ell[i + k - 1])$. We claim that after it does that, it eventually traverses the reverse edge $(w_\ell[i + k - 1], w_\ell)$, after which it would immediately traverse $(w_\ell, w_\ell[i+k])$, which is a contradiction. Proof of the claim: Since $s$'s connected component is a tree, deleting the undirected edge $\{w_\ell, w_\ell[i+k-1]\}$ would disconnect the component (Theorem 0.6). After deleting this edge, $s$ is in $w_\ell$'s connected component (by the path $s \to w_1 \to \cdots \to w_\ell$) and thus not in $w_\ell[i+k-1]$'s connected component, which means

the algorithm will not reject while it's inside $w_\ell[i+k-1]$'s connected component. Since the algorithm doesn't run forever, the walk must eventually leave that component in the only possible way, which is the edge $(w_\ell[i+k-1], w_\ell)$.

Now, suppose $\ell = 0$. Consider the least $k$ such that the algorithm doesn't traverse $(s, s[k])$. Such a $k$ exists since the algorithm doesn't traverse $(s, w_1)$. We must have $k > 0$ since $(s, s[0])$ is the first edge the algorithm traverses. The same argument from the previous paragraph shows that after traversing $(s, s[k-1])$, the algorithm would eventually traverse the reverse edge $(s[k-1], s)$ and then immediately traverse $(s, s[k])$, which is a contradiction.

**Exercise 1.11.a:**

$$\text{let } \text{Reg}[i] \leftarrow \text{Reg}[j] \Leftrightarrow \text{Reg}[k] \quad \rightsquigarrow \quad \begin{array}{l} \text{let } \text{Reg}[i] \leftarrow \text{Reg}[j] \oplus \text{Reg}[k] \\ \text{let } \text{Reg}[i] \leftarrow \neg\text{Reg}[i] \end{array}$$

Then, update the line numbers of all branch instructions.

**Exercise 1.11.b:** Observe that $x \Rightarrow y = (\neg x) \vee y$ for all $x, y \in \{0,1\}^W$. First, assume that there are no hardcoded constants and that $i \neq j \neq k$:

$$\begin{array}{lcl} \text{let Reg}[i] \leftarrow \text{Reg}[j] \Rightarrow \text{Reg}[k] & \rightsquigarrow & \text{let Reg}[j] \leftarrow \neg\text{Reg}[j] \\ & & \text{let Reg}[i] \leftarrow \text{Reg}[j] \vee \text{Reg}[k] \\ & & \text{let Reg}[j] \leftarrow \neg\text{Reg}[j] \end{array}$$

If $i = j \neq k$ then the last line of the substitution should be removed. If $j = k$ then:

$$\text{let Reg}[i] \leftarrow \text{Reg}[j] \Rightarrow \text{Reg}[j] \quad \rightsquigarrow \quad \text{let Reg}[i] \leftarrow \neg\, 0$$

If a constant stands in for Reg$[k]$ but not for Reg$[j]$, then the first substitution above still works (with the last line removed if $i = j$). If constants stand in for both Reg$[j]$ and Reg$[k]$, then Reg$[i]$ can just be assigned a negated constant (since if $x, y \in \{0,1\}^W$ are constants, then $\neg(x \Rightarrow y) = x \wedge (\neg y)$ is also a constant, though $(\neg x) \vee y$ itself is not a constant). Finally, suppose a constant stands in for Reg$[j]$ but not for Reg$[k]$. Assuming $i \neq k$:

$$\begin{array}{lcl} \text{let Reg}[i] \leftarrow x \Rightarrow \text{Reg}[k] & \rightsquigarrow & \text{let Reg}[i] \leftarrow \neg\, x \\ & & \text{let Reg}[i] \leftarrow \text{Reg}[i] \vee \text{Reg}[k] \end{array}$$

Assuming $i = k$:

$$\begin{array}{lcl} \text{let Reg}[i] \leftarrow x \Rightarrow \text{Reg}[i] & \rightsquigarrow & \text{let Reg}[i] \leftarrow \neg\text{Reg}[i] \\ & & \text{let Reg}[i] \leftarrow x \wedge \text{Reg}[i] \\ & & \text{let Reg}[i] \leftarrow \neg\text{Reg}[i] \end{array}$$

Then, update the target line numbers of all branch instructions.

**Exercise 1.12:** Perhaps the simplest way to check whether Reg[$j$]'s leftmost bit is 1 is to do an unsigned comparison to see whether Reg[$j$] is greater than its bitwise negation. Employing a fresh "register $h$":

| | | |
|---|---|---|
| let Reg[$i$] $\leftarrow$ Reg[$j$] $\ggg$ Reg[$k$] | $\rightsquigarrow$ | let Reg[$h$] $\leftarrow \neg$Reg[$j$]<br>if Reg[$j$] $>$ Reg[$h$] then goto 3 lines below<br>let Reg[$i$] $\leftarrow$ Reg[$j$] $\gg$ Reg[$k$]<br>goto 6 lines below<br>let Reg[$h$] $\leftarrow \neg\, 0$<br>let Reg[$h$] $\leftarrow$ Reg[$h$] $\gg$ Reg[$k$]<br>let Reg[$h$] $\leftarrow \neg$Reg[$h$]<br>let Reg[$i$] $\leftarrow$ Reg[$j$] $\gg$ Reg[$k$]<br>let Reg[$i$] $\leftarrow$ Reg[$i$] $\vee$ Reg[$h$] |

This also works with constants standing in for Reg[$j$] and/or Reg[$k$].

Then, update the target line numbers of all preexisting branch instructions.

**Exercise 1.13.a:** Note that $x \wedge y = \neg((\neg x) \vee (\neg y))$ for all $x, y \in \{0,1\}^W$. Employing two fresh "registers $g$ and $h$":

$$\begin{array}{l} \text{let Reg}[i] \leftarrow \text{Reg}[j] \wedge \text{Reg}[k] \end{array} \quad \rightsquigarrow \quad \begin{array}{l} \text{let Reg}[g] \leftarrow \neg\text{Reg}[j] \\ \text{let Reg}[h] \leftarrow \neg\text{Reg}[k] \\ \text{let Reg}[i] \leftarrow \text{Reg}[g] \vee \text{Reg}[h] \\ \text{let Reg}[i] \leftarrow \neg\text{Reg}[i] \end{array}$$

This also works with constants standing in for Reg[$j$] and/or Reg[$k$].

Then, update the target line numbers of all branch instructions, and apply Theorem 1.2.

The proof of Theorem 1.2 reintroduces bitwise-and instructions when masking Reg[0] to make the "effective word size" match the original word size, but this can be sidestepped by applying a similar substitution:

$$\begin{array}{l} \text{let Reg}[0] \leftarrow \text{Reg}[0] \wedge \text{Reg}[1] \end{array} \quad \rightsquigarrow \quad \begin{array}{l} \text{let Reg}[0] \leftarrow \neg\text{Reg}[0] \\ \text{let Reg}[1] \leftarrow \neg\text{Reg}[1] \\ \text{let Reg}[0] \leftarrow \text{Reg}[0] \vee \text{Reg}[1] \\ \text{let Reg}[0] \leftarrow \neg\text{Reg}[0] \end{array}$$

The mask in Reg[1] doesn't need to be kept (it may be overwritten in the next instruction anyway), so we don't bother re-negating it at the end of the substitution.

**Exercise 1.13.b:** Note that $x \vee y = \neg((\neg x) \wedge (\neg y))$ for all $x, y \in \{0,1\}^W$. Employing two fresh "registers $g$ and $h$":

$$\begin{array}{l} \text{let Reg}[i] \leftarrow \text{Reg}[j] \vee \text{Reg}[k] \end{array} \quad \rightsquigarrow \quad \begin{array}{l} \text{let Reg}[g] \leftarrow \neg\text{Reg}[j] \\ \text{let Reg}[h] \leftarrow \neg\text{Reg}[k] \\ \text{let Reg}[i] \leftarrow \text{Reg}[g] \wedge \text{Reg}[h] \\ \text{let Reg}[i] \leftarrow \neg\text{Reg}[i] \end{array}$$

This also works with constants standing in for Reg$[j]$ and/or Reg$[k]$.

Then, update the target line numbers of all branch instructions, and apply Theorem 1.2.

The proof of Theorem 1.2 reintroduces bitwise-or instructions when constructing constants, but this can be sidestepped by replacing "let Reg[0] $\leftarrow$ Reg[0]$\vee$1" with "let Reg[0] $\leftarrow$ Reg[0]+1" since the least significant bit of Reg[0] is guaranteed to be 0 here.

**Exercise 1.13.c:** Note that $\neg x = x \oplus (0-1)$ for all $x \in \{0,1\}^W$ since $0-1$ is the all-1 word. Employing a fresh "register $h$":

$$\begin{array}{l} \text{let Reg}[i] \leftarrow \neg\text{Reg}[j] \end{array} \quad \rightsquigarrow \quad \begin{array}{l} \text{let Reg}[h] \leftarrow 0-1 \\ \text{let Reg}[i] \leftarrow \text{Reg}[j] \oplus \text{Reg}[h] \end{array}$$

This also works with a constant standing in for Reg$[j]$.

Then, update the target line numbers of all branch instructions, and apply Theorem 1.2.

The proof of Theorem 1.2 reintroduces bitwise-not instructions when doing the masking, but this can be sidestepped by replacing "let Reg[1] $\leftarrow \neg\, 0$" with "let Reg[1] $\leftarrow 0-1$" here.

**Exercise 1.14:** Here's pseudocode for the new program:

run the original program, ignoring all write instructions, and using an extra register $m$
    to remember the largest output address that would be written to
    (in place of "write Out[Reg[$j$]] ← Reg[$i$]": if Reg[$j$] > $m$ then let $m$ ← Reg[$j$])
for $a \leftarrow 0, 1, \ldots, m$:
    reset the registers and work segment for the original program
    run the original program, ignoring all write instructions, and using an extra register $v$
        (reinitialized to 0) to remember the latest value that would be written to Out[$a$]
        (in place of "write Out[Reg[$j$]] ← Reg[$i$]": if Reg[$j$] = $a$ then let $v$ ← Reg[$i$])
    write $v$

**Exercise 1.15:** Let $\mathsf{P}'$ and $\mathsf{L}'$ be the classes of all total problems in $\mathsf{P}$ and $\mathsf{L}$, respectively. We show that $\mathsf{P} = \mathsf{L}$ iff $\mathsf{P}' = \mathsf{L}'$:

$\Rightarrow$: This is trivial.

$\Leftarrow$: Assume $\mathsf{P}' = \mathsf{L}'$. To see that $\mathsf{P} = \mathsf{L}$, consider any promise problem $A \in \mathsf{P}$. Say program $\Pi$ solves $A$ with time efficiency $\leq cN^d$ for some integers $c, d$. On invalid inputs to $A$, $\Pi$ might take more than $cN^d$ steps. Define a program $\Pi'$ that runs $\Pi$ but counts how many steps it takes, and rejects if it takes more than $cN^d$ steps. Since $\Pi'$ runs in poly time on all inputs (including invalid inputs to $A$), it solves some total problem $A' \in \mathsf{P}'$ that's consistent with $A$. By the assumption that $\mathsf{P}' = \mathsf{L}'$, we have $A' \in \mathsf{L}'$. Thus some log-space program solves $A'$ and therefore solves $A$, so $A \in \mathsf{L}$.

**Exercise 1.16:** This oracle reduction from Directed Min Path Height to Directed Reachability does binary search over the possible path heights:

```
on input (G, s, t):
    let G' be G but without height labels on the edges
    query the oracle on input (G', s, t)
    if the oracle rejected: halt and report that t is not reachable from s
    initialize lo ← 0 and hi ← m
    while lo < hi:
        let mid ← ⌊(lo + hi)/2⌋
        let G'_mid be G but with all edges of height > mid removed, and without height labels
        query the oracle on input (G'_mid, s, t)
        if the oracle accepted: update hi ← mid
        if the oracle rejected: update lo ← mid + 1
    output lo
```

Assuming $t$ is reachable from $s$, the reduction maintains the loop invariant that the minimum height of any path from $s$ to $t$ is $\geq lo$ and $\leq hi$, because the oracle accepts iff $G$ has a path from $s$ to $t$ of height $\leq mid$. The loop terminates since $hi - lo$ decreases in each iteration. Thus the output is correct since the invariant holds at the end when $lo = hi$.

In fact, the loop terminates within $O(\log m) = O(\log N)$ iterations since $hi - lo$ decreases by about a factor of 2 in each iteration. Each iteration of the reduction takes $O(N)$ time. Using BFS to implement the oracle, each query (which has size $\leq N$) gets answered in $O(N)$ time. Thus the overall time efficiency is $O(N \log N)$, with word size $O(\log N)$.

**Exercise 1.17:** This reduction program uses register aliases $N$, $n$, $m$, $i$, $j$, *xbit*, *ybit*, *qbit*, and *ptr*. The word size is $\lceil \log(N+1) \rceil$ since no register ever exceeds $N$ and there is a 2-bit constant.

```
 0. let n ← ⌊N/2⌋
 1. let m ← N − 1
 2. write-query n
 3. write-query m
 4. if j ≥ n then goto line 20
 5.     let i ← 0
 6.     if i ≥ m then goto line 18
 7.         let qbit ← 0
 8.         if i < j then goto line 15
 9.         let ptr ← i − j
10.         if ptr ≥ n then goto line 15
11.         read xbit ← In[ptr]
12.         let ptr ← n + j
13.         read ybit ← In[ptr]
14.         let qbit ← xbit · ybit
15.         write-query qbit
16.         let i ← i + 1
17.         goto line 6
18.     let j ← j + 1
19.     goto line 4
20. query
```

**Exercise 1.18.a:** Assume $A \leq_p^o B$ and $B \in \mathsf{EXP}$. Consider the algorithm that runs the reduction and uses $B$'s exponential-time algorithm to answer each oracle query. By definition, this algorithm solves $A$. Let $N$ denote $A$'s input size. If the reduction produces a query of size $M$, $B$'s algorithm takes $2^{\text{poly}\,M}$ time to answer the query. Since the reduction makes only $\text{poly}\,N$ many queries, each of $\text{poly}\,N$ size, the time efficiency of $A$'s algorithm is:

$$\underbrace{\text{poly}\,N}_{\substack{\text{time of}\\ \text{reduction}}} + \underbrace{\text{poly}\,N}_{\substack{\text{number}\\ \text{of queries}}} \cdot \underbrace{2^{\text{poly}}}_{\substack{\text{time of } B\text{'s}\\ \text{algorithm}}} \Big( \underbrace{\text{poly}\,N}_{\substack{\text{query}\\ \text{size}}} \Big) = 2^{\text{poly}\,N}$$

Thus $A \in \mathsf{EXP}$. Formally, this can be seen by the same proof as Lemma 1.12. The erase phases do not dominate the running time, and word size $\text{poly}\,N$ should be used to accommodate the largest word size needed by $B$'s algorithm on any of the oracle queries.

**Exercise 1.18.b:** Assume $A \leq_{qp}^{o} B$ and $B \in \mathsf{QUASIP}$. Consider the algorithm that runs the reduction and uses $B$'s quasi-polynomial-time algorithm to answer each oracle query. By definition, this algorithm solves $A$. Let $N$ denote $A$'s input size. If the reduction produces a query of size $M$, $B$'s algorithm takes $2^{\text{polylog}\, M}$ time to answer the query. Since the reduction makes only $2^{\text{polylog}\, N}$ many queries, each of $2^{\text{polylog}\, N}$ size, the time efficiency of $A$'s algorithm is:

$$\underbrace{2^{\text{polylog}\, N}}_{\substack{\text{time of}\\ \text{reduction}}} + \underbrace{2^{\text{polylog}\, N}}_{\substack{\text{number}\\ \text{of queries}}} \cdot \underbrace{2^{\text{poly}(\log(\underbrace{2^{\text{polylog}\, N}}_{\substack{\text{query}\\ \text{size}}}))}}_{\substack{\text{time of } B\text{'s}\\ \text{algorithm}}} = 2^{\text{polylog}\, N} + 2^{\text{polylog}\, N + \text{poly}(\text{polylog}\, N)} = 2^{\text{polylog}\, N}$$

Thus $A \in \mathsf{QUASIP}$. Formally, this can be seen by the same proof as Lemma 1.12. The erase phases do not dominate the running time, and word size $\text{polylog}(2^{\text{polylog}\, N}) = \text{polylog}\, N$ should be used to accommodate the largest word size needed by $B$'s algorithm on any of the oracle queries.

**Exercise 1.19.a:** Let $u_1, \ldots, u_n$ be $G$'s nodes. Observe that $G$ is connected iff for every $i \in \{2, \ldots, n\}$, $u_i$ is reachable from $u_1$:

$\Rightarrow$: This is immediate from the definition of "connected."

$\Leftarrow$: Assume that for every $i \in \{2, \ldots, n\}$, $u_i$ is reachable from $u_1$. To see that $G$ is connected, consider any two nodes $u_j$ and $u_k$. If $j = 1$ or $k = 1$ then there is a walk between $u_j$ and $u_k$ by assumption. Otherwise, there is a walk from $u_j$ to $u_1$ and a walk from $u_1$ to $u_k$ by assumption, and concatenating these yields a walk from $u_j$ to $u_k$. Thus $u_k$ is reachable from $u_j$.

Hence, the following oracle reduction from Undirected Connectivity to Undirected Reachability is correct:

```
on input G with nodes u_1, ..., u_n:
    for i ← 2, ..., n:
        query the oracle on input (G, u_1, u_i)
        if the oracle rejected: reject
    accept
```

**Exercise 1.19.b:** Map $G$ (with nodes $u_1, \ldots, u_n$) to $G'$ where:

- $G'$ has $n-1$ copies of $G$.
- For each $i \in \{2, \ldots, n\}$, $G'$ has an edge between $u_i$ in the $(i-1)^{\text{st}}$ copy and $u_1$ in the $i^{\text{th}}$ copy.
- $s$ is $u_1$ in the $1^{\text{st}}$ copy.
- $t$ is $u_n$ in the $(n-1)^{\text{st}}$ copy.

This reduction only needs log space, and it's correct because $G$ is connected iff for every $i \in \{2, \ldots, n\}$, $u_i$ is reachable from $u_1$ in $G$ (Exercise 1.19.a) iff $t$ is reachable from $s$ in $G'$:

$\Rightarrow$: Assume that for every $i \in \{2, \ldots, n\}$, $u_i$ is reachable from $u_1$ in $G$. Then $t$ is reachable from $s$ in $G'$ by a path that goes from $u_1$ to $u_2$ in the $1^{\text{st}}$ copy, then takes the edge to $u_1$ in the $2^{\text{nd}}$ copy, then goes from $u_1$ to $u_3$ in the $2^{\text{nd}}$ copy, then takes the edge to $u_1$ in the $3^{\text{rd}}$ copy, and so on.

$\Leftarrow$: Any path from $s$ to $t$ in $G'$ must have the form described in the previous paragraph, so its segments within the individual copies must be paths from $u_1$ to $u_i$ for each $i \in \{2, \ldots, n\}$.

**Exercise 1.20.a:** The idea is to have $\Pi_3$ run $\Pi_1$ and $\Pi_2$ "in parallel," alternating steps between the two (a step of $\Pi_1$, a step of $\Pi_2$, a step of $\Pi_1$, a step of $\Pi_2$, and so on) until one of them halts. Thus if $\Pi_1(x)$ takes $t$ steps and $\Pi_2(x)$ takes $> t$ steps, then $\Pi_3(x)$ will run $t$ steps of $\Pi_1(x)$ and at most $t$ steps of $\Pi_2(x)$, and thus have only $2t$ running time (plus some bookkeeping overhead), and similarly if $\Pi_2(x)$ finishes before $\Pi_1(x)$.

More formally: $\Pi_3$ has two work segments, $\text{Mem}_1$ for $\Pi_1$ and $\text{Mem}_2$ for $\Pi_2$ (which is OK by Lemma 1.5), and has sets of registers $\text{Reg}_1$ for $\Pi_1$ and $\text{Reg}_2$ for $\Pi_2$. If $\Pi_1$ has $L_1$ many lines and $\Pi_2$ has $L_2$ many lines, then $\Pi_3$ has $4 \cdot L_1 \cdot L_2$ many lines. For each tuple $(i_1, i_2, j) \in \{0, \ldots, L_1 - 1\} \times \{0, \ldots, L_2 - 1\} \times \{1, 2\}$—representing that $\Pi_1$ should execute line $i_1$ next, and $\Pi_2$ should execute line $i_2$ next, and it's $\Pi_j$'s turn now—$\Pi_3$ has a pair of instructions: The first instruction in the pair is the instruction from line $i_j$ of $\Pi_j$ (updated to use $\text{Mem}_j$ and $\text{Reg}_j$ and modified target line number if it's a branch). The second instruction in the pair is an unconditional branch to the first instruction in the pair corresponding to $(i_1 + 1, i_2, 2)$ if $j = 1$, or to $(i_1, i_2 + 1, 1)$ if $j = 2$ (or we just omit the unconditional branch if it would go to a nonexistent line number with $i_1 + 1 \geq L_1$ or $i_2 + 1 \geq L_2$).

**Exercise 1.20.b:** The idea is to have $\Pi_4$ run $\Pi_1$ and $\Pi_2$ "in parallel," but instead of strictly alternating between them, $\Pi_4$ takes a step of whichever one will have used less work memory so far. Thus if $\Pi_1(x)$ uses $s$ words (the first $s$ words of its work segment) and $\Pi_2(x)$ uses $> s$ words, then $\Pi_4(x)$ will use $s$ words for $\Pi_1(x)$ and at most $s$ words for $\Pi_2(x)$, and thus use only $(2s + O(1))W$ memory space, and similarly if $\Pi_2(x)$ uses less work space than $\Pi_1(x)$.

More formally: $\Pi_4$ has two work segments, $\text{Mem}_1$ for $\Pi_1$ and $\text{Mem}_2$ for $\Pi_2$ (which is OK by Lemma 1.5), and has sets of registers $\text{Reg}_1$ for $\Pi_1$ and $\text{Reg}_2$ for $\Pi_2$. Also, $\Pi_4$ has a register nicknamed $a$ that holds what will be the highest work segment address accessed so far by $\Pi_1$ after executing the next step of $\Pi_1$. Also, $\Pi_4$ has a similar register $b$ for $\Pi_2$. Suppose $\Pi_1$ has $L_1$ many lines and $\Pi_2$ has $L_2$ many lines. For each tuple $(i_1, i_2) \in \{0, \ldots, L_1 - 1\} \times \{0, \ldots, L_2 - 1\}$—representing that $\Pi_1$ should execute line $i_1$ next, and $\Pi_2$ should execute line $i_2$ next—$\Pi_4$ has a bundle of instructions:

- If line $i_1$ of $\Pi_1$ is a load or store, the bundle updates $a \leftarrow \max(a, \text{address of load/store})$.
- If line $i_2$ of $\Pi_2$ is a load or store, the bundle updates $b \leftarrow \max(b, \text{address of load/store})$.
- If $a \leq b$, the bundle executes line $i_1$ of $\Pi_1$ (updated to use $\text{Mem}_1$ and $\text{Reg}_1$ and modified target line number if it's a branch), then branches to the bundle corresponding to $(i_1 + 1, i_2)$ (if a branch wasn't already taken).
- If $a > b$, the bundle executes line $i_2$ of $\Pi_2$ (updated to use $\text{Mem}_2$ and $\text{Reg}_2$ and modified target line number if it's a branch), then branches to the bundle corresponding to $(i_1, i_2 + 1)$ (if a branch wasn't already taken).

**Exercise 1.20.c:** If $\Pi_1$ and $\Pi_2$ solve a search problem, they might not output the same solution as each other, so we must not mix their outputs together. We update $\Pi_3$ and $\Pi_4$ to first determine which of $\Pi_1(x)$ or $\Pi_2(x)$ uses less time (for $\Pi_3$) or space (for $\Pi_4$), by running the entire simulations as above but ignoring all write instructions, then reset all the registers and work memory and just run whichever of $\Pi_1(x)$ or $\Pi_2(x)$ was more efficient, executing the write instructions this time.

**Exercise 2.1:** Consider this algorithm for Rock Scissors Paper Coloring:

```
let G′ be the undirected version of G, with arrow directions erased
for each connected component of G′:
    pick an arbitrary node s in the component and assign it an arbitrary color, say c(s) = rock
    run BFS in G′ starting from s, assigning colors to nodes as they're discovered:
        whenever a node u causes the discovery of a node v:
            if the corresponding edge in G is (u, v):
                let c(v) be the unique color beaten by c(u)
            if the corresponding edge in G is (v, u):
                let c(v) be the unique color that beats c(u)
if c(u) beats c(v) for every edge (u, v) of G: accept
else: reject
```

Say a coloring is *winning* iff $c(u)$ beats $c(v)$ for every edge $(u, v)$ of $G$. If the algorithm accepts, then it has found a winning coloring. Conversely, suppose $G$ has a winning coloring. To see that the algorithm accepts, we claim it maintains the invariant that $G$ has a winning coloring that agrees with the algorithm's color assignment $c$ to the nodes discovered so far. To see that the invariant is maintained when we assign rock to the first node $s$ in a connected component of $G'$:

- If a winning coloring assigns scissors to $s$, we can obtain another winning coloring (that agrees with $c$ so far) by changing the colors in that component: scissors to rock, and rock to paper, and paper to scissors.
- If a winning coloring assigns paper to $s$, we can obtain another winning coloring (that agrees with $c$ so far) by changing the colors in that component: paper to rock, and rock to scissors, and scissors to paper.

During BFS, if the invariant holds before we color $v$, it must continue to hold after we color $v$, because in every winning coloring of $G$, the color of either endpoint of an edge is uniquely determined by the color of the other endpoint. At termination, the invariant guarantees that the algorithm's final color assignment is winning, so the algorithm accepts.

**Exercise 2.2:**

$S_0 = \{0\}$

$S_1 = S_0 \cup (S_0 + 7) = \{0\} \cup \{7\} = \{0, 7\}$

$S_2 = S_1 \cup (S_1 + 3) = \{0, 7\} \cup \{3, 10\} = \{0, 3, 7, 10\}$

$S_3 = S_2 \cup (S_2 + 4) = \{0, 3, 7, 10\} \cup \{4, 7, 11, 14\} = \{0, 3, 4, 7, 10, 11, 14\}$

$S_4 = S_3 \cup (S_3 + 10) = \{0, 3, 4, 7, 10, 11, 14\} \cup \{10, 13, 14, 17, 20, 21, 24\}$
$= \{0, 3, 4, 7, 10, 11, 13, 14, 17, 20, 21, 24\}$

The output is "reject" since $19 \notin S_4$.

**Exercise 2.3.a:** View Wit[1], . . . , Wit[$n$] as an assignment to the $n$ variables. This verifier program uses word size $O(\log N)$ and registers nicknamed $N$, *ptr*, *next*, $j$, $i$, $b$, $xi$. It uses *ptr* as a pointer into the input. The outer loop iterates over the clauses. The inner loop checks whether a particular clause is satisfied and rejects if not. The inner loop has three iterations (using $j$ as the counter)—one iteration per literal in the clause. It reads $i$ as the index of the literal's variable, and $b$ as the bit indicating whether the literal is positive. Then $xi$ is the bit assigned to the $i^{\text{th}}$ variable. This literal satisfies the clause iff $xi = b$.

```
 0. let ptr ← 2
 1. if ptr = N then goto line 16
 2.     let next ← ptr + 6
 3.     let j ← 0
 4.     if j = 3 then goto line 15
 5.         let j ← j + 1
 6.         read i ← In[ptr]
 7.         let ptr ← ptr + 1
 8.         read b ← In[ptr]
 9.         let ptr ← ptr + 1
10.         read-witness xi ← Wit[i]
11.         if xi > 1 then goto line 15
12.         if xi ≠ b then goto line 4
13.         let ptr ← next
14.         goto line 1
15.     reject
16. accept
```

**Exercise 2.3.b:** For each node index $u \in [n]$, $\text{Wit}[u] = 1$ means $u \in S$, and $\text{Wit}[u] = 0$ means $u \notin S$. This verifier program uses word size $O(\log N)$ and registers nicknamed $N$, $n$, $k$, *uind*, *vind*, *size*, *ptr*, *deg*, *end*. The left column checks that $|S| \geq k$, and the right column checks that $S$ is independent.

```
 0. let ptr ← N − 1
 1. read k ← In[ptr]
 2. read n ← In[0]
 3. if u = n then goto line 9
 4.     let u ← u + 1
 5.     read-witness uind ← Wit[u]
 6.     let size ← size + uind
 7.     if uind ≤ 1 then goto line 3
 8.     reject
 9. if size ≥ k then goto line 11
10. reject
```

```
11. let u ← 0
12. if u = n then goto line 25
13.     let u ← u + 1
14.     read-witness uind ← Wit[u]
15.     if uind = 0 then goto line 12
16.     read ptr ← In[u]
17.     read deg ← In[ptr]
18.     let end ← ptr + deg
19.     if ptr = end then goto line 12
20.         let ptr ← ptr + 1
21.         read v ← In[ptr]
22.         read-witness vind ← Wit[v]
23.         if vind = 0 then goto line 19
24.         reject
25. accept
```

**Exercise 2.3.c:** View Wit[1], ..., Wit[$n$] as the labels of nodes in the order they're visited by some full path from $s$ to $t$. This verifier program uses word size $O(\log N)$ and registers nicknamed $N$, $n$, $i$, $s$, $t$, $u$, $v$, $w$, *uthere*, *ptr*, *outdeg*, *end*. (Here, $w$ refers to a node, not to the whole witness.) In the left column, after checking that the witness list starts with $s$ and ends with $t$, we check that the list only contains valid node labels and that each node appears at most once (and thus exactly once) in the list. For the latter, we let Mem[$u$] be 1 the first time we encounter node label $u \in [n]$ in the witness list, and reject if we encounter $u$ again. In the right column, we check that node Wit[$i$] has an edge to node Wit[$i+1$] for each $i$.

```
 0. read n ← In[0]
 1. let ptr ← N − 2
 2. read s ← In[ptr]
 3. read-witness u ← Wit[1]
 4. if u ≠ s then goto line 33
 5. let ptr ← N − 1
 6. read t ← In[ptr]
 7. read-witness u ← Wit[n]
 8. if u ≠ t then goto line 33
 9. if i = n then goto line 18
10.     let i ← i + 1
11.     read-witness u ← Wit[i]
12.     if u < 1 then goto line 33
13.     if u > n then goto line 33
14.     load uthere ← Mem[u]
15.     if uthere = 1 then goto line 33
16.     store Mem[u] ← 1
17.     goto line 9
18. let i ← 1
19. let u ← s
20. if i = n then goto line 32
21.     let i ← i + 1
22.     read-witness w ← Wit[i]
23.     read ptr ← In[u]
24.     read outdeg ← In[ptr]
25.     let end ← ptr + outdeg
26.     if ptr = end then goto line 33
27.         let ptr ← ptr + 1
28.         read v ← In[ptr]
29.         if v ≠ w then goto line 26
30.         let u ← w
31.         goto line 20
32. accept
33. reject
```

**Exercise 2.4:** Assume $A \leq_p^m B$ by reduction $F$, and $B \in \mathsf{NP}$ by verifier $V$. Then $A \in \mathsf{NP}$ by a verifier $V'$ that, on input $x$, computes $F(x)$ and then runs $V$ on input $F(x)$ using the same purported witness $w$. This is correct because:

$$A(x) = 1 \iff B(F(x)) = 1 \iff \big(\exists w : \ V(F(x); w) \text{ accepts}\big) \iff \big(\exists w : \ V'(x; w) \text{ accepts}\big)$$

Also, $V'$ is poly-time since if $x$ has size $N$, then computing $F(x)$ takes $\text{poly}\, N$ time, and $F(x)$ has $\text{poly}\, N$ size, and running $V$ on input $F(x)$ takes $\text{poly}(\text{poly}\, N) = \text{poly}\, N$ time. The word size of $V$ on input $x$ should be at least the word size of $V'$ on input $F(x)$.

**Exercise 2.5.a:** Let $V_A$ and $V_B$ be poly-time verifiers for $A$ and $B$. We may assume $V_A$ and $V_B$ have the same word size. Consider a verifier $V$ that, on input $x$ and purported witness $w$, runs $V_A$ and $V_B$ on the same $x$ and $w$, and accepts iff at least one of them accepts. Succinctly, $V(x;w) = V_A(x;w) \vee V_B(x;w)$. Then $V$ is poly-time and is a correct verifier for $A \vee B$ because:

$$\begin{aligned}
(A \vee B)(x) = 1 &\iff (A(x) = 1) \vee (B(x) = 1) \\
&\iff \big(\exists w : \ V_A(x;w) \text{ accepts}\big) \vee \big(\exists w : \ V_B(x;w) \text{ accepts}\big) \\
&\iff \big(\exists w : \ (V_A(x;w) \text{ accepts}) \vee (V_B(x;w) \text{ accepts})\big) \\
&\iff \big(\exists w : \ V(x;w) \text{ accepts}\big)
\end{aligned}$$

Thus $A \vee B \in \mathsf{NP}$.

**Exercise 2.5.b:** Let $V_A$ and $V_B$ be poly-time verifiers for $A$ and $B$. We may assume $V_A$ and $V_B$ have the same word size. Consider a verifier $V$ that, on input $x$ and purported witness $(w_A, w_B)$, runs $V_A$ and $V_B$ on the same $x$ with $w_A$ for $V_A$ and $w_B$ for $V_B$, and accepts iff both of them accept. Succinctly, $V(x; w_A, w_B) = V_A(x; w_A) \wedge V_B(x; w_B)$. Then $V$ is poly-time and is a correct verifier for $A \wedge B$ because:

$$\begin{aligned}
(A \wedge B)(x) = 1 &\iff (A(x) = 1) \wedge (B(x) = 1) \\
&\iff \big(\exists w : \ V_A(x; w) \text{ accepts}\big) \wedge \big(\exists w : \ V_B(x; w) \text{ accepts}\big) \\
&\iff \big(\exists w_A, w_B : \ (V_A(x; w_A) \text{ accepts}) \wedge (V_B(x; w_B) \text{ accepts})\big) \\
&\iff \big(\exists w_A, w_B : \ V(x; w_A, w_B) \text{ accepts}\big)
\end{aligned}$$

Thus $A \wedge B \in \mathsf{NP}$.

**Exercise 2.6:** Let $V$ be such a verifier. We modify $V$ into a verifier $V'$ that is correct for $A$, takes $O(T)$ execution steps, accesses only the first $O(T)$ words of its witness segment, and has $O(\log \max(T, N))$-bounded word size, thereby showing $A \in \mathsf{NTIME}[T]$.

$V'$ expects the witness to be rearranged so the word at the first address $V$ would access is now at address 0, and the word at the second unique address $V$ would access (not counting re-reads of the first address) is now at address 1, and the word at the third unique address $V$ would access (not counting re-reads of the first two addresses) is now at address 2, and so on. Whenever $V$ reads a witness segment address it hasn't read before, $V'$ instead reads from the next fresh witness segment address and caches the word in its work memory. Whenever $V$ re-reads a witness segment address, $V'$ instead looks up the corresponding word in the cache.

$V'$ uses three work segments (which is OK by Lemma 1.5, which also works for verifiers):

- $\text{Mem}_1$ corresponds to $V$'s work segment.
- $\text{Mem}_2$ records which witness segment addresses $V$ has already accessed: $\text{Mem}_2[a] = 1$ if $V$ has already read from address $a$, and $\text{Mem}_2[a] = 0$ otherwise.
- $\text{Mem}_3$ is the cache of witness segment words: $\text{Mem}_3[a]$ is the word $V$ would have read at address $a$ (if $V$ read from address $a$ yet).

$V'$ has the same registers as $V$ and a register nicknamed *wsize* to remember the number of unique witness segment addresses $V$ has accessed so far. The code of $V'$ is the code of $V$ but using $\text{Mem}_1$ in place of Mem and with the following substitution, which is simple to de-sugar:

read-witness Reg[$i$] ← Wit[Reg[$j$]] ⇝

```
if Mem2[Reg[j]] = 1:
    load Reg[i] ← Mem3[Reg[j]]
else:
    read-witness Reg[i] ← Wit[wsize]
    let wsize ← wsize + 1
    store Mem3[Reg[j]] ← Reg[i]
    store Mem2[Reg[j]] ← 1
```

($V'$ needs another register to load $\text{Mem}_2[\text{Reg}[j]]$ into.) Here's the efficiency analysis:

- $V'$ takes $O(T)$ steps since each step of $V$ corresponds to $O(1)$ steps of $V'$.
- $V'$ accesses only the first $O(T)$ words of its witness segment, because it reads them in sequence.
- $V'$ uses the same word size as $V$. Note that *wsize* fits in this word size since *wsize* $\leq$ the largest address accessed by $V$.

Now, we argue that $V'$ is correct since $V$ is correct:

$$\big(\exists w : \ V(x; w) \text{ accepts}\big) \iff \big(\exists w' : \ V'(x; w') \text{ accepts}\big)$$

$\Rightarrow$: Suppose $V(x; w)$ accepts. Let $a_0, a_1, a_2, \ldots$ be the unique witness segment addresses accessed by $V(x; w)$, in the order they are first accessed. Define $w'$ by $w'_i = w_{a_i}$ for each $i \geq 0$. The first time $V(x; w)$ reads $w_{a_i}$, $V'(x; w')$ will read $w'_i$, and each subsequent time $V(x; w)$ reads $w_{a_i}$, $V'(x; w')$ will read the cached copy. Thus $V'(x; w')$ faithfully runs $V(x; w)$ and outputs the same thing (accept).

$\Leftarrow$: Suppose $V'(x;w')$ accepts. Let $a_0, a_1, a_2, \ldots$ be such that when $V'(x;w')$ reads $w'_i$, $V$ is reading address $a_i$ of its witness segment. Define $w$ by $w_{a_i} = w'_i$ for each $i \geq 0$ (and the other words of $w$ can be 0). Thus $V'(x;w')$ faithfully runs $V(x;w)$, so $V(x;w)$ accepts since $V'(x;w')$ does.

**Exercise 2.7.a:** The general DNF construction from Lemma 2.9 yields:

$$f(x) = \bigvee_{i=0}^{n} (\overline{x_1} \wedge \cdots \wedge \overline{x_i} \wedge x_{i+1} \wedge \cdots \wedge x_n)$$

**Exercise 2.7.b:** The bits of $x$ are sorted iff there is no consecutive 10 in $x$:

$$f(x) = \bigwedge_{i=1}^{n-1} (\overline{x_i} \vee x_{i+1})$$

**Exercise 2.8.a:** There are $2^n - \binom{n}{2} - n - 1$ assignments on which $f$ evaluates to 1, so the general DNF construction from Lemma 2.9 has exponential size, and isn't monotone anyway. Instead, we design a monotone DNF expressing that there exist three variables that are assigned 1:

$$f(x) = \bigvee_{i<j<k} (x_i \wedge x_j \wedge x_k)$$

This has $\binom{n}{3}$ terms, each of width 3, so the size is $\binom{n}{3} \cdot 3 = n(n-1)(n-2)/2 \leq n^3$.

**Exercise 2.8.b:** Lemma 2.9 yields a CNF with $\binom{n}{2} + n + 1$ clauses, each of width $n$, but it isn't monotone. Instead, we design a monotone CNF expressing that for every two variables, there exists a different variable that is assigned 1:

$$f(x) = \bigwedge_{i<j} \left( \bigvee_{k \notin \{i,j\}} x_k \right)$$

This has $\binom{n}{2}$ clauses, each of width $n-2$, so the size is $\binom{n}{2} \cdot (n-2) = n(n-1)(n-2)/2 \leq n^3$.

**Exercise 2.9:** One way to phrase the proof is with the probabilistic method. Suppose $\varphi$ is a $k$-CNF with fewer than $2^k$ clauses. Pick a uniformly random assignment. For each clause $C$ of $\varphi$:

$$\mathbf{Pr}[C \text{ is unsatisfied}] = \prod_{i=1}^{k} \mathbf{Pr}[i^{\text{th}} \text{ literal of } C \text{ is unsatisfied}] = \prod_{i=1}^{k} 1/2 = 2^{-k}$$

By a union bound:

$$\begin{aligned}
\mathbf{Pr}[\varphi \text{ is unsatisfied}] &= \mathbf{Pr}[\exists \text{ unsatisfied clause in } \varphi] \\
&\leq \sum_{\text{clause } C \text{ in } \varphi} \mathbf{Pr}[C \text{ is unsatisfied}] \\
&= (\text{number of clauses in } \varphi) \cdot 2^{-k} \\
&< 2^k \cdot 2^{-k} \\
&= 1
\end{aligned}$$

Since $\mathbf{Pr}[\varphi \text{ is satisfied}] > 0$, there exists a satisfying assignment for $\varphi$.

**Exercise 2.10.a:** This is NP-complete. It's in NP since a poly-time verifier can view the witness as a tuple of three assignments and check that they're distinct and each satisfy $\varphi$. We reduce SAT to this problem by mapping $\varphi$ to a CNF $\varphi'$ that's $\varphi$ but also has a clause $(x_{n+1} \vee x_{n+2})$ where $x_{n+1}$ and $x_{n+2}$ are fresh variables (that aren't in $\varphi$). To show this poly-time mapping reduction is correct, we argue that $\varphi$ has a satisfying assignment iff $\varphi'$ has at least three satisfying assignments:

$\Rightarrow$: If $\varphi$ has a satisfying assignment, then the same assignment together with $x_{n+1}x_{n+2} = 10$ satisfies $\varphi'$, and the same assignment together with $x_{n+1}x_{n+2} = 01$ satisfies $\varphi'$, and the same assignment together with $x_{n+1}x_{n+2} = 11$ satisfies $\varphi'$.

$\Leftarrow$: If $\varphi'$ has at least three satisfying assignments, then any such assignment (ignoring $x_{n+1}$ and $x_{n+2}$) also satisfies $\varphi$.

**Exercise 2.10.b:** This is in P by an algorithm that accepts iff for each variable $x_i$, either $\varphi$ has no $x_i$ literals or $\varphi$ has no $\overline{x_i}$ literals. To show this poly-time algorithm is correct, we argue that it accepts iff there exists an assignment that satisfies none of $\varphi$'s clauses:

$\Rightarrow$: Suppose for each variable $x_i$, either there's no $x_i$ or there's no $\overline{x_i}$. Assign $x_i = 1$ if there's no $x_i$, and assign $x_i = 0$ if there's no $\overline{x_i}$. Then every literal in every clause of $\varphi$ is unsatisfied.

$\Leftarrow$: Suppose some assignment satisfies none of $\varphi$'s clauses. For each variable $x_i$, if $x_i = 1$ then there's no $x_i$ (otherwise a clause containing $x_i$ would be satisfied), and if $x_i = 0$ then there's no $\overline{x_i}$ (otherwise a clause containing $\overline{x_i}$ would be satisfied).

**Exercise 2.10.c:** This is NP-complete. It's in NP since a poly-time verifier can view the witness as an assignment and check that it satisfies all but one of $\varphi$'s clauses. We reduce SAT to this problem by mapping $\varphi$ to a CNF $\varphi'$ that's $\varphi$ but also has an empty clause (). To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ has an assignment that satisfies all but one clause:

$\Rightarrow$: If $\varphi$ has a satisfying assignment, then the same assignment satisfies all clauses of $\varphi'$ except the new empty clause.

$\Leftarrow$: If $\varphi'$ has an assignment that satisfies all but one clause, then the one unsatisfied clause must be the empty one, so the same assignment satisfies $\varphi$.

For someone who doesn't like allowing empty clauses, we could instead put a pair of clauses $(x_i) \wedge (\overline{x_i})$ in $\varphi'$, where $x_i$ is any variable (it doesn't matter which variable).

**Exercise 2.10.d:** This is in P by an algorithm that accepts iff there exist two clauses in $\varphi$ that consist of different sets of literals. To show this poly-time algorithm is correct, we argue that it accepts iff there exists an assignment that satisfies some but not all of $\varphi$'s clauses:

$\Rightarrow$: Suppose clauses $C_1$ and $C_2$ consist of different sets of literals. This means one of them has a literal that the other doesn't—say, $C_1$ has $x_i$ but $C_2$ doesn't have $x_i$. Then we can assign the variables that appear in $C_2$ so $C_2$ is unsatisfied, but assign $x_i = 1$ to satisfy $C_1$. ($C_2$ might have $\overline{x_i}$, in which case unsatisfying $C_2$ already entails $x_i = 1$, which is fine.)

$\Leftarrow$: Suppose there exists an assignment that satisfies some clause $C_1$—say, $x_i = 1$ and $C_1$ contains $x_i$—but doesn't satisfy some clause $C_2$. Then $C_2$ must not have the literal $x_i$ (otherwise $C_2$ would also be satisfied by $x_i = 1$), so these two clauses consist of different sets of literals.

**Exercise 2.10.e:** This is NP-complete. It's in NP since a poly-time verifier can view the witness as an assignment and check that it satisfies $\varphi$ and that $x_i x_{i+1} \neq 11$ for all $i < n$. We reduce SAT to this problem by mapping $\varphi(x_1 \cdots x_n)$ to a CNF $\varphi'(x_1 x_2 \cdots x_{2n-1} x_{2n})$ that's $\varphi$ but with $x_i$ changed to $x_{2i}$ everywhere. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ has a satisfying assignment such that $x_i x_{i+1} \neq 11$ for all $i < 2n$:

$\Rightarrow$: If $\varphi$ has a satisfying assignment to $x_1 x_2 \cdots x_n$, then the same assignment to $x_2 x_4 \cdots x_{2n}$, along with $x_1 = x_3 = \cdots = x_{2n-1} = 0$, satisfies $\varphi'$, and for every pair of consecutive variables, at least one of them (the odd-index one) is not assigned 1.

$\Leftarrow$: If $\varphi'$ has a satisfying assignment (such that $x_i x_{i+1} \neq 11$ for all $i < 2n$), then the assignment to $x_2 x_4 \cdots x_{2n}$ is an assignment to $x_1 x_2 \cdots x_n$ that satisfies $\varphi$.

Someone who's bothered by the variables $x_1 x_3 \cdots x_{2n-1}$ not appearing in $\varphi'$ could add clauses $(\overline{x_1}) \wedge (\overline{x_3}) \wedge \cdots \wedge (\overline{x_{2n-1}})$.

**Exercise 2.10.f:** This is in P by an algorithm that accepts iff $\varphi$ is satisfied by at least one of the following $n+1$ many assignments: $000\cdots000$, $000\cdots001$, $000\cdots011$, …, $001\cdots111$, $011\cdots111$, $111\cdots111$. That is, for each $i \in \{0,\ldots,n\}$, we consider the assignment with $x_1 = x_2 = \cdots = x_i = 0$ and $x_{i+1} = x_{i+2} = \cdots = x_n = 1$. Since these are the only assignments such that $x_i x_{i+1} \neq 10$ for all $i < n$, the algorithm is correct by definition. It runs in $O(N^2)$ time since there are $n+1 \leq O(N)$ many assignments to try, and it takes $O(N)$ time to evaluate $\varphi$ on any such assignment.

**Exercise 2.11.a:** This is in NP since SAT is. We reduce SAT to this problem by mapping $\varphi(x_1 \cdots x_n)$ to a CNF $\varphi'(x_1 \cdots x_n, y_1 \cdots y_n)$ that's $\varphi$ but with each occurrence of $\overline{x_i}$ changed to $y_i$, and with new clauses $(x_i \vee y_i) \wedge (\overline{x_i} \vee \overline{y_i})$ for each $i$. The modified original clauses only have positive literals, and each new clause has only positive literals $(x_i \vee y_i)$ or only negative literals $(\overline{x_i} \vee \overline{y_i})$. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ is satisfiable:

$\Rightarrow$: If $\varphi$ has a satisfying assignment to $x_1 \cdots x_n$, then $\varphi'$ is satisfied by this assignment along with $y_1 \cdots y_n =$ the bitwise negation of $x_1 \cdots x_n$. The modified original clauses are satisfied like in $\varphi$, and $(x_i \vee y_i)$ is satisfied since one of $x_i, y_i$ is 1, and $(\overline{x_i} \vee \overline{y_i})$ is satisfied since the other of $x_i, y_i$ is 0.

$\Leftarrow$: If $\varphi'$ has a satisfying assignment to $x_1 \cdots x_n, y_1 \cdots y_n$, then $\varphi$ is satisfied by the $x_1 \cdots x_n$ part of this assignment. Since each $(x_i \vee y_i) \wedge (\overline{x_i} \vee \overline{y_i})$ is satisfied, $x_i$ and $y_i$ must be assigned opposite bits, so $y_i$ is equivalent to $\overline{x_i}$. Thus the original clauses are satisfied like their counterparts in $\varphi'$.

**Exercise 2.11.b:** This is in NP since SAT is. We reduce SAT to this problem by mapping $\varphi(x_1 \cdots x_n)$ to a CNF $\varphi'$ defined as follows: For each $i$, define $o_i$ as the number of occurrences (positive or negative) of $x_i$ in $\varphi$. Then $\varphi'$ has variables $x_{i,1}x_{i,2}\cdots x_{i,o_i}$ for each $i$. In $\varphi'$ we have a copy of $\varphi$ but with the $j^{\text{th}}$ occurrence of $x_i$ changed to $x_{i,j}$ (and if this occurrence happens to be a negative literal $\overline{x_i}$, it becomes $\overline{x_{i,j}}$). So far, every variable occurs in only one clause of $\varphi'$. For each $i$ such that $o_i \geq 2$, we add to $\varphi'$ the clauses $(\overline{x_{i,1}} \vee x_{i,2}) \wedge (\overline{x_{i,2}} \vee x_{i,3}) \wedge \cdots \wedge (\overline{x_{i,o_i-1}} \vee x_{i,o_i}) \wedge (\overline{x_{i,o_i}} \vee x_{i,1})$. These clauses are equivalent to the cycle of implications $x_{i,1} \Rightarrow x_{i,2} \Rightarrow \cdots \Rightarrow x_{i,o_i} \Rightarrow x_{i,1}$, which are satisfied iff $x_{i,1} = x_{i,2} = \cdots = x_{i,o_i}$ (since if one of these variables is assigned 1, the cycle ensures they're all assigned 1). Each variable occurs two more times in these new clauses (or zero more times if $o_i \leq 1$). To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ is satisfiable:

$\Rightarrow$: If $\varphi$ has a satisfying assignment to $x_1 \cdots x_n$, then $\varphi'$ is satisfied by the assignment where $x_{i,1} = x_{i,2} = \cdots = x_{i,o_i} = x_i$ for each $i$. The modified original clauses are satisfied like in $\varphi$, and the new clauses are satisfied since $x_{i,1} = x_{i,2} = \cdots = x_{i,o_i}$ for each $i$.

$\Leftarrow$: If $\varphi'$ has a satisfying assignment, then for each $i$, $x_{i,1}x_{i,2}\cdots x_{i,o_i}$ must all be assigned the same bit, since the new clauses are satisfied. Assigning $x_i$ this common bit (for each $i$), the original clauses of $\varphi$ are satisfied like their counterparts in $\varphi'$.

**Exercise 2.12:** Suppose the algorithm rejects because $x_i$ and $\overline{x_i}$ are reachable from each other. We claim that $\varphi$ is unsatisfiable. Consider any assignment. If $x_i = 0$ and $\overline{x_i} = 1$, then at least one implication on a path from $\overline{x_i}$ to $x_i$ must be unsatisfied, since $1 \Rightarrow 0$ is false. If $x_i = 1$ and $\overline{x_i} = 0$, then at least one implication on a path from $x_i$ to $\overline{x_i}$ must be unsatisfied.

Now, suppose the algorithm doesn't reject. We claim that it produces a consistent assignment—that is, no variable gets assigned both 0 and 1—and that this assignment satisfies $\varphi$.

Let $\ell_i \rightsquigarrow \ell_j$ mean $\ell_j$ is reachable from $\ell_i$ (including the possibility that $\ell_i$ and $\ell_j$ are the same node). A key observation is that by the definition of the implication graph, if $\ell_i \rightsquigarrow \ell_j$ then $\overline{\ell_j} \rightsquigarrow \overline{\ell_i}$.

Suppose for contradiction some variable $x_k$ gets assigned both 0 and 1. There are two cases:

- Suppose $x_k$ is assigned both 0 and 1 during the same outer iteration, associated with $x_i$, say. For one of the literals $\ell_i \in \{x_i, \overline{x_i}\}$, all literals reachable from $\ell_i$ (including $\ell_i$ itself) are assigned 1. Since both $x_k$ and $\overline{x_k}$ are among these literals, that means $\ell_i \rightsquigarrow x_k$ and $x_k \rightsquigarrow \overline{\ell_i}$ (since $\ell_i \rightsquigarrow \overline{x_k}$), so $\ell_i \rightsquigarrow \overline{\ell_i}$.
  - If $\ell_i$ is $\overline{x_i}$ then $\overline{x_i} \rightsquigarrow x_i$ (as we just showed), which means the algorithm actually would have chosen $\ell_i$ to be $x_i$ (by the innermost "if" condition).
  - If $\ell_i$ is $x_i$ then $x_i \rightsquigarrow \overline{x_i}$ (as we just showed) and $\overline{x_i} \rightsquigarrow x_i$ (by the innermost "if" condition), so the algorithm actually would have rejected.
- Suppose $x_k$ gets assigned 0 and assigned 1 during different outer iterations. Say, $\ell_k$ is assigned 1 during the $x_i$ iteration where $\ell_i$ is assigned 1, and later, $\overline{\ell_k}$ is assigned 1 during the $x_j$ iteration where $\ell_j$ is assigned 1. This means $\ell_i \rightsquigarrow \ell_k$ and $\ell_k \rightsquigarrow \overline{\ell_j}$ (since $\ell_j \rightsquigarrow \overline{\ell_k}$), so $\ell_i \rightsquigarrow \overline{\ell_j}$. Thus, $x_j$ would have been assigned during the $x_i$ iteration, so nothing at all would get assigned during the $x_j$ iteration.

Now that we know the algorithm produces a consistent assignment, we show that this assignment satisfies $\varphi$. Consider any clause $(\ell_j \vee \ell_k)$ of $\varphi$. Suppose $x_j$ gets assigned before $x_k$. (The situation is symmetric if $x_k$ gets assigned before $x_j$.) If $\ell_j = 1$ then the clause is satisfied by $\ell_j$. Now, suppose $\overline{\ell_j} = 1$, and say this assignment happens in the $x_i$ iteration with $\ell_i$ getting assigned 1. Then $\ell_i \rightsquigarrow \overline{\ell_j}$, and we know $\overline{\ell_j} \rightarrow \ell_k$ is an edge, so $\ell_i \rightsquigarrow \ell_k$. Thus $\ell_k$ is assigned 1 in the same outer iteration, and the clause is satisfied by $\ell_k$.

**Exercise 2.13.a:** $(\overline{x_2} \vee x_3) \wedge (\overline{x_3}) \wedge (x_2 \vee \overline{x_3}) \wedge ()$

**Exercise 2.13.b:** To obtain $\varphi|_a$, we delete from $\varphi$ each term having at least one literal that evaluates to 0 under $a$ (since the term is already unsatisfied), and we delete from all other terms of $\varphi$ any literals that evaluate to 1 under $a$ (leaving only literals whose variables are $*$s in $a$).

$(x_3) \vee (x_4) \vee (\overline{x_3} \wedge \overline{x_4}) \vee (x_3 \wedge \overline{x_4})$

**Exercise 2.14.a:** To show INDEPENDENT SET SEARCH $\leq_p^o$ INDEPENDENT SET, consider this poly-time reduction:

```
on input (G, k):
    query the oracle on input (G, k), and reject if the oracle rejected
    initialize H ← G and S ← ∅
    while |S| < k:
        pick any node v of H
        let H' ← H but with v (and its edges) removed
        query the oracle on input (H', k − |S|)
        if the oracle accepted: update H ← H'
        if the oracle rejected:
            update H ← H but with v and all of v's neighbors (and their edges) removed
            update S ← S ∪ {v}
    output S
```

Assume $G$ has an independent set of size $k$. The reduction maintains these loop invariants:

*(1)* $H$ is a subgraph of $G$ that contains none of the nodes of $S$.
*(2)* $H$ has an independent set of size $k - |S|$.
*(3)* For every independent set in $H$, its union with $S$ is an independent set in $G$.

These invariants trivially hold before the first iteration. It's straightforward to see that *(1)* is maintained. Inside the loop, there's always an option for $v$, because *(2)* ensures that $H$ has at least $k - |S| > 0$ nodes. If the oracle accepts then *(2)* is maintained, by the definition of INDEPENDENT SET, and *(3)* is maintained since $H$ shrinks and $S$ is unchanged. Now, suppose the oracle rejects. Before updating $H$ and $S$, every (in particular, some) independent set of size $k - |S|$ in $H$ contains $v$. Such an independent set doesn't include any of $v$'s neighbors. Thus if we remove $v$ from such a set, it would be an independent set in the updated $H$ and have size $k - |S|$ (for the updated $|S|$), so *(2)* is maintained. Also, *(3)* is maintained since all of $v$'s neighbors get excluded from $H$.

The loop terminates in poly time since $H$ shrinks in every iteration. At termination we have $|S| = k$, and *(3)* implies that $S$ is an independent set in $G$, so the output is correct.

**Exercise 2.14.b:** To show DIRECTED FULL PATH SEARCH $\leq_p^o$ DIRECTED FULL PATH, consider this poly-time reduction:

```
on input (G, s, t):
    query the oracle on input (G, s, t), and reject if the oracle rejected
    initialize H ← G and let u_1 = s
    for i ← 1, 2, ..., n − 1 where n is the number of nodes:
        for each edge (u_i, v) in H:
            let H' ← H but with u_i (and its edges) removed
            query the oracle on input (H', v, t)
            if the oracle accepted:
                update H ← H' and let u_{i+1} = v
                continue to the next outer iteration
    output u_1 → u_2 → ··· → u_n
```

Assume $G$ has a full path from $s$ to $t$. The reduction maintains these loop invariants:

*(1)* $u_1 \to u_2 \to \cdots \to u_i$ is a path in $G$.
*(2)* $H$ is a subgraph of $G$ containing all nodes except $u_1, u_2, \ldots, u_{i-1}$.
*(3)* $H$ has a full path from $u_i$ to $t$.

These invariants trivially hold before the first iteration. Assume these hold at the start of iteration $i$. We have $u_i \neq t$ by *(2)* and *(3)* and the fact that $i < n$. *(3)* ensures that the oracle will accept for at least one edge $(u_i, v)$, so the reduction will define $u_{i+1}$. Whichever node the reduction defines $u_{i+1}$ to be, *(3)* is maintained, by the definition of DIRECTED FULL PATH. Also, *(2)* is maintained by the definition of $H'$, and *(1)* is maintained since $u_{i+1}$ is in $H$ and is thus not among $u_1, \ldots, u_{i-1}$, by *(2)*. At termination, $u_1 \to \cdots \to u_n$ is a full path in $G$ by *(1)*, and $u_n = t$ since we argued that $u_i \neq t$ for each $i < n$, so the output is correct.

**Exercise 2.14.c:** We show

$$\text{Graph 3-Coloring Search} \leq_p^o \text{Graph 3-Coloring with Some Preassigned Colors} \leq_p^m \text{Graph 3-Coloring}$$

which implies Graph 3-Coloring Search $\leq_p^o$ Graph 3-Coloring (Lemma 1.13).

| Graph 3-Coloring with Some Preassigned Colors |
|---|
| Input: Undirected graph $G$, partial 3-color assignment $c$ to $G$'s nodes |
| Output: Does there exist a proper 3-coloring of $G$ that's consistent with $c$? |

For Graph 3-Coloring with Some Preassigned Colors $\leq_p^m$ Graph 3-Coloring, map $(G, c)$ to $G'$ where $G'$ has a copy of $G$ and a triangle with nodes $\{r, g, b\}$, where:

- $r$ has edges to all nodes $v$ with $c(v) =$ green or $c(v) =$ blue.
- $g$ has edges to all nodes $v$ with $c(v) =$ red or $c(v) =$ blue.
- $b$ has edges to all nodes $v$ with $c(v) =$ red or $c(v) =$ green.

This reduction takes poly time. To show that it's correct, we argue that $G$ has a proper 3-coloring consistent with $c$ iff $G'$ has a proper 3-coloring:

$\Rightarrow$: Assume $G$ has a proper 3-coloring consistent with $c$. Augmenting this by making $r$ red, $g$ green, and $b$ blue, we have a proper 3-coloring of $G'$.

$\Leftarrow$: Assume $G'$ has a proper 3-coloring. Since colors are interchangeable, we may assume $r$ is red, $g$ is green, and $b$ is blue. This proper 3-coloring of $G$ must be consistent with $c$, because:

- Each node $v$ with $c(v) =$ red is adjacent to $g$ and $b$ and is thus red.
- Each node $v$ with $c(v) =$ green is adjacent to $r$ and $b$ and is thus green.
- Each node $v$ with $c(v) =$ blue is adjacent to $r$ and $g$ and is thus blue.

For Graph 3-Coloring Search $\leq_p^o$ Graph 3-Coloring with Some Preassigned Colors, consider this poly-time reduction:

```
on input G:
    initialize c ← empty 3-color assignment to G's nodes
    query the oracle on input (G, c), and reject if the oracle rejected
    for each node v:
        let c(v) ← red
        query the oracle on input (G, c)
        if the oracle rejected:
            update c(v) ← green
            query the oracle on input (G, c)
            if the oracle rejected:
                update c(v) ← blue
    output c
```

Assume $G$ has a proper 3-coloring. The reduction maintains the loop invariant that $G$ has a proper 3-coloring consistent with $c$, at the boundaries of iterations. This invariant is maintained if the oracle accepts the red query or the green query, by the definition of Graph 3-Coloring with Some Preassigned Colors. If the oracle rejects both those queries, then every (in particular, some) proper 3-coloring consistent with the $c$ from the beginning of the iteration colors $v$ blue, so the invariant is maintained in this case as well. At termination, the invariant says $G$ has a proper 3-coloring consistent with $c$, and the only such assignment is $c$ itself, so the output is correct.

**Exercise 2.14.d:** To show SUBSET SUM SEARCH $\leq_p^o$ SUBSET SUM, consider this poly-time reduction:

```
on input (x_1, ..., x_n, t):
    query the oracle on input (x_1, ..., x_n, t), and reject if the oracle rejected
    initialize I ← ∅ and s ← t
    for i ← n, n−1, ..., 1:
        if x_i = s:
            update I ← I ∪ {i} and s ← 0
            halt and output I
        if x_i < s:
            query the oracle on input (x_1, ..., x_{i−1}, s − x_i)
            if the oracle accepted: update I ← I ∪ {i} and s ← s − x_i
```

Assume $t$ is achievable from $x_1, \ldots, x_n$. The reduction maintains these loop invariants:

*(1)* $s = t - x_I$.
*(2)* $s$ is achievable from $x_1, \ldots, x_i$.

These invariants trivially hold at the start of the first iteration ($i = n$). It's straightforward to see that *(1)* is maintained. *(2)* is maintained if $x_i = s$ since 0 is always achievable. If the oracle accepts in iteration $i$ then *(2)* is maintained, by the definition of SUBSET SUM. If the oracle rejects, then every (in particular, some) way of achieving $s$ from $x_1, \ldots, x_i$ doesn't use $x_i$, so $s$ is achievable from $x_1, \ldots, x_{i-1}$ and *(2)* is maintained.

The reduction cannot complete all $n$ iterations without terminating (with some $x_i = s$), since otherwise the final $s$ would be positive but achievable from nothing (by *(2)* with $i = 0$), which is a contradiction. At termination we have $s = 0$ and $s = t - x_I$ (by *(1)*), so $x_I = t$ and the output is correct.

**Exercise 2.15:** We define a decision problem $C$ and show that $C \in \mathsf{NP}$ and $B \leq_p^o C$. Since $A$ is NP-complete, we have $C \leq_p^m A$ and thus $B \leq_p^o A$ (Lemma 1.13).

Index the bits of $w \in (\{0,1\}^W)^T = \{0,1\}^{TW}$ as $w_1 w_2 \cdots w_{TW}$. A partial assignment is an element of $\{0,1,*\}^{TW}$. Say $w \in \{0,1\}^{TW}$ is consistent with a partial assignment $a$ when $w_i = a_i$ for all $i$ such that $a_i \neq *$.

(call this problem $C$)

Input: Valid input $x$ to $A$, and partial assignment $a$

Output: Does there exist $w \in (\{0,1\}^W)^T$ such that $V(x;w)$ accepts and $w$ is consistent with $a$?

$C \in \mathsf{NP}$ by a poly-time verifier that reads $w$ from the witness segment and checks that $V(x;w)$ accepts and that $w$ is consistent with $a$. This reduction shows $B \leq_p^o C$:

initialize $a \leftarrow **\cdots*$
query the oracle on input $(x,a)$
if the oracle rejected: reject (report that no such $w$ exists)
for $i \leftarrow 1,\ldots,TW$:
    let $b$ be the same partial assignment as $a$ except that $b_i = 1$
    query the oracle on input $(x,b)$
    if the oracle accepted: update $a_i \leftarrow 1$
    if the oracle rejected: update $a_i \leftarrow 0$
output $a$

The first oracle query's answer indicates whether $A(x) = 1$. The output is correct if $A(x) = 0$, so assume $A(x) = 1$. The loop maintains the invariant that there exists $w$ such that $V(x;w)$ accepts and $w$ is consistent with $a$. Inside the loop, if the oracle accepts, then of course the invariant continues to hold since the new $a$ equals $b$. To see that the invariant is maintained if the oracle rejects, assume there exists $w$ such that $V(x;w)$ accepts and $w$ is consistent with the $a$ from the beginning of the iteration. This $w$ must have $w_i = 0$ (since if it had $w_i = 1$ then it would be consistent with $b$ and the oracle would accept), so $w$ is consistent with the new $a$ (after letting $a_i \leftarrow 0$).

When the loop terminates, $a \in \{0,1\}^{TW}$ is an assignment. By the loop invariant, there exists $w$ such that $V(x;w)$ accepts and $w$ is consistent with $a$. The only assignment consistent with $a$ is $a$ itself, so $V(x;a)$ accepts and the output is correct.

**Exercise 2.16.a:** The idea is similar to the search-to-decision reduction for SAT (Theorem 2.14). We only consider one variable (rather than iterating through all variables), but we query the oracle twice—with both possible bits plugged into that variable.

```
on input φ(x_1 x_2 ··· x_n):
    if n = 0 (no variables):
        if φ has no clauses: accept
        else: reject
    else:
        query the oracle on input φ|_{0**···*}
        query the oracle on input φ|_{1**···*}
        if the oracle accepted at least one of those queries: accept
        else: reject
```

This is correct because $\varphi$ is satisfiable iff either $\varphi|_{0**\cdots*}$ or $\varphi|_{1**\cdots*}$ is satisfiable. Those queries are each smaller than $\varphi$.

**Exercise 2.16.b:** On input $z$, either the downward self-reduction outputs $A(z)$ without making a query, or it produces a query $y$ (smaller than $z$) and a bit $b$ such that $A(z) = A(y) \oplus b$. (That is, $b$ indicates whether to flip the output. In a mapping reduction, $b$ would always be 0.)

```
on input x:
    initialize z ← x and a ← 0
    repeat:
        run the downward self-reduction on input z
        if it outputs A(z) without making a query: halt and output A(z) ⊕ a
        else if it produces (y, b): update z ← y and a ← b ⊕ a
```

This maintains the invariant that $A(x) = A(z) \oplus a$. The "else" case maintains the invariant since before updating $z$ and $a$ we have $A(x) = A(z) \oplus a = (A(y) \oplus b) \oplus a = A(y) \oplus (b \oplus a)$, so after updating $z$ and $a$ we have $A(x) = A(z) \oplus a$ again.

The algorithm halts since the size of $z$ decreases in each iteration but can't go below 0. The output $A(z) \oplus a$ is correct by the invariant, so this algorithm solves $A$.

If $x$ has size $N$, then the loop has at most $N + 1$ iterations, each of which takes time $\text{poly}\, N$ since the downward self-reduction takes time poly in the size of $z$, which is $\leq N$. Thus the algorithm runs in time $(N + 1) \cdot \text{poly}\, N = \text{poly}\, N$. We conclude that $A \in \mathsf{P}$.

**Exercise 2.17:** Suppose there exists a proper $k$-coloring—that is, a partition of the nodes into $k$ many independent sets. At least one of these independent sets must have $\geq n/k$ nodes (since if each had $< n/k$ nodes, then the total number of nodes would be $< k \cdot n/k = n$). The set of all nodes outside such an independent set is a node cover (Claim 2.17) of size $\leq n - n/k$.

**Exercise 2.18:** Consider this algorithm for CLIQUE:

```
on input (G, k) where G has n nodes (numbered 1, ..., n) and m edges:
    if (k choose 2) > m: reject
    for every k-tuple of distinct nodes (v_1, ..., v_k) ∈ [n]^k:
        for each pair (i, j) with 1 ≤ i < j ≤ k:
            if v_i and v_j are not adjacent: continue to the next k-tuple
        accept
    reject
```

Since every clique with $k$ nodes has $\binom{k}{2}$ edges, we know $G$ has no clique with $k$ nodes if $G$ has fewer than $\binom{k}{2}$ edges. The rest of the algorithm is a standard brute force search. Only log space is used when $\binom{k}{2} > m$, so assume $\binom{k}{2} \leq m$. This implies $k^2 \leq 2m + k \leq N$, so $k \leq \sqrt{N}$. Each $v_i$ is one $O(\log N)$-bit word. Thus the algorithm needs $O(\sqrt{N} \log N)$ bits to remember the current $(v_1, \ldots, v_k)$, plus $O(\log N)$ bits (a constant number of words) to check whether $\{v_1, \ldots, v_k\}$ is a clique and update the $k$-tuple (for the next iteration) if not.

**Exercise 2.19:** INDEPENDENT HALF $\in$ NP by a poly-time verifier that checks that the purported witness is an independent set containing at least half of $G$'s nodes. We show INDEPENDENT SET $\leq_p^m$ INDEPENDENT HALF by mapping $(G, k)$ to this graph $G'$:

- If $k \leq n/2$, where $n$ is the number of nodes in $G$, then $G'$ contains a copy of $G$ and $n-2k$ new isolated nodes (having degree 0). Note that $G'$ has $2n-2k$ nodes. To show the reduction is correct, we argue that $G$ has an independent set of size $\geq k$ iff $G'$ has an independent set of size $\geq n-k$:

  $\Rightarrow$: Any independent set of size $\geq k$ in $G$, together with all $n-2k$ new nodes, forms an independent set of size $\geq k+(n-2k) = n-k$ in $G'$.

  $\Leftarrow$: Any independent set of size $\geq n-k$ in $G'$ must contain $\geq (n-k)-(n-2k) = k$ nodes in $G$, so those nodes form an independent set of size $\geq k$ in $G$.

- If $k > n/2$ then $G'$ contains a copy of $G$ and a clique of $2(k+1)-n$ new nodes, isolated from the rest of $G$. Note that $G'$ has $2(k+1)$ nodes. To show the reduction is correct, we argue that $G$ has an independent set of size $\geq k$ iff $G'$ has an independent set of size $\geq k+1$:

  $\Rightarrow$: Any independent set of size $\geq k$ in $G$, together with any one of the new nodes, forms an independent set of size $\geq k+1$ in $G'$.

  $\Leftarrow$: Any independent set of size $\geq k+1$ in $G'$ must contain $\leq 1$ new node and thus $\geq k$ nodes in $G$, so those nodes form an independent set of size $\geq k$ in $G$.

  (Actually, this case is unnecessary for showing INDEPENDENT HALF is NP-complete, since our reduction for 3-SAT $\leq_p^m$ INDEPENDENT SET always produces $(G, k)$ with $k \leq n/2$.)

In all cases, the reduction takes poly time.

**Exercise 2.20:** SHY SET $\in$ NP by a poly-time verifier that checks that the purported witness is a shy set of size $\geq k$ in $G$. We show INDEPENDENT SET $\leq_p^m$ SHY SET by mapping $(G,k)$ to $(G',k')$ where $G'$ contains a copy of $G$, and each node $v$ of $G$ has a new neighbor $v'$ (which is only adjacent to $v$) in $G'$, and $k' = k+n$ where $n$ is the number of nodes in $G$. To show this poly-time mapping reduction is correct, we argue that $G$ has an independent set of size $\geq k$ iff $G'$ has a shy set of size $\geq k'$:

$\Rightarrow$: Consider any independent set $S$ of size $\geq k$ in $G$. Let $S'$ contain $S$ along with all $n$ new nodes. Then $S'$ is a shy set of size $\geq k'$ in $G'$: Each $v \in S$ is adjacent to $v' \in S'$ but not adjacent to any other node in $S$ (since $S$ is independent) or to any other new node (by the definition of $G'$), and each new node $v'$ has only one neighbor in $G'$ (namely $v$) so certainly at most one neighbor in $S'$.

$\Leftarrow$: Consider any shy set $S'$ of size $\geq k'$ in $G'$. Let $S$ contain all nodes $v$ of $G$ such that $v \in S'$ and $v' \in S'$. Then $S$ is an independent set of size $\geq k$ in $G$: Each $v \in S$ is adjacent to $v' \in S'$ and so cannot be adjacent to any other $u \in S$ since $S'$ is shy. Also, $|S'| \geq k+n$ implies $|S| \geq k$ by the contrapositive: Since $|S|$ many nodes $v$ of $G$ contribute 2 to $|S'|$ (the two nodes $v$ and $v'$) and $n-|S|$ many nodes $v$ of $G$ contribute at most 1 to $|S'|$ (at most one of the nodes $v$ or $v'$), it follows that $|S'| \leq |S| \cdot 2 + (n-|S|) \cdot 1 = |S| + n$ and thus if $|S| < k$ then $|S'| < k+n$.

**Exercise 2.21:** STINGY 2-SAT $\in$ NP by a poly-time verifier that checks that the purported witness is an assignment that satisfies $\varphi$ and has $\leq k$ many variables assigned 1. We show NODE COVER $\leq_p^m$ STINGY 2-SAT by mapping $(G, k)$ to $(\varphi, k)$ where each node $v$ in $G$ has an associated variable $x_v$ in $\varphi$, and each edge $\{u, v\}$ in $G$ has an associated clause $(x_u \vee x_v)$ in $\varphi$. To show this poly-time mapping reduction is correct, we argue that $G$ has a node cover of size $\leq k$ iff $\varphi$ has a satisfying assignment with $\leq k$ many variables assigned 1:

$\Rightarrow$: Suppose $S$ is a node cover of size $\leq k$ in $G$. Then the assignment where $x_v = 1$ if $v \in S$ and $x_v = 0$ if $v \notin S$ satisfies $\varphi$ since $S$ is a node cover (for each clause $(x_u \vee x_v)$, we have $x_u = 1$ or $x_v = 1$ since $u \in S$ or $v \in S$ since $S$ covers the edge $\{u, v\}$), and the number of variables assigned 1 is $|S| \leq k$.

$\Leftarrow$: Suppose some assignment satisfies $\varphi$ with $\leq k$ many variables assigned 1. Then the set of nodes $S$ where $v \in S$ if $x_v = 1$ and $v \notin S$ if $x_v = 0$ is a node cover in $G$ since $\varphi$ is satisfied (for each edge $\{u, v\}$, we have $u \in S$ or $v \in S$ since $x_u = 1$ or $x_v = 1$ since the assignment satisfies the clause $(x_u \vee x_v)$), and $|S|$ is the number of variables assigned 1, which is $\leq k$.

**Exercise 2.22:** DOMINATING SET $\in$ NP by a poly-time verifier that checks that the purported witness is a dominating set of size $\leq k$ in $G$. We show NODE COVER $\leq_p^m$ DOMINATING SET by mapping $(G, k)$ to $(G', k)$ where $G'$ is $G$ but with any nodes of degree 0 removed, and for each edge $e = \{u, v\}$ of $G$, $G'$ has not only the edge $e$ but also a path of two new edges $\{u, w_e\}, \{w_e, v\}$ where $w_e$ is a new node. To show this poly-time mapping reduction is correct, we argue that $G$ has a node cover of size $\leq k$ iff $G'$ has a dominating set of size $\leq k$:

$\Rightarrow$: Suppose $S$ is a node cover of size $\leq k$ in $G$. We may assume $S$ contains no nodes of degree 0 (since removing such nodes from $S$ would preserve the node cover property). Then we claim that $S$ is a dominating set in $G'$. Consider any node of $G'$. If it's an original node $v$ from $G$, then there's an edge $\{u, v\}$ in $G$ (since $v$ has degree $\geq 1$), and either $u \in S$ or $v \in S$ since $S$ is a node cover in $G$, so $v$ is dominated by $S$ since $G'$ has the edge $\{u, v\}$. If it's a new node $w_e$ associated with some edge $e = \{u, v\}$ from $G$, then again either $u \in S$ or $v \in S$ since $S$ is a node cover in $G$, so $w_e$ is dominated by $S$ since $G'$ has the edges $\{u, w_e\}, \{w_e, v\}$.

$\Leftarrow$: Suppose $S'$ is a dominating set of size $\leq k$ in $G'$. Define a subset $S$ of $G$'s nodes: If $v \in S'$ is an original node of $G$, then put $v$ in $S$, and if $w_e \in S'$ is a new node of $G'$ associated with some edge $e = \{u, v\}$ of $G$, then put one of $u$ or $v$ in $S$ (it doesn't matter which one) if it isn't already there. Then $|S| \leq |S'|$, and we claim that $S$ is a node cover in $G$. Consider any edge $e = \{u, v\}$ of $G$. Since $w_e$ is dominated by $S'$, we have either $u \in S'$ and thus $u \in S$, or $v \in S'$ and thus $v \in S$, or $w_e \in S'$ and thus either $u \in S$ or $v \in S$. In all cases, edge $e$ is covered by $S$.

**Exercise 2.23:** ALMOST CLIQUE $\in$ NP by a poly-time verifier that checks that the purported witness is an almost clique of size $\geq k$ in $G$. We show CLIQUE $\leq_p^m$ ALMOST CLIQUE by mapping $(G,k)$ to $(G',k')$ where $G'$ has a copy of $G$ and two new nodes $y$ and $z$, which both have edges to all nodes of $G$ but $\{y,z\}$ is not an edge in $G'$, and $k' = k+2$. To show this poly-time mapping reduction is correct, we argue that $G$ has a clique of size $\geq k$ iff $G'$ has an almost clique of size $\geq k'$:

$\Rightarrow$: Consider any clique $S$ of size $\geq k$ in $G$. Then $S \cup \{y,z\}$ is an almost clique of size $\geq k'$ in $G'$ since $\{y,z\}$ is the only edge not present in $G'$ between two of those nodes. This is because for any other pair of nodes in $S \cup \{y,z\}$, if both are in $S$ then there's an edge since $S$ is a clique, and if one is $y$ or $z$ then there's an edge by the definition of $G'$.

$\Leftarrow$: Consider any almost clique $S$ of size $\geq k'$ in $G'$.

- Suppose $y \in S$ and $z \in S$. Then $\{y,z\}$ must be the missing edge of $S$, so $S \setminus \{y,z\}$ is a clique of size $\geq k$ in $G$.
- Suppose $y \notin S$ and $z \notin S$. If $\{u,v\}$ denotes the missing edge of $S$, then $S \setminus \{u\}$ is a clique of size $\geq k'-1 > k$ in $G$ (as is $S \setminus \{v\}$).
- Suppose $y \in S$ and $z \notin S$ (or symmetrically, $y \notin S$ and $z \in S$). If $\{u,v\}$ denotes the missing edge of $S$, then $u \neq y$ since $y$ has edges to all nodes in $G$, and $S \setminus \{u,y\}$ is a clique of size $\geq k$ in $G$ (as is $S \setminus \{v,y\}$).

**Exercise 2.24:** We show INDEPENDENT SET $\leq_p^m$ INDEPENDENT SET GAP COUNTING by mapping $(G,k)$ to $(G',k')$ as follows: Say $G=(V,E)$ and $|V|=n$. Then $G'=(V',E')$ where $V'=V\times[n+1]$ and $E'=\left\{\left\{(u,i),(v,j)\right\} : \{u,v\}\in E\right\}$, and $k'=(n+1)k$. To show this poly-time mapping reduction is correct, we argue that:

*(i)* If $G$ has an independent set of size $\geq k$, then $G'$ has $\geq 2^{k'}$ many independent sets.

*(ii)* If $G$ has no independent set of size $\geq k$, then $G'$ has $\leq 2^{k'-1}$ many independent sets.

A key observation is that $S'\subseteq V'$ is an independent set in $G'$ iff

$$S'_V=\left\{u : (u,i)\in S' \text{ for at least one } i\right\}\subseteq V$$

is an independent set in $G$:

$\Rightarrow$: Assume $S'$ is independent. For all $u\in S'_V$ and $v\in S'_V$, we have $(u,i)\in S'$ and $(v,j)\in S'$ for some $i,j$. Since $\left\{(u,i),(v,j)\right\}\notin E'$, we have $\{u,v\}\notin E$. Thus $S'_V$ is independent.

$\Leftarrow$: Assume $S'_V$ is independent. For all $(u,i)\in S'$ and $(v,j)\in S'$, we have $u\in S'_V$ and $v\in S'_V$. Since $\{u,v\}\notin E$, we have $\left\{(u,i),(v,j)\right\}\notin E'$. Thus $S'$ is independent.

Proof of *(i)*: Assume $S$ is an independent set of size $\geq k$ in $G$. There are $\geq (2^{n+1})^k$ many sets $S'$ such that $S'_V\subseteq S$, because for each $u\in S$ there are $2^{n+1}$ possibilities of $\left\{i\in[n+1] : (u,i)\in S'\right\}$. Each such $S'$ is an independent set in $G'$, by the key observation. Thus $G'$ has $\geq 2^{(n+1)k}=2^{k'}$ many independent sets.

Proof of *(ii)*: Assume every independent set in $G$ has size $\leq k-1$. There are $\leq 2^n$ many independent sets $S$ in $G$. For each such $S$, there are $\leq (2^{n+1})^{k-1}$ many sets $S'$ such that $S'_V=S$. For every independent set $S'$ in $G'$, $S'_V$ is an independent set in $G$ by the key observation. Thus $G'$ has $\leq 2^n\cdot 2^{(n+1)(k-1)}=2^{n+(n+1)k-(n+1)}=2^{k'-1}$ many independent sets.

**Exercise 2.25.a:** Map $(G, s, t)$ to the graph $G'$ that's $G$ but with any incoming edges to $s$ removed and any outgoing edges from $t$ removed. To show this poly-time mapping reduction is correct, we argue that $G$ has a full path from $s$ to $t$ iff there exist $s'$ and $t'$ such that $G'$ has a full path from $s'$ to $t'$:

$\Rightarrow$: Any full path from $s$ to $t$ in $G$ must not use any incoming edges to $s$ or any outgoing edges from $t$, so it's also a full path from $s' = s$ to $t' = t$ in $G'$.

$\Leftarrow$: Any full path from $s'$ to $t'$ in $G'$ must have $s' = s$ since $s$ has no incoming edges (hence can only be visited at the start) and $t' = t$ since $t$ has no outgoing edges (hence can only be visited at the end) and is thus a full path from $s$ to $t$ in $G$.

**Exercise 2.25.b:** Map $G$ to $(G', s', t')$ where $G'$ contains a copy of $G$ and a new node $s'$ with edges to all nodes of $G$, and a new node $t'$ with edges from all nodes of $G$. To show this poly-time mapping reduction is correct, we argue that (there exist $s$ and $t$ such that $G$ has a full path from $s$ to $t$) iff $G'$ has a full path from $s'$ to $t'$:

$\Rightarrow$: For any full path from some $s$ to some $t$ in $G$, prepending $s' \rightarrow s$ and appending $t \rightarrow t'$ yields a full path from $s'$ to $t'$ in $G'$.

$\Leftarrow$: For any full path from $s'$ to $t'$ in $G'$, removing the first edge $s' \rightarrow s$ and removing the last edge $t \rightarrow t'$ yields a full path from some $s$ to some $t$ in $G$.

**Exercise 2.26.a:** We combine insights from our poly-time algorithms for Graph 2-Coloring (Theorem 2.1) and Small Subset Sum (Theorem 2.3). For each $i$, the following algorithm computes the set $S_i$ of all possible numbers of red nodes in all proper 2-colorings of $G$'s first $i$ connected components. Since the $i^{\text{th}}$ connected component's proper 2-coloring (if it exists) is unique except for interchanging the two colors (say, red and blue), this gives two possible numbers (of red nodes in the $i^{\text{th}}$ connected component) to add to each number in $S_{i-1}$.

```
if the number of nodes n is odd: reject
compute c ← number of connected components in G
let S_0 ← {0}
for i ← 1, 2, ..., c:
    find a proper 2-coloring of G's i^th connected component
    if that's impossible: reject
    let a_i and b_i be the numbers of red and blue nodes in that proper 2-coloring
    let S_i ← (S_{i-1} + a_i) ∪ (S_{i-1} + b_i)
if n/2 ∈ S_c: accept
else: reject
```

The algorithm runs in poly time, including the line $S_i \leftarrow (S_{i-1}+a_i)\cup(S_{i-1}+b_i)$ since $|S_{i-1}| \leq n$.

**Exercise 2.26.b:** Connected Equitable 3-Coloring $\in$ NP by a poly-time verifier that checks that the purported witness is a proper 3-coloring of $G$ with equal numbers of red, green, and blue nodes. Defining Connected 3-Coloring to be Graph 3-Coloring but where the input graph must be connected, we know Connected 3-Coloring is NP-complete since our reduction for 3-Sat $\leq_p^m$ Graph 3-Coloring (Theorem 2.25) always produces a connected graph and thus also shows 3-Sat $\leq_p^m$ Connected 3-Coloring. We show Connected 3-Coloring $\leq_p^m$ Connected Equitable 3-Coloring by mapping $G$ to $G'$ where $G'$ contains a copy of $G$, and for each node $v$ in $G$, we have two new nodes $v'$ and $v''$ forming a triangle with $v$ in $G'$. Note that $G'$ is still connected. To show this poly-time mapping reduction is correct, we argue that $G$ has a proper 3-coloring iff $G'$ has an equitable proper 3-coloring:

$\Rightarrow$: Consider any proper 3-coloring of $G$. For each node $v$ in $G$, give its companions $v'$ and $v''$ the two colors other than $v$'s color. This yields a proper 3-coloring of $G'$, and it has equal numbers of red, green, and blue nodes since in each triangle $v, v', v''$, there is one of each color.

$\Leftarrow$: Any proper 3-coloring of $G'$ (with equal numbers of red, green, and blue nodes, though that doesn't matter now) is also a proper 3-coloring of $G$ if we ignore the new nodes, since $G'$ contains a copy of $G$.

**Exercise 2.27:** BARELY 3-SAT $\in$ NP by a poly-time verifier that checks that the purported witness is an assignment satisfying one and only one literal in each clause of $\varphi$. We show GRAPH 3-COLORING $\leq_p^m$ BARELY 3-SAT by mapping $G$ to $\varphi$ where each node $v$ in $G$ has three associated variables $v_r$, $v_g$, $v_b$ and one associated clause $(v_r \vee v_g \vee v_b)$, and each edge $e = \{u, v\}$ in $G$ has three associated variables $e_r$, $e_g$, $e_b$ and three associated clauses $(u_r \vee v_r \vee e_r) \wedge (u_g \vee v_g \vee e_g) \wedge (u_b \vee v_b \vee e_b)$. To show this poly-time mapping reduction is correct, we argue that $G$ has a proper 3-coloring iff $\varphi$ has a barely satisfying assignment:

$\Rightarrow$: Suppose $c : V \to \{r, g, b\}$ is a proper 3-coloring of $G$'s set of nodes $V$. For each node $v$, assign 1 to $v_{c(v)}$ and 0 to the other two associated variables (barely satisfying $(v_r \vee v_g \vee v_b)$), and for each edge $e = \{u, v\}$, assign 0 to $e_{c(u)}$ (barely satisfying $(u_{c(u)} \vee v_{c(u)} \vee e_{c(u)})$) and 0 to $e_{c(v)}$ (barely satisfying $(u_{c(v)} \vee v_{c(v)} \vee e_{c(v)})$) and 1 to the other associated variable (barely satisfying the other associated clause).

$\Leftarrow$: Suppose some assignment barely satisfies $\varphi$. For each node $v$, since $(v_r \vee v_g \vee v_b)$ is barely satisfied, we assign $v$ the unique color $c(v)$ such that $v_{c(v)} = 1$. To see that this 3-color assignment is proper, consider any edge $e = \{u, v\}$. If $c(u) = c(v)$ then the clause $(u_{c(u)} \vee v_{c(u)} \vee e_{c(u)})$ wouldn't be barely satisfied, since $u_{c(u)}$ and $v_{c(u)}$ would both be 1.

**Exercise 2.28:** Balanced Tripartition $\in$ NP by a poly-time verifier that checks that the purported witness is a partition of $[n]$ into $I,J,K$ such that $x_I = x_J = x_K$, using an Iterated Addition algorithm. We show Balanced Partition $\leq_p^m$ Balanced Tripartition. On input $(x_1,\ldots,x_n)$:

- If $x_{[n]}$ is even then map to $(x_1,\ldots,x_n,x_{n+1})$ where $x_{n+1} = x_{[n]}/2$. To show that the reduction is correct, we argue that there exists $I \subseteq [n]$ such that $x_I = x_{[n]}/2$ (that is, $x_I = x_{[n]\setminus I}$) iff there exists a partition of $[n+1]$ into $I',J',K'$ such that $x_{I'} = x_{J'} = x_{K'}$:

  $\Rightarrow$: Assume $x_I = x_{[n]\setminus I} = x_{[n]}/2$. Letting $I' = I$, $J' = [n] \setminus I$, and $K' = \{n+1\}$, we have $x_{I'} = x_{J'} = x_{K'}$.

  $\Leftarrow$: Assume $x_{I'} = x_{J'} = x_{K'} = x_{[n+1]}/3 = x_{[n]}/2$. By symmetry of $I',J',K'$, we may assume $n+1 \notin I'$. Letting $I = I'$, we have $I \subseteq [n]$ and $x_I = x_{[n]}/2$.
- If $x_{[n]}$ is odd then the input is necessarily a no-input, so map to some trivial no-input of Balanced Tripartition such as (1).

In all cases, the reduction takes poly time.

**Exercise 2.29.a:** Consider this algorithm on input $(G, s, t)$:

```
topologically order G's nodes v_1, v_2, ..., v_n, so i < j for each edge (v_i, v_j)
if v_1 ≠ s or v_n ≠ t: reject
for i ← 1, 2, ..., n−1:
    if (v_i, v_{i+1}) is not an edge in G: reject
accept
```

$G$ has a topological order since it's a dag (Theorem 0.9). To show this algorithm is correct, we argue that it accepts iff $G$ has a full path from $s$ to $t$:

$\Rightarrow$: If the algorithm accepts, then $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n$ is a full path from $s$ to $t$ in $G$.

$\Leftarrow$: Assume the algorithm rejects. If $v_1 \neq s$ then $G$ has no full path from $s$ to $t$ since $v_1$ has indegree 0 and thus $v_1$ is not reachable from $s$. If $v_n \neq t$ then $G$ has no full path from $s$ to $t$ since $v_n$ has outdegree 0 and thus $t$ is not reachable from $v_n$. Now, assume $v_1 = s$ and $v_n = t$ and $(v_i, v_{i+1})$ is not an edge in $G$ for some $i \in [n-1]$, and suppose for contradiction that $G$ has a full path from $s$ to $t$. Consider the least $j$ such that this path does not step from $v_j$ to $v_{j+1}$. (Such a $j$ exists since the path cannot step from $v_i$ to $v_{i+1}$.) Thus the path begins $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{j-1} \rightarrow v_j \rightarrow v_k$ for some $k > j+1$. But then the path cannot visit $v_{j+1}$ since $v_{j+1}$ is not reachable from $v_k$. This contradicts the assumption that the path is full (visits all nodes).

A topological order can be found in linear time, by the proof of Theorem 0.9. Thus this algorithm runs in linear time.

**Exercise 2.29.b:** EXACT PATH LENGTH IN A DAG $\in$ NP by a poly-time verifier that checks that the purported witness is a path of length $T$ from $s$ to $t$ in $G$, using an ITERATED ADDITION algorithm to calculate the path's length. We show SUBSET SUM $\leq_p^m$ EXACT PATH LENGTH IN A DAG by mapping $(x_1, \ldots, x_n, T)$ to $(G, s, t, \ell, T')$ (where $\ell$ is the tuple of edge lengths) as follows: $G$ has a path with nodes $v_0, v_1, \ldots, v_{2n}$, on which each of the $2n$ edges has length 1. For each $i \in [n]$, $G$ has an edge of length $x_i + 2$ from $v_{2i-2}$ to $v_{2i}$. Note that $G$ is indeed a dag (without multi-edges) since the nodes are topologically ordered—each edge goes from a lower index node to a higher index node. Let $s = v_0$ and $t = v_{2n}$ and $T' = T + 2n$. To show this poly-time mapping reduction is correct, we argue that there exists $I \subseteq [n]$ with $x_I = T$ iff $G$ has a path of length $T'$ from $s$ to $t$:

$\Rightarrow$: Assume $x_I = T$. Consider this path from $s$ to $t$: For each $i \in [n]$, if $i \in I$ then take the edge $v_{2i-2} \rightarrow v_{2i}$ of length $x_i + 2$, otherwise take the edges $v_{2i-2} \rightarrow v_{2i-1} \rightarrow v_{2i}$ of length $1 + 1 = 2$. This path has total length $\sum_{i \in I}(x_i + 2) + \sum_{i \in [n] \setminus I} 2 = x_I + 2n = T'$.

$\Leftarrow$: Consider any path of length $T'$ from $s$ to $t$. Define $I \subseteq [n]$ as follows: For each $i \in [n]$, if the path takes the edge $v_{2i-2} \rightarrow v_{2i}$ then include $i$ in $I$, otherwise if the path takes the edges $v_{2i-2} \rightarrow v_{2i-1} \rightarrow v_{2i}$ then exclude $i$ from $I$. As in the $\Rightarrow$ direction, this path has total length $x_I + 2n$, so we must have $x_I = T' - 2n = T$.

**Exercise 2.29.c:** Consider this algorithm on input $(G, s, t, \ell, T)$ where $\ell$ is the tuple of edge lengths:

```
topologically order G's nodes v_1, v_2, ..., v_n, so i < j for each edge (v_i, v_j)
let a, b be such that s = v_a and t = v_b
if a > b: reject
initialize S_j ← ∅ for each j ∈ [n]
let S_a ← {0}
for j ← a+1, a+2, ..., b:
    for each edge (v_i, v_j):
        for each r ∈ S_i:
            insert r + ℓ_(v_i,v_j) into S_j (if not already there)
if T ∈ S_b: accept
else: reject
```

Using the notation $S+x = \{r+x : r \in S\}$, we have $S_j = \emptyset$ for each $j < a$ and $S_j = \bigcup_{\text{edge } (v_i, v_j)}(S_i + \ell_{(v_i,v_j)})$ for each $a < j \leq b$. The algorithm is correct since $S_j$ is the set of all lengths of paths from $s$ to $v_j$. Since each path has $\leq n$ edges and each edge has length $\leq n^2$, each path has length $\leq n^3$ and thus each $S_j \subseteq \{0, 1, \ldots, n^3\}$. The outer two loops just iterate over all nodes and edges and hence have $\leq N$ iterations, where $N$ is the input size. The innermost loop has $\leq n^3 + 1 \leq N^3$ iterations. Each of the $\leq N^4$ iterations takes $O(1)$ time. Finding a topological order takes $O(N)$ time, by the proof of Theorem 0.9. Thus the whole algorithm runs in poly time.

**Exercise 2.30.a:** We show 2-NAE SAT $\leq_p^m$ GRAPH 2-COLORING, which implies 2-NAE SAT $\in$ P since GRAPH 2-COLORING $\in$ P (Theorem 2.1). The input $\varphi$ can have two types of 2-NAE clauses:

- Equality type: $\neg(x_i \Longleftrightarrow \overline{x_j})$ and $\neg(\overline{x_i} \Longleftrightarrow x_j)$ are equivalent to $(x_i \Longleftrightarrow x_j)$, which is satisfied iff $x_i$ and $x_j$ are assigned the same bit, and thus $x_i$ and $x_j$ might as well be the same variable.
- Non-equality type: $\neg(x_i \Longleftrightarrow x_j)$ and $\neg(\overline{x_i} \Longleftrightarrow \overline{x_j})$ are equivalent to $(x_i \oplus x_j)$, which is satisfied iff $x_i$ and $x_j$ are assigned opposite bits.

First, we map $\varphi$ to $\varphi'$: For each equality type 2-NAE clause $(x_i \Longleftrightarrow x_j)$, we remove it and replace $x_j$ with $x_i$ throughout the whole formula. Thus $\varphi$ is satisfiable iff $\varphi'$ is satisfiable. Since $\varphi'$ only has non-equality type 2-NAE clauses, we map it to the graph $G$ with nodes representing the variables of $\varphi'$ and with an edge $\{x_i, x_j\}$ for each 2-NAE clause $(x_i \oplus x_j)$ of $\varphi'$. Viewing 0 and 1 as the two colors, $(x_i \oplus x_j)$ is satisfied by an assignment iff the edge $\{x_i, x_j\}$ is multicolored by the corresponding 2-color assignment. Thus $\varphi'$ (and hence $\varphi$) is satisfiable iff $G$ has a proper 2-coloring. This reduction takes poly time.

**Exercise 2.30.b:** 4-NAE SAT $\in$ NP by a verifier that checks that the purported witness is a satisfying assignment. We show 3-SAT $\leq_p^m$ 4-NAE SAT. Map 3-CNF $\varphi$ to 4-NAE formula $\varphi'$ by creating a new variable $z$ and changing each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ (where $\ell_1, \ell_2, \ell_3$ are literals) to $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow \ell_3 \Leftrightarrow z)$. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ is satisfiable:

$\Rightarrow$: For any assignment that satisfies $\varphi$, the same assignment along with $z = 0$ satisfies $\varphi'$, since if at least one of $\ell_1, \ell_2, \ell_3$ is 1, then among $\ell_1, \ell_2, \ell_3, z$ there's at least one 1 (the same $\ell_i$) and at least one 0 (namely $z$).

$\Leftarrow$: Consider any assignment that satisfies $\varphi'$. Since the bit values are interchangeable (swapping 0 and 1 in any assignment doesn't change the value of $\varphi'$), we may assume $z = 0$. Now the same assignment (ignoring $z$) satisfies $\varphi$, since if there's at least one 1 among $\ell_1, \ell_2, \ell_3, z$ then it must be among $\ell_1, \ell_2, \ell_3$.

**Exercise 2.30.c:** 3-NAE SAT $\in$ NP by a verifier that checks that the purported witness is a satisfying assignment. We show 4-NAE SAT $\leq_p^m$ 3-NAE SAT. Map 4-NAE formula $\varphi$ to 3-NAE formula $\varphi'$ by replacing the $i^{\text{th}}$ 4-NAE clause $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow \ell_3 \Leftrightarrow \ell_4)$ (where $\ell_1, \ell_2, \ell_3, \ell_4$ are literals) with two 3-NAE clauses $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow y_i) \wedge \neg(\ell_3 \Leftrightarrow \ell_4 \Leftrightarrow \overline{y_i})$ where $y_i$ is a fresh variable, for all $i$. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ is satisfiable:

$\Rightarrow$: Consider any assignment that satisfies $\varphi$. For the $i^{\text{th}}$ 4-NAE clause $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow \ell_3 \Leftrightarrow \ell_4)$ of $\varphi$, there's at least one 1 and at least one 0 among $\ell_1, \ell_2, \ell_3, \ell_4$. The corresponding 3-NAE clauses in $\varphi'$ are satisfied by the same assignment along with:

- $y_i = 1$ if $\ell_1 = \ell_2 = 0$ or if $\ell_3 = \ell_4 = 1$.
- $y_i = 0$ if $\ell_1 = \ell_2 = 1$ or if $\ell_3 = \ell_4 = 0$.
- $y_i$ doesn't matter otherwise.

$\Leftarrow$: Consider any assignment that satisfies $\varphi'$. The same assignment (ignoring $y_1, y_2, \ldots$) satisfies $\varphi$: For the $i^{\text{th}}$ 4-NAE clause $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow \ell_3 \Leftrightarrow \ell_4)$ of $\varphi$, if $y_i = 0$ then there's a 1 among $\ell_1, \ell_2$ and a 0 among $\ell_3, \ell_4$, and if $y_i = 1$ then there's a 0 among $\ell_1, \ell_2$ and a 1 among $\ell_3, \ell_4$.

**Exercise 2.30.d:** Max Cut Decision $\in$ NP by a verifier that checks that the purported witness is a cut that's crossed by $\geq k$ edges. We show 3-NAE Sat $\leq_p^m$ Max Cut Decision. Map 3-NAE formula $\varphi$ to $(G, k)$ where $G$ is as in the proof of Theorem 2.21, and $k = n + 5m$ where $n$ is the number of variables and $m$ is the number of 3-NAE clauses. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $G$ has a cut that's crossed by $\geq k$ edges:

$\Rightarrow$: Consider any assignment that satisfies $\varphi$. Define a cut of $G$ with a "0 side" and a "1 side," where each node is on the side corresponding to the value of the literal labeling the node. This cut is crossed by each of the $n$ variable gadget edges, each of the $3m$ clause-to-variable edges, and 2 of the 3 edges in each 3-NAE clause gadget (since the literals in the 3-NAE clause are either two 0s and one 1 or one 0 and two 1s under the assignment), for a total of $k = n + 5m$ edges.

$\Leftarrow$: Consider any cut of $G$ that's crossed by $\geq k$ edges. Say one side is the "0 side" and the other is the "1 side." It's impossible to cut more than $n + 5m$ of the $n + 6m$ edges in $G$, since in each of the $m$ 3-NAE clause gadget triangles, at least one edge must be uncut (since there are three nodes but only two sides). So the cut must be crossed by exactly $k$ edges, and the $m$ uncut edges must be one per 3-NAE clause gadget. Thus all variable gadget edges, all clause-to-variable edges, and 2 out of 3 edges in each 3-NAE clause gadget must be cut. Assign each variable $x_i$ the value corresponding to the side of the cut that the $x_i$ variable gadget node is on. Since all variable gadget edges and all clause-to-variable edges are cut, every node must be on the side corresponding to its literal's value under this assignment. Each 3-NAE clause of $\varphi$ must be satisfied since 2 out of 3 of its gadget's edges are cut (because if it weren't satisfied, then all the 3-NAE clause gadget's nodes would be on the same side, so none of this gadget's edges would be cut). Thus $\varphi$ is satisfiable.

**Exercise 2.30.e:** MAX 2-SAT DECISION $\in$ NP by a verifier that checks that the purported witness is an assignment satisfying $\geq k$ clauses. We show MAX CUT DECISION $\leq_p^m$ MAX 2-SAT DECISION. Map $(G, k)$ to $(\varphi, k')$ where each node $v$ in $G$ becomes a variable $x_v$ in $\varphi$, each edge $\{u, v\}$ in $G$ becomes a pair of clauses $(x_u \vee x_v) \wedge (\overline{x_u} \vee \overline{x_v})$ in $\varphi$, and $k' = k + m$ where $m$ is the number of edges in $G$. To show this poly-time mapping reduction is correct, we argue that $G$ has a cut crossed by $\geq k$ edges iff $\varphi$ has an assignment satisfying $\geq k'$ clauses:

$\Rightarrow$: Consider any cut crossed by $\geq k$ edges in $G$, say with a "0 side" and a "1 side." Assign each variable $x_v$ according to which side of the cut $v$ is on. If edge $\{u, v\}$ crosses the cut, then both clauses $(x_u \vee x_v) \wedge (\overline{x_u} \vee \overline{x_v})$ are satisfied, since one of $x_u, x_v$ is 1 and the other is 0. If edge $\{u, v\}$ doesn't cross the cut, then one of its two clauses is satisfied, since $x_u, x_v$ are either both 1 or both 0. Thus this assignment satisfies $\geq 2 \cdot k + 1 \cdot (m - k) = k'$ clauses.

$\Leftarrow$: Consider any assignment satisfying $\geq k'$ clauses in $\varphi$. Cut $G$'s nodes into a "0 side" and a "1 side" by putting each node $v$ on the side corresponding to $x_v$'s value. For each edge $\{u, v\}$, if both clauses $(x_u \vee x_v) \wedge (\overline{x_u} \vee \overline{x_v})$ are satisfied, then $x_u \neq x_v$ and so $\{u, v\}$ crosses this cut. It's impossible for neither clause to be satisfied. So the number of edges with both clauses satisfied, and hence crossing the cut, must be $\geq k' - m = k$.

**Exercise 2.30.f:** We show 3-NAE SAT $\leq_p^m$ GRAPH 3-COLORING. Map 3-NAE formula $\varphi$ to $G$ as in the proof of Theorem 2.21 but with an extra "base" node adjacent to all variable gadget nodes. To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $G$ has a proper 3-coloring, using colors $\{0, 1, 2\}$:

$\Rightarrow$: Consider any assignment that satisfies $\varphi$. Let "base" have color 2. Color the variable gadget nodes according to the satisfying assignment: Node $x_i$'s color is variable $x_i$'s value, and node $\overline{x_i}$'s color is the opposite bit. Since each 3-NAE clause is satisfied, we can color its gadget as follows: Give color 0 to one node whose literal has value 0, give color 1 to one node whose literal has value 1, and give color 2 to the other node. This 3-color assignment is proper: Each variable gadget edge, "base"-to-variable-gadget edge, 3-NAE clause triangle edge, and clause-gadget-to-variable-gadget edge is multicolored.

$\Leftarrow$: Consider any proper 3-coloring of $G$. Since colors are interchangeable, we may assume "base" has color 2. For each variable $x_i$, the nodes $x_i, \overline{x_i}$ must be colored $0, 1$ or $1, 0$ since they're adjacent to each other and to "base". Assign each variable $x_i$ the same bit as node $x_i$'s color, and note that the literal $\overline{x_i}$ is the same bit as node $\overline{x_i}$'s color. For each 3-NAE clause gadget, the nodes must have all three colors due to the triangle edges. The node with color 0 is adjacent to a variable gadget node with value 1, so the corresponding literal in the 3-NAE clause has value 0. The node with color 1 is adjacent to a variable gadget node with value 0, so the corresponding literal in the 3-NAE clause has value 1. Each 3-NAE clause in $\varphi$ is therefore satisfied by this assignment.

**Exercise 2.31.a:** Node Disjoint Paths $\in$ NP by a verifier that checks that the purported witness consists of node disjoint paths from $s_i$ to $t_i$ for all $i$. We show 3-Sat $\leq_p^m$ Node Disjoint Paths. Map an $n$-variable $m$-clause CNF $\varphi$ to $\big(G, (s_1, t_1), \ldots, (s_{m+n}, t_{m+n})\big)$:

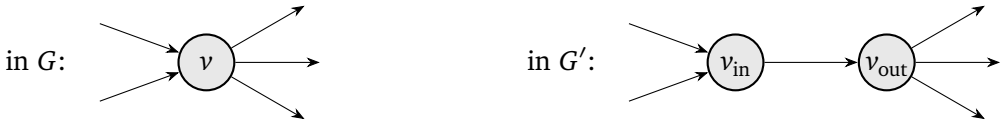
- Each clause has a separate five-node gadget in $G$: For the $i^{\text{th}}$ clause, node $s_i$ has edges to three nodes representing the clause's three literals, and those three nodes each have an edge to $t_i$.
- For each variable $x_i$, $G$ has two paths from $s_{m+i}$ to $t_{m+i}$: One path goes through all nodes representing $x_i$ literals in the clause gadgets (in any order), and the other path goes through all nodes representing $\overline{x_i}$ literals.

To show this poly-time mapping reduction is correct, we argue that $\varphi$ is satisfiable iff there exist node disjoint paths from $s_i$ to $t_i$ for all $i$:

$\Rightarrow$: Suppose $\varphi$ has a satisfying assignment $a \in \{0,1\}^n$. For variable $x_i$, if $a_i = 0$ then pick the path from $s_{m+i}$ to $t_{m+i}$ through the $x_i$ nodes, and if $a_i = 1$ then pick the path from $s_{m+i}$ to $t_{m+i}$ through the $\overline{x_i}$ nodes. For the $i^{\text{th}}$ clause, pick the path from $s_i$ to $t_i$ through a node representing a satisfied literal. These paths are node disjoint.

$\Leftarrow$: Suppose there exist node disjoint paths from $s_i$ to $t_i$ for all $i$. For variable $x_i$, assign $a_i = 0$ if the path from $s_{m+i}$ to $t_{m+i}$ goes through the $x_i$ nodes, and assign $a_i = 1$ if the path from $s_{m+i}$ to $t_{m+i}$ goes through the $\overline{x_i}$ nodes. (These are the only two possible paths.) For the $i^{\text{th}}$ clause, the path from $s_i$ to $t_i$ must go through a node representing a satisfied literal since all the paths are node disjoint. Thus $a$ satisfies $\varphi$.

**Exercise 2.31.b:** EDGE DISJOINT PATHS $\in$ NP by a verifier that checks that the purported witness consists of edge disjoint paths from $s_i$ to $t_i$ for all $i$. We show NODE DISJOINT PATHS $\leq_p^m$ EDGE DISJOINT PATHS. Map $\big(G, (s_1, t_1), \ldots, (s_k, t_k)\big)$ to $\big(G', (s_{1,\text{in}}, t_{1,\text{out}}), \ldots, (s_{k,\text{in}}, t_{k,\text{out}})\big)$ where each node $v$ in $G$ corresponds to an edge $(v_{\text{in}}, v_{\text{out}})$ in $G'$, and each edge $(u, w)$ in $G$ corresponds to an edge $(u_{\text{out}}, w_{\text{in}})$ in $G'$:




To show this poly-time mapping reduction is correct, we argue that $G$ has node disjoint paths from $s_i$ to $t_i$ for all $i$ iff $G'$ has edge disjoint paths from $s_{i,\text{in}}$ to $t_{i,\text{out}}$ for all $i$:

$\Rightarrow$: For any node disjoint paths in $G$ from $s_i$ to $t_i$ for all $i$, the corresponding paths in $G'$ are edge disjoint.

$\Leftarrow$: For any edge disjoint paths in $G'$ from $s_{i,\text{in}}$ to $t_{i,\text{out}}$ for all $i$, the corresponding paths in $G$ are node disjoint, because if two of the paths in $G$ shared a node $v$, then the corresponding paths in $G'$ would share the edge $(v_{\text{in}}, v_{\text{out}})$.

**Exercise 2.32:** To express "$p \geq a$": For each $i$ such that $a_i = 1$, include the clause $\left(p_i \vee \bigvee_{j>i\,:\,a_j=0} p_j\right)$. To modify this to express "$p > a$": Also include the clause $\left(\bigvee_{j\,:\,a_j=0} p_j\right)$ (which is the empty clause if $a = 1 \cdots 1$). The size is $O(W^2)$.

**Exercise 2.33:** Consider any CNF expressing "$p \neq q$". For each assignment of the form $(p, q) = (a, a)$, the CNF evaluates to 0 and so at least one clause is falsified. We claim that no clause in the CNF can be falsified by two such assignments. By the pigeonhole principle (pigeon $a$ flies into a hole corresponding to a clause falsified by $(a, a)$), this implies that the CNF has at least $2^W$ clauses, since there are that many possibilities of $a$. Suppose for contradiction that $(a, a)$ and $(b, b)$ falsify the same clause, where $a \neq b$. Falsifying the clause means being consistent with a certain partial assignment $(c, d)$. Thus $a$ and $b$ are both consistent with $c$ and with $d$. So $(a, b)$ is consistent with $(c, d)$ and therefore falsifies the clause and the CNF. But $(a, b)$ should satisfy the CNF since $a \neq b$, so this is a contradiction.

**Exercise 2.34.a:** We modify the CNF from the proof of Theorem 2.31 so that the conditions "if $p = 0$", "if $p = 1$", "if $p = 2$", ..., "if $p = W-1$" only check the least significant $\lfloor \log_2(W-1) \rfloor + 1$ bits of $p$, rather than all bits of $p$. The "$p \leq W-1$" requirement means only those least significant bits matter, since the other bits of $p$ must be 0 for the CNF to be satisfied anyway.

Define $j = \lfloor \log_2(W-1) \rfloor$ as the index of the most significant 1 in the binary representation of $W-1$. We may assume $W > 1$. To express "$p = \lfloor \log_2(q) \rfloor$":

$$\begin{array}{l} \text{if } p_j \cdots p_0 = 0 \text{ then } q_{W-1} \cdots q_1 = 0 \cdots 000 \\ \text{if } p_j \cdots p_0 = 1 \text{ then } q_{W-1} \cdots q_1 = 0 \cdots 001 \\ \text{if } p_j \cdots p_0 = 2 \text{ then } q_{W-1} \cdots q_2 = 0 \cdots 01 \\ \vdots \\ \text{if } p_j \cdots p_0 = W-1 \text{ then } q_{W-1} = 1 \\ p \leq W-1 \end{array}$$

For each "if", the "then ..." part has size $O(W)$, and the "if" condition multiplies the size by $O(j) = O(\log W)$. Since there are $W$ many "if" statements, each of size $O(W \log W)$, their total size is $O(W^2 \log W)$. The "$p \leq W-1$" part only contributes $+ O(W^2)$ to the size.

**Exercise 2.34.b:** We claim that for every assignment $a \in \{0,1\}^W$, there is a CNF expressing "$p \leq a$" with size $O(W + \log^2 a)$. With $a = W - 1$, we get size $O(W + \log^2 W) = O(W)$.

Now, we prove the claim. Define $j = \lfloor \log_2(a) \rfloor$ as the index of the most significant 1 in $a$, assuming $a \neq 0$. The CNF "$p_{W-1} \cdots p_{j+1} = 0 \cdots 0$ and $p_j \cdots p_0 \leq a_j \cdots a_0$" does the job. The "$p_{W-1} \cdots p_{j+1} = 0 \cdots 0$" part has size $O(W)$. The "$p_j \cdots p_0 \leq a_j \cdots a_0$" part has size $O((j+1)^2) = O(\log^2 a)$ using the construction from §2.8.1.

If $a = 0$, then "$p \leq a$" just means "$p = 0 \cdots 0$", which has size $O(W)$.

**Exercise 2.35:** There are $\leq W$ options for $i$, and $\leq W$ options for $j$, and $O(1)$ options for $a_i b_i c_i$, and each clause has width $O(W)$, so the size is $O(W \cdot W \cdot 1 \cdot W) = O(W^3)$.

To prove that this CNF expresses "$p = q + r$", we show that for every assignment $(p, q, r) = (a, b, c)$, we have $a \neq b + c$ iff at least one clause is falsified.

$\Rightarrow$: Suppose $a \neq b + c$, and let $i$ be the least significant index such that $a_i \neq (b + c)_i$.

- Suppose $b_j c_j \in \{01, 10\}$ for all $j < i$. Then $a_{i-1} \cdots a_0 = 1 \cdots 1$ by the minimality of $i$, and $a_i \neq (b+c)_i = b_i \oplus c_i$ since there's no carry into position $i$. Thus $(a, b, c)$ falsifies the clause:

$$\text{“}p_i q_i r_i \neq a_i b_i c_i \text{ or } p_{i-1} \cdots p_0 \neq 1 \cdots 1\text{”}$$

- Suppose $b_j c_j \in \{00, 11\}$ for some $j < i$, and consider the greatest such $j$. Thus $b_k c_k \in \{01, 10\}$ for all $k$ with $i > k > j$.
  - Suppose $b_j c_j = 00$. Then there's no carry into position $j + 1$, so $a_{i-1} \cdots a_{j+1} = 1 \cdots 1$ by the minimality of $i$, and $a_i \neq (b + c)_i = b_i \oplus c_i$ since there's no carry into position $i$. Thus $(a, b, c)$ falsifies the clause:

$$\text{“}p_i q_i r_i \neq a_i b_i c_i \text{ or } p_{i-1} \cdots p_{j+1} \neq 1 \cdots 1 \text{ or } q_j r_j \neq 00\text{”}$$

  - Suppose $b_j c_j = 11$. Then there's a carry into position $j + 1$, which is propagated all the way into position $i$, so $a_{i-1} \cdots a_{j+1} = 0 \cdots 0$ by the minimality of $i$, and $a_i \neq (b + c)_i = 1 \oplus b_i \oplus c_i$, that is, $a_i = b_i \oplus c_i$. Thus $(a, b, c)$ falsifies the clause:

$$\text{“}p_i q_i r_i \neq a_i b_i c_i \text{ or } p_{i-1} \cdots p_{j+1} \neq 0 \cdots 0 \text{ or } q_j r_j \neq 11\text{”}$$

$\Leftarrow$: Suppose at least one clause is falsified. We don't need separate notation for the $a_i b_i c_i$ in the clause definition and the $a_i b_i c_i$ of our assignment $(a, b, c)$, because these must be equal—otherwise the clause would be satisfied by the "$p_i q_i r_i \neq a_i b_i c_i$" part.

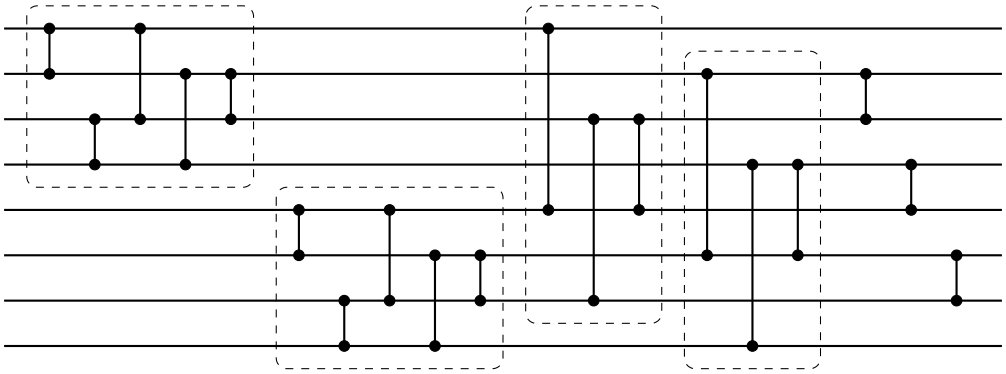
- Suppose "$p_{i-1} \cdots p_0 \neq 1 \cdots 1$" is falsified, where $a_i \neq b_i \oplus c_i$. That is, $a_{i-1} \cdots a_0 = 1 \cdots 1$.
  - Suppose $b_k c_k \in \{01, 10\}$ for all $k < i$, so there's no carry into position $i$. Then $a \neq b + c$ since $a_i \neq b_i \oplus c_i = (b + c)_i$.
  - Suppose $b_k c_k \in \{00, 11\}$ for some $k < i$, and consider the least such $k$. Then $b_\ell c_\ell \in \{01, 10\}$ for all $\ell < k$, so there's no carry into position $k$. Then $a \neq b + c$ since $a_k = 1 \neq b_k \oplus c_k = (b + c)_k$.
- Suppose "$p_{i-1} \cdots p_{j+1} \neq 1 \cdots 1$ or $q_j r_j \neq 00$" is falsified, where $a_i \neq b_i \oplus c_i$. That is, $a_{i-1} \cdots a_{j+1} = 1 \cdots 1$ and $b_j c_j = 00$.
  - Suppose $b_k c_k \in \{01, 10\}$ for all $k$ with $i > k > j$, so there's no carry into position $i$. Then $a \neq b + c$ since $a_i \neq b_i \oplus c_i = (b + c)_i$.
  - Suppose $b_k c_k \in \{00, 11\}$ for some $k$ with $i > k > j$, and consider the least such $k$. Then $b_\ell c_\ell \in \{01, 10\}$ for all $\ell$ with $k > \ell > j$, so there's no carry into position $k$. Then $a \neq b + c$ since $a_k = 1 \neq b_k \oplus c_k = (b + c)_k$.
- Suppose "$p_{i-1} \cdots p_{j+1} \neq 0 \cdots 0$ or $q_j r_j \neq 11$" is falsified, where $a_i = b_i \oplus c_i$. That is, $a_{i-1} \cdots a_{j+1} = 0 \cdots 0$ and $b_j c_j = 11$.

- Suppose $b_k c_k \in \{01, 10\}$ for all $k$ with $i > k > j$, so there's a carry into position $i$. Then $a \neq b + c$ since $a_i = b_i \oplus c_i \neq 1 \oplus b_i \oplus c_i = (b + c)_i$.
- Suppose $b_k c_k \in \{00, 11\}$ for some $k$ with $i > k > j$, and consider the least such $k$. Then $b_\ell c_\ell \in \{01, 10\}$ for all $\ell$ with $k > \ell > j$, so there's a carry into position $k$. Then $a \neq b + c$ since $a_k = 0 \neq 1 \oplus b_k \oplus c_k = (b + c)_k$.

**Exercise 2.36:** By definition, $\mathsf{NTIME}[N] \subseteq \mathsf{P}$ means that for every $A \in \mathsf{NTIME}[N]$ there exists $d$ such that $A \in \mathsf{TIME}[N^d]$. We need to swap the quantifiers, so $d$ doesn't depend on $A$. We accomplish this using NP-completeness.

Assume $\mathsf{NTIME}[N] \subseteq \mathsf{P}$. Since $\textsc{Sat} \in \mathsf{NTIME}[N]$, there exists $c$ such that $\textsc{Sat} \in \mathsf{TIME}[N^c]$. By Corollary 2.33, $\mathsf{NTIME}[N] \subseteq \mathsf{TIME}[N^c \operatorname{polylog} N] \subseteq \mathsf{TIME}[N^{c+1}]$. So we can take $d = c + 1$.

**Exercise 2.37:**



**Exercise 2.38:** As noted in §2.8.5, we may assume the array contains $n$ distinct numbers. Also, since the exact values don't matter—only the relative order matters—we may assume the array contains the numbers $[n] = \{1, 2, \ldots, n\}$ (the smallest number might as well be 1, and so on). Thus, the given comparison network is not a sorting network iff there exists a permutation of $[n]$ that it fails to sort. Here's a poly-time verifier:

```
given a purported witness w:
    if w isn't a permutation of [n]: reject
    simulate the given comparison network with w as the initial array contents
    if the array contains 1, 2, ..., n in order at the end: reject
    else: accept
```

The point is that a witness (a counterexample to being a sorting network) never requires numbers that are too large to handle in poly time. In fact, it can be shown that it also suffices to use some $w \in \{0,1\}^n$ as the witness (the *zero-one principle*), but that's not necessary for showing NOT A SORTING NETWORK $\in$ NP.

**Exercise 2.39:** $\Leftarrow$: Assume $\mathsf{coNP} = \mathsf{NP}$. Since $B \in \mathsf{NP}$, we have $B \in \mathsf{coNP}$.

$\Rightarrow$: Assume $B \in \mathsf{coNP}$. For every $A \in \mathsf{NP}$, we have $A \leq_p^m B$ and thus $\overline{A} \leq_p^m \overline{B}$ by the same reduction. This implies $A \in \mathsf{coNP}$ since a verifier for $\overline{A}$ can run the mapping reduction and then run a verifier for $\overline{B}$ on the query. Thus $\mathsf{NP} \subseteq \mathsf{coNP}$. By complementing both sides, we also have $\mathsf{coNP} \subseteq \mathsf{cocoNP} = \mathsf{NP}$. We conclude that $\mathsf{coNP} = \mathsf{NP}$.

**Exercise 2.40:** Abbreviate WHICH IS SATISFIABLE as $B$. Since $B(\varphi_0, \varphi_1) = \text{SAT}(\varphi_1)$, we have $B \in \text{NP}$ by a verifier that ignores $\varphi_0$ and just runs a SAT verifier on $\varphi_1$. Since $\overline{B}(\varphi_0, \varphi_1) = \text{SAT}(\varphi_0)$, we have $\overline{B} \in \text{NP}$ by a verifier that ignores $\varphi_1$ and just runs a SAT verifier on $\varphi_0$. Thus $B \in \text{NP} \cap \text{coNP}$.

To see that $B$ is $(\text{NP} \cap \text{coNP})$-hard, consider any $A \in \text{NP} \cap \text{coNP}$. Since SAT is NP-hard, we have $A \leq_p^m \text{SAT}$ by some reduction $F_1$ and $\overline{A} \leq_p^m \text{SAT}$ by some reduction $F_0$. We have $A \leq_p^m B$ by mapping $x$ to $(\varphi_0, \varphi_1) = (F_0(x), F_1(x))$. This is correct because: If $A(x) = 1$ then $\varphi_1$ is satisfiable and $\varphi_0$ isn't and thus $(\varphi_0, \varphi_1)$ is a valid input to $B$ and $B(\varphi_0, \varphi_1) = 1$. If $A(x) = 0$ then $\varphi_0$ is satisfiable and $\varphi_1$ isn't and thus $(\varphi_0, \varphi_1)$ is a valid input to $B$ and $B(\varphi_0, \varphi_1) = 0$.

**Exercise 2.41:** Suppose NP∪coNP = PSPACE. To see that PSPACE ⊆ NP (hence NP = PSPACE), consider any $A \in \mathsf{PSPACE}$. Define a decision problem $B$ that takes as input $(x, b)$ where $x$ is a valid input to $A$ and $b$ is a bit, and $B(x, b) = A(x) \oplus b$. Then $B \in \mathsf{PSPACE}$ by an algorithm that runs a poly-space algorithm to compute $A(x)$ and then xors the result with $b$. Thus by assumption, $B \in \mathsf{NP} \cup \mathsf{coNP}$. If $B \in \mathsf{NP}$ then $A \in \mathsf{NP}$ by a verifier that, on input $x$, runs a poly-time verifier for $B$ on input $(x, 0)$, which works since $B(x, 0) = A(x)$. If $B \in \mathsf{coNP}$ (that is, $\overline{B} \in \mathsf{NP}$) then $A \in \mathsf{NP}$ by a verifier that, on input $x$, runs a poly-time verifier for $\overline{B}$ on input $(x, 1)$, which works since $\overline{B}(x, 1) = \overline{A(x) \oplus 1} = A(x)$. Either way, $A \in \mathsf{NP}$.

**Exercise 2.42:** Assume $A \leq_p^o B$ and $B \in \mathsf{NP} \cap \mathsf{coNP}$. Say $B \in \mathsf{NP}$ by a verifier $V_1$, and $\overline{B} \in \mathsf{NP}$ by a verifier $V_0$. Then $A \in \mathsf{NP}$ by a verifier $U$ that views the purported witness as $w = (q, a_1, w_1, a_2, w_2, \ldots, a_q, w_q)$ where $q$ is the number of oracle queries the reduction makes, $a_i \in \{0,1\}$ is the answer to the $i^{\text{th}}$ query, and $w_i$ is a witness for the $i^{\text{th}}$ query. The size of each $w_i$ is $cN^d$ for integers $c, d$ big enough that $cN^d$ is larger than the running time of both $V_1$ and $V_0$ on any query the reduction might make on an input of size $N$.

```
U(x; q, a_1, w_1, ..., a_q, w_q):
    for i ← 1, ..., q:
        continue the execution of the reduction on input x
        if it terminates without making another query: reject
        let y_i be the i^th query
        run V_{a_i}(y_i; w_i)
        if it rejected: reject
        tell the reduction that a_i is the answer to the query
    continue the execution of the reduction on input x
    if it makes another query: reject
    output the same bit the reduction does
```

$U$ also rejects if the word $a_i$ doesn't fit in a single bit, or if some word of $w_i$ doesn't fit in the word size for $V_{a_i}(y_i)$.

$U$ runs in poly time. To see that $U$ is correct, we prove that for every valid input $x$:

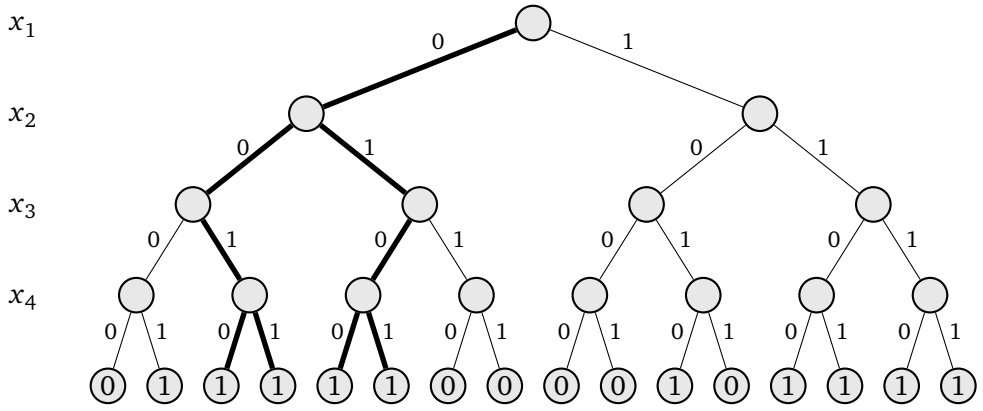
$$A(x) = 1 \iff \big(\exists w : \ U(x; w) \text{ accepts}\big)$$

$\Rightarrow$: Suppose $A(x) = 1$. Let $w = (q, a_1, w_1, \ldots, a_q, w_q)$ be the actual number of queries $q$, the actual query answers $a_i = B(y_i)$, and actual witnesses $w_i$ for the queries. Then $U(x; w)$ runs the reduction exactly as if the answers came from a real oracle for $B$, and it doesn't reject before the reduction finishes: It doesn't reject due to the "number of queries" checks, and it doesn't reject due to $V_{a_i}(y_i; w_i)$ rejecting, because $a_i = B(y_i)$ means there indeed exists $w_i$ such that $V_{a_i}(y_i; w_i)$ accepts (by completeness of $V_1$ and $V_0$). Thus $U(x; w)$ accepts because the reduction accepts when $A(x) = 1$ and the queries are answered according to $B$.

$\Leftarrow$: Suppose there exists $w$ such that $U(x; w)$ accepts. Then $w$ has the form $(q, a_1, w_1, \ldots, a_q, w_q)$ where each $a_i$ is a single bit. Since $V_{a_i}(y_i; w_i)$ accepts, we must have $a_i = B(y_i)$ (by soundness of $V_1$ and $V_0$). Thus $a_1, \ldots, a_q$ are the actual answers to the queries the reduction makes, and $q$ is the actual number of queries (since otherwise $U(x; w)$ would reject if the reduction tried to make a $(q+1)^{\text{st}}$ query or terminated after fewer than $q$ queries). Thus $U(x; w)$ runs the reduction to completion, exactly as the reduction would run if the answers came from a real oracle for $B$. Since $U(x; w)$ accepts, the reduction accepts input $x$, so $A(x) = 1$.

Since $A \leq_p^o B$ implies $\overline{A} \leq_p^o B$ by flipping the reduction's output, we also have $\overline{A} \in \mathsf{NP}$ and thus $A \in \mathsf{coNP}$.

**Exercise 3.1:**




P1 has exactly one winning strategy, depicted with the bold edges: P1 plays $x_1 = 0$, and then after P2 plays $x_2$, P1 responds by making $x_3 \neq x_2$.

**Exercise 3.2.a:** Given a monotone CNF $\varphi(x_1 \cdots x_n)$, we accept iff each clause contains at least one $\exists$-quantified variable (that is, with odd index). To show this log-space algorithm is correct, we argue that it accepts iff $\exists x_1 \cdots \forall x_n : \ \varphi(x_1 \cdots x_n) = 1$:

$\Rightarrow$: If each clause contains at least one $\exists$-quantified variable, which is a positive literal, then P1 can win the game by assigning 1 to all these variables.

$\Leftarrow$: If some clause contains only $\forall$-quantified variables, which are positive literals, then P2 can win the game by assigning 0 to all these variables.

**Exercise 3.2.b:** For a CNF $\varphi$, say a variable $x_i$ is *pure* when $x_i$ occurs only positively or only negatively throughout $\varphi$—that is, there do not exist both a clause with $x_i$ and a clause with $\overline{x_i}$.

Map a 3-CNF $\varphi(x_1 \cdots x_n)$ in which each clause has at least one pure variable to a 2-CNF $\psi(x_1 \cdots x_n)$ as follows: For each clause of $\varphi$, if it contains an $\exists$-quantified pure variable then we delete the clause, and otherwise we pick one $\forall$-quantified pure variable in the clause and shrink the clause by removing that literal. To show this log-space mapping reduction is correct, we argue that $\exists x_1 \cdots \forall x_n : \ \varphi(x_1 \cdots x_n) = 1$ iff $\exists x_1 \cdots \forall x_n : \ \psi(x_1 \cdots x_n) = 1$:

$\Rightarrow$: Assume P1 has a winning strategy for $\varphi$. Then P1 can apply the same strategy to $\psi$, except that when P2 assigns the "good" value to a pure variable ($x_i = 1$ and $x_i$ occurs positively, or $x_i = 0$ and $x_i$ occurs negatively), P1 continues as if P2 had assigned the "bad" value. This guarantees that each clause of $\psi$ is satisfied because the corresponding clause in $\varphi$ is satisfied by a literal whose variable is not pure, and thus that literal remains in the clause in $\psi$ and is assigned the same value.

$\Leftarrow$: Assume P1 has a winning strategy for $\psi$. Then P1 can apply the same strategy to $\varphi$, but assigning the "good" value to each $\exists$-quantified pure variable. This guarantees that $\varphi$ is satisfied because each clause that was deleted from $\psi$ is satisfied by a literal with an $\exists$-quantified pure variable, and each other clause is satisfied by the same literal that satisfies the corresponding clause in $\psi$.

**Exercise 3.2.c:** We use a subroutine similar to the one for QSAT $\in$ PSPACE (Lemma 3.1) but exploiting the fact that if $x_i$ appears only positively or only negatively in the contraction $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$, then we need not recurse on both possible values of $x_i$, since the current player would only make the obvious move: If it's P1's turn, he would assign $x_i$ to satisfy the clauses in which it appears. If it's P2's turn, he would assign $x_i$ so as not to satisfy the clauses in which it appears.

wins$(i, a_1 \cdots a_{i-1})$:
  if $i = n+1$: evaluate and return $\varphi(a_1 \cdots a_n)$
  if $i$ is odd and $x_i$ isn't in $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$, or $i$ is even and $\overline{x_i}$ isn't in $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$:
    return $\neg$ wins$(i+1, a_1 \cdots a_{i-1} 0)$
  if $i$ is odd and $\overline{x_i}$ isn't in $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$, or $i$ is even and $x_i$ isn't in $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$:
    return $\neg$ wins$(i+1, a_1 \cdots a_{i-1} 1)$
  return $\neg$ wins$(i+1, a_1 \cdots a_{i-1} 0) \vee \neg$ wins$(i+1, a_1 \cdots a_{i-1} 1)$

If wins$(i, a_1 \cdots a_{i-1})$ reaches the return statement on the last line, then $x_i$ and $\overline{x_i}$ both appear in $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$, so $\varphi|_{a_1 \cdots a_{i-1} 0 * \cdots *}$ and $\varphi|_{a_1 \cdots a_{i-1} 1 * \cdots *}$ both have fewer clauses than $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}$ since each assignment to $x_i$ satisfies at least one clause, which disappears from the contraction.

This implies that in the tree where nodes represent calls to wins, every root-to-leaf path has at most $c \log n$ nodes with siblings, since each such node has fewer clauses in the contraction of $\varphi$ than its parent does, and $\varphi$ starts with at most $c \log n$ clauses. Thus the tree has at most $2^{c \log n} = n^c$ leaves, and each root-to-leaf path has $n+1$ nodes, so the tree has poly $n$ many nodes. Each node contributes poly $n$ time, so the overall time efficiency is poly $n$ = poly $N$.

**Exercise 3.3:** We already showed that if $s$ = Lima then P1 has a winning strategy, and if $s$ = Montreal then P2 has a winning strategy.

If $s$ = Seoul then P1 has a winning strategy: P1 should go to Lagos. Then P2 must go to Stockholm, and then P1 goes to Montreal, and then P2 must go to Lima, and then P1 should go to Athens, at which point P2 loses since the only outgoing edges lead to Seoul and Stockholm, which were both already visited.

If $s$ = Lagos then P1 has a winning strategy: P1 should go to Seoul. Then P2 must go to Lima, and then P1 should go to Athens, and then P2 must go to Stockholm, and then P1 goes to Montreal, at which point P2 loses since the only outgoing edges lead to Lima and Lagos, which were both already visited.

If $s$ = Stockholm then P1 has a winning strategy: P1 goes to Montreal. If P2 goes to Lagos, then P1 goes to Seoul, and then P2 must go to Lima, and then P1 goes to Athens, at which point P2 loses since the only outgoing edges lead to Seoul and Stockholm, which were both already visited. On the other hand, if P2 goes from Montreal to Lima, then P1 should go to Athens, and then P2 must go to Seoul, and then P1 goes to Lagos, at which point P2 loses since the only outgoing edges lead to Seoul and Stockholm, which were both already visited.

If $s$ = Atlanta then P2 has a winning strategy: P1 must go to Athens. Then P2 should go to Stockholm. Then P1 must go to Montreal, and then P2 goes to Lima, at which point P1 loses since the only outgoing edges lead to Athens and Atlanta, which were both already visited.

If $s$ = Athens then P2 has a winning strategy: First, suppose P1 goes to Stockholm, and then P2 goes to Montreal. If P1 goes to Lima, then P2 goes to Atlanta, at which point P1 loses since the only outgoing edges lead to Athens and Atlanta, which were both already visited. On the other hand, if P1 goes from Montreal to Lagos, then P2 goes to Seoul, and then P1 must go to Lima, and then P2 goes to Atlanta, at which point P1 loses since the only outgoing edges lead to Athens and Atlanta, which were both already visited. Now, suppose P1 goes from Athens to Seoul at the beginning. Then P2 should go to Lagos. Then P1 must go to Stockholm, and then P2 goes to Montreal, and then P1 must go to Lima, and then P2 goes to Atlanta, at which point P1 loses since the only outgoing edges lead to Athens and Atlanta, which were both already visited.

**Exercise 3.4:** We design a subroutine "wins" that has access to the input $G$, takes as an argument an independent set $S$ consisting of all nodes played so far, and returns the bit indicating whether the player whose turn it is (P1 if $|S|$ is even, P2 if $|S|$ is odd) has a winning strategy for the rest of the game. To solve Node Kayles, call wins($\emptyset$).

The current player can win iff they can make a move so the other player cannot then win:

```
wins(S):
    for each node v ∉ S such that u ∉ S for all edges {u, v}:
        if wins(S ∪ {v}) = 0: return 1
    return 0
```

This recursive subroutine maintains the invariant that wins($S$) $= 1$ iff in the game on $G$ with all nodes of $S$ and their neighbors removed, the player who goes first (which may be P1 or P2 from the overall game on $G$) has a winning strategy. This holds for the base cases when no nodes remain: wins($S$) $= 0$ because the current player has already lost. For non-base cases, the invariant is maintained because wins($S$) $= 1$ iff there exists a playable node $v$ (a move for the current player) such that wins($S \cup \{v\}$) $= 0$, which means whoever played $v$ has a winning strategy (with $S$ and $v$ and all their neighbors excluded) since their opponent (who has the next turn) doesn't.

Since $|S| + 1$ is the depth of a call, and a base case must be reached when all $n$ nodes are in $S$ (or sooner), the overall recursion depth is $\leq n + 1 \leq N$. Each stack frame needs poly$N$ bits for the argument and other local data. The space efficiency is:

$$\big(N \text{ stack frames at a time}\big) \cdot \big(\text{poly}N \text{ bits per stack frame}\big) \;=\; \text{poly}N \text{ bits}$$

**Exercise 3.5:** This problem is in PSPACE since QSAT is. We show QSAT $\leq_\ell^m$ this problem by mapping $\varphi(x_1 \cdots x_n)$ to $\psi(x_1 \cdots x_n)$ where $\psi$ is $\varphi$ except for each dummy variable $\exists x_i$ of $\varphi$, we have a width-1 clause $(x_i)$ in $\psi$, and for each dummy variable $\forall x_i$ of $\varphi$, we add the literal $x_i$ to some clause from $\varphi$ (it doesn't matter which one) in $\psi$. Note that $\psi(x_1 \cdots x_n)$ has no dummy variables. To show this log-space mapping reduction is correct, we argue that $\exists x_1 \cdots \forall x_n : \varphi(x_1 \cdots x_n) = 1$ iff $\exists x_1 \cdots \forall x_n : \psi(x_1 \cdots x_n) = 1$:

$\Rightarrow$: Assume P1 has a winning strategy for $\varphi$. Then P1 can apply the same strategy to $\psi$, but assigning $x_i = 1$ if $\exists x_i$ is a dummy variable of $\varphi$. This guarantees that $\psi$ is satisfied because each new clause $(x_i)$ is satisfied, and each clause that came from $\varphi$ remains satisfied—the addition of $\forall$-quantified dummy variables to the clause doesn't hurt, and the clause is unaffected by P1's strategy on $\exists$-quantified dummy variables.

$\Leftarrow$: Assume P1 has a winning strategy for $\psi$. Then P1 can apply the same strategy to $\varphi$, except that when P2 assigns $x_i = 1$ where $\forall x_i$ is a dummy variable, P1 continues as if P2 had assigned $x_i = 0$. This guarantees that each clause of $\varphi$ is satisfied because the corresponding clause in $\psi$ is satisfied even with all $\forall$-quantified dummy variables that were added to it (if any) assigned 0.

**Exercise 3.6.a:** Suppose $G$ has nodes $s, u, v, w$ and edges $(s,u),(s,v),(v,w)$. Then $(G,s)$ is a yes-input of GEOGRAPHY, and hence a no-input of $\overline{\text{GEOGRAPHY}}$, since P1 can go from $s$ to $u$, and then P2 is stuck. But $(G,s)$ is a yes-input of MISÈRE GEOGRAPHY since P1 can go from $s$ to $v$, and then P2 must go to $w$, and then P1 is stuck.

**Exercise 3.6.b:** Misère Geography $\in$ PSPACE by an algorithm like the one for showing Geography $\in$ PSPACE (Lemma 3.2) except that wins$(v_1, v_2, \ldots, v_i)$ returns 1 if there is no edge $(v_i, v)$ in $G$ such that $v \notin \{v_1, \ldots, v_i\}$.

We show Geography $\leq_\ell^m$ Misère Geography by mapping $(G, s)$ to $(G', s)$ where $G'$ is $G$ but with a new node $t$ that has no outgoing edges and has an incoming edge from each node of $G$. To show this log-space mapping reduction is correct, we argue that P1 has a winning strategy in the Geography game on $(G, s)$ iff P1 has a winning strategy in the Misère Geography game on $(G', s)$:

$\Rightarrow$: If P1 has a winning strategy on $(G, s)$, P1 can apply the same strategy on $(G', s)$. Since P2 would get stuck in $G$, P2 will inevitably go to $t$ in $G'$, and so P1 is stuck in $G'$ and wins.

$\Leftarrow$: If P2 has a winning strategy on $(G, s)$, P2 can apply the same strategy on $(G', s)$. Since P1 would get stuck in $G$, P1 will inevitably go to $t$ in $G'$, and so P2 is stuck in $G'$ and wins.

**Exercise 3.7:** STRING COMPATIBLE GEOGRAPHY ∈ PSPACE since GEOGRAPHY ∈ PSPACE. We show GEOGRAPHY $\leq_\ell^m$ STRING COMPATIBLE GEOGRAPHY by mapping $(G,s)$ to $(G',s)$ where $G'$ is like $G$ except that each edge $e=(u,v)$ of $G$ becomes a path of three edges $(u,y_e),(y_e,z_e),(z_e,v)$ in $G'$, where $y_e$ and $z_e$ are new nodes. This $G'$ is string compatible because the only nodes that can have outneighbors in common are $z_e$ nodes for different edges $e$ of $G$, and each of those nodes has only one outneighbor. This log-space mapping reduction is correct because P1 has a winning strategy on $(G,s)$ iff P1 has a winning strategy on $(G',s)$: For each edge $e=(u,v)$ in $G$, P1 traversing $u \to v$ in $G$ is equivalent to P1,P2,P1 traversing $u \to y_e \to z_e \to v$ in $G'$, and P2 traversing $u \to v$ in $G$ is equivalent to P2,P1,P2 traversing $u \to y_e \to z_e \to v$ in $G'$.

**Exercise 3.8:** Here is such a poly-time oracle reduction:

query the oracle on input $\varphi|_{a_1 \cdots a_{i-1} * \cdots *}(x_i \cdots x_n)$, and reject if the oracle rejected
query the oracle on input $\varphi|_{a_1 \cdots a_{i-1} 00 * \cdots *}(x_{i+2} \cdots x_n)$, and output 1 if the oracle rejected
query the oracle on input $\varphi|_{a_1 \cdots a_{i-1} 01 * \cdots *}(x_{i+2} \cdots x_n)$, and output 1 if the oracle rejected
output 0

The reduction rejects iff there is no such $a_i$. If the reduction outputs 0, then this is correct because

$$\exists x_{i+2} \cdots \forall x_n : \ \varphi(a_1 \cdots a_{i-1} 00 x_{i+2} \cdots x_n) = 1$$

and

$$\exists x_{i+2} \cdots \forall x_n : \ \varphi(a_1 \cdots a_{i-1} 01 x_{i+2} \cdots x_n) = 1$$

since the oracle accepted the second and third queries, which means:

$$\forall x_{i+1} \ \exists x_{i+2} \cdots \forall x_n : \ \varphi(a_1 \cdots a_{i-1} 0 x_{i+1} \cdots x_n) = 1$$

If the reduction outputs 1, then this is correct because

$$\forall x_{i+1} \ \exists x_{i+2} \cdots \forall x_n : \ \varphi(a_1 \cdots a_{i-1} 0 x_{i+1} \cdots x_n) = 1$$

is false, which means

$$\forall x_{i+1} \ \exists x_{i+2} \cdots \forall x_n : \ \varphi(a_1 \cdots a_{i-1} 1 x_{i+1} \cdots x_n) = 1$$

must be true since the reduction didn't reject.

**Exercise 3.9:** We claim that P2 has a winning strategy iff $\sum_{\text{even } i} x_i = t$ and $x_i = x_{i-1}$ for all even $i$. A log-space algorithm can check whether this holds, using a log-space algorithm for Iterated Addition. Now, we prove the claim:

$\Leftarrow$: Assume $\sum_{\text{even } i} x_i = t$ and $x_i = x_{i-1}$ for all even $i$. Here is a winning strategy for P2: For each even $i$, P2 includes $i \in I$ if and only if P1 didn't include $i-1 \in I$. Thus for each even $i$, exactly one of $i-1$ or $i$ is in $I$, so at the end, $\sum_{i \in I} x_i = \sum_{\text{even } i} x_i = t$ and P2 wins.

$\Rightarrow$: Assume either $\sum_{\text{even } i} x_i \neq t$ or $x_i \neq x_{i-1}$ for some even $i$.

First, assume $\sum_{\text{even } i} x_i \neq t$ and $x_i = x_{i-1}$ for all even $i$. If $\sum_{\text{even } i} x_i < t$ then P1 can win by including no odd $i \in I$, guaranteeing that $\sum_{i \in I} x_i \leq \sum_{\text{even } i} x_i < t$. If $\sum_{\text{even } i} x_i > t$ then P1 can win by including all odd $i \in I$, guaranteeing that $\sum_{i \in I} x_i \geq \sum_{\text{odd } i} x_i = \sum_{\text{even } i} x_i > t$.

Now, assume $x_i \neq x_{i-1}$ for some even $i$. Consider the greatest even $i^*$ such that $x_{i^*} \neq x_{i^*-1}$. Here is a winning strategy for P1: Play arbitrarily until it's time to pick whether to include $i^*-1 \in I$. Since $x_{i^*} \neq x_{i^*-1}$ and $x_{i^*-1} > 0$ and $x_{i^*} > 0$, the four numbers $0$, $x_{i^*-1}$, $x_{i^*}$, $x_{i^*-1} + x_{i^*}$ are all distinct. Thus $t - \sum_{i^*-2 \geq i \in I} x_i - \sum_{\text{even } i > i^*} x_i$ is either not in $\{0, x_{i^*}\}$ or not in $\{x_{i^*-1}, x_{i^*-1} + x_{i^*}\}$ (possibly in neither). In the former case, P1 doesn't include $i^* - 1 \in I$. In the latter case, P1 includes $i^*-1 \in I$. After P2 picks whether to include $i^* \in I$, it is guaranteed that $\sum_{\text{even } i > i^*} x_i \neq t - \sum_{i^* \geq i \in I} x_i$. Then P1 plays like in the previous paragraph: If $\sum_{\text{even } i > i^*} x_i < t - \sum_{i^* \geq i \in I} x_i$ then P1 can win by including no odd $i > i^*$ in $I$, guaranteeing that $\sum_{i \in I} x_i \leq \sum_{i^* \geq i \in I} x_i + \sum_{\text{even } i > i^*} x_i < t$. If $\sum_{\text{even } i > i^*} x_i > t - \sum_{i^* \geq i \in I} x_i$ then P1 can win by including all odd $i > i^*$ in $I$, guaranteeing that $\sum_{i \in I} x_i \geq \sum_{i^* \geq i \in I} x_i + \sum_{\text{odd } i > i^*} x_i = \sum_{i^* \geq i \in I} x_i + \sum_{\text{even } i > i^*} x_i > t$ since $x_i = x_{i-1}$ for all even $i > i^*$.

**Exercise 3.10:** We can turn any streaming verifier for $A$ with $O(T)$ time efficiency into an ordinary verifier for $A$ with $O(T)$ time efficiency by using a register *waddr* to remember the next witness segment address to read from. Instead of sequential-access read-witness, the ordinary verifier can read-witness from address *waddr* and then increment *waddr*. The number of computation steps at most doubles, and the highest address read from the witness segment is still $O(T)$.

We can turn any ordinary verifier for $A$ with $O(T)$ time efficiency into a streaming verifier for $A$ with $O(T)$ time efficiency by using a register *waddr* to remember the highest witness segment address read from so far ("high water mark") and an extra work segment to remember the witness up through address *waddr*. Instead of random-access read-witness, the streaming verifier can check if the address is $\leq$ *waddr* and if so, load the word from the "witness work segment," and if not, read-witness enough words to append to the "witness work segment" and bring *waddr* up to the desired address. Since the ordinary verifier only reads from the first $O(T)$ words of the witness segment, the streaming verifier only spends $O(T)$ steps dealing with its "witness work segment."

**Exercise 3.11.a:** To show the reduction is correct, we prove that $G$ has no proper 2-coloring iff there exists $u$ such that $u_{\text{right}}$ is reachable from $u_{\text{left}}$ in $G_{\text{bip}}$ (that is, the reduction rejects).

$\Rightarrow$: Suppose $G$ has no proper 2-coloring. Then $G$ has an odd-length cycle (Theorem 2.2). Pick an arbitrary node $u$ on this cycle. As we traverse the edges from $u$ around to $u$, the corresponding sequence of edges in $G_{\text{bip}}$ forms a path from $u_{\text{left}}$ to $u_{\text{right}}$. Thus $u_{\text{right}}$ is reachable from $u_{\text{left}}$ in $G_{\text{bip}}$.

$\Leftarrow$: Suppose there is a path from $u_{\text{left}}$ to $u_{\text{right}}$. This path must have odd length. The corresponding sequence of edges in $G$ forms an odd-length closed walk containing $u$. Thus $G$ has no proper 2-coloring since otherwise the node colors would have to alternate as we traverse the closed walk, leading to the contradiction that $u$ has the opposite color as itself. (Theorem 2.2 only talks about cycles, but it also works for closed walks.)

The reduction takes log space since the loop just needs to remember $u$, and the construction of $G_{\text{bip}}$ is simple enough to write down in log space.

**Exercise 3.11.b:** Suppose $G$'s set of nodes is $\{1, 2, \ldots, n\}$. Map $G$ to the graph $G'$ that has $n$ copies of $G_{\text{bip}}$, and a node $s$ adjacent to $i_{\text{left}}$ in the $i^{\text{th}}$ copy for each $i$, and a node $t$ adjacent to $i_{\text{right}}$ in the $i^{\text{th}}$ copy for each $i$. The reduction only needs log space, and it's correct since $G$ has no proper 2-coloring iff there exists $i$ such that $i_{\text{right}}$ is reachable from $i_{\text{left}}$ in (the $i^{\text{th}}$ copy of) $G_{\text{bip}}$ (Exercise 3.11.a) iff $t$ is reachable from $s$ in $G'$.

**Exercise 3.11.c:** Map formula $\varphi$ to graph $G$ where:

- For each variable $x_i$ that appears in $\varphi$, $G$ has a corresponding node.
- For each constraint of the form $(x_i \oplus x_j)$ or $(\overline{x_i} \oplus \overline{x_j})$, $G$ has an edge $\{x_i, x_j\}$.
- For the $k^{\text{th}}$ constraint of the form $(x_i \oplus \overline{x_j})$ or $(\overline{x_i} \oplus x_j)$, $G$ has a fresh node $y_k$ and two edges $\{x_i, y_k\}$ and $\{y_k, x_j\}$.

This reduction only needs log space, and it's correct because $\varphi$ has a satisfying assignment iff $G$ has a proper 2-coloring:

$\Leftarrow$: Suppose $G$ has a proper 2-coloring, and call the colors 0 and 1. Then the 2-color assignment to the $x$ nodes is an assignment to $\varphi$'s variables, and it satisfies $\varphi$ because:

- Each constraint of the form $(x_i \oplus x_j)$ or $(\overline{x_i} \oplus \overline{x_j})$ is satisfied since $x_i$ and $x_j$ are assigned opposite bits since the edge $\{x_i, x_j\}$ is multicolored.
- The $k^{\text{th}}$ constraint of the form $(x_i \oplus \overline{x_j})$ or $(\overline{x_i} \oplus x_j)$ is satisfied since $x_i$ and $x_j$ are assigned the same bit since the edges $\{x_i, y_k\}$ and $\{y_k, x_j\}$ are multicolored.

$\Rightarrow$: Suppose $\varphi$ has a satisfying assignment. View this as a 2-color assignment to the $x$ nodes of $G$. Each $\{x_i, x_j\}$ edge is multicolored since the corresponding constraint $(x_i \oplus x_j)$ or $(\overline{x_i} \oplus \overline{x_j})$ is satisfied. Since the $k^{\text{th}}$ constraint of the form $(x_i \oplus \overline{x_j})$ or $(\overline{x_i} \oplus x_j)$ is satisfied, $x_i$ and $x_j$ have the same color and thus we can assign $y_k$ the opposite color so the edges $\{x_i, y_k\}$ and $\{y_k, x_j\}$ are multicolored.

Since 2-Xor Sat $\leq_\ell^m$ Graph 2-Coloring $\in$ L, we conclude 2-Xor Sat $\in$ L (Lemma 1.15).

**Exercise 3.12:** STRONG CONNECTIVITY $\in$ NL by a log-space streaming verifier that loops through all ordered pairs of nodes $u, v$ and checks that the witness demonstrates a walk from $u$ to $v$ of length $\leq n-1$, as in the proof that DIRECTED REACHABILITY $\in$ NL. We show DIRECTED REACHABILITY $\leq_\ell^m$ STRONG CONNECTIVITY by mapping $(G, s, t)$ to the graph $G'$ containing $G$ and edges from all non-$s$ nodes to $s$ and edges from $t$ to all non-$t$ nodes. To show this log-space mapping reduction is correct, we argue that $G$ has a walk from $s$ to $t$ iff $G'$ is strongly connected:

$\Rightarrow$: Assume $G$ has a walk from $s$ to $t$. To see that $G'$ is strongly connected, we consider any ordered pair of nodes $u, v$ and show that $G'$ has a walk from $u$ to $v$: Follow the edge from $u$ to $s$ (if $u \neq s$), then the assumed walk from $s$ to $t$, then the edge from $t$ to $v$ (if $v \neq t$).

$\Leftarrow$: Assume $G'$ is strongly connected. In particular, $G'$ has a walk from $s$ to $t$. There's no reason for this walk to use any new edge of $G'$, since taking a new edge to $s$ would just revisit $s$, and taking a new edge from $t$ would mean $t$ was already reached. Thus the assumed walk from $s$ to $t$ only uses edges of $G$.

**Exercise 3.13.a:** Since NL = coNL, it suffices to prove $\overline{\text{DAG DECISION}}$ is NL-complete.

$\overline{\text{DAG DECISION}} \in$ NL by a log-space streaming verifier that checks that the witness is a closed walk of length $\leq n$ (which exists iff $G$ has a cycle). We show DIRECTED REACHABILITY $\leq_\ell^m$ $\overline{\text{DAG DECISION}}$ by mapping $(G, s, t)$ to $G'$ which is the layered version of $G$ (with horizontal edges between copies of $t$) with $n$ layers (where $n$ is the number of nodes in $G$) and with an edge from $t'$ (the last layer's copy of $t$) to $s'$ (the first layer's copy of $s$). To show this log-space mapping reduction is correct, we argue that $G$ has a path from $s$ to $t$ iff $G'$ has a cycle:

$\Rightarrow$: Assume $G$ has a path from $s$ to $t$. Since the path has length $\leq n-1$ and $G'$ has $n$ layers, there is a corresponding path in $G'$ starting at $s'$ and using horizontal edges as needed to reach $t'$. Combining this path with the edge $(t', s')$ yields a cycle in $G'$.

$\Leftarrow$: Assume $G'$ has a cycle. The cycle must use the edge $(t', s')$ since $G'$ would certainly be acyclic without $(t', s')$. Removing $(t', s')$ from the cycle yields a path from $s'$ to $t'$, which corresponds to a path from $s$ to $t$ in $G$ (stopping as soon as we reach $t$, and trimming out any cycles).

**Exercise 3.13.b:** DIRECTED REACHABILITY ON A DAG $\in$ NL since DIRECTED REACHABILITY $\in$ NL. We show DIRECTED REACHABILITY $\leq_\ell^m$ DIRECTED REACHABILITY ON A DAG by mapping $(G, s, t)$ to $(G', s', t')$ as in the proof of Lemma 1.21 with $n$ layers (where $n$ is the number of nodes in $G$). The graph $G'$ is a dag since the layers provide a topological order—every edge goes from one layer to the next. This log-space mapping reduction is correct because $G$ has a path from $s$ to $t$ iff $G'$ has a path from $s'$ to $t'$, as we know from the proof of Lemma 1.21 and the fact that $G'$ has $n$ layers and every path in $G$ has length $\leq n-1$.

**Exercise 3.13.c:** UNDIRECTED SHORTEST PATH DECISION $\in$ NL for essentially the same reason that DIRECTED SHORTEST PATH DECISION $\in$ NL. We show DIRECTED REACHABILITY $\leq_\ell^m$ UNDIRECTED SHORTEST PATH DECISION by mapping $(G,s,t)$ to $(G',s',t',k)$ where $k = n-1$ (where $n$ is the number of nodes in $G$) and $G',s',t'$ are like in the proof of Lemma 1.21 with $k+1 = n$ layers but with undirected edges in $G'$. To show this log-space mapping reduction is correct, we argue that $G$ has a path from $s$ to $t$ iff $G'$ has a path from $s'$ to $t'$ of length $\leq k$:

$\Rightarrow$: A path from $s$ to $t$ has length $\leq k$ and yields a path from $s'$ to $t'$ of length $k$, as in the proof of Lemma 1.21.

$\Leftarrow$: Assume $G'$ has a path from $s'$ to $t'$ of length $\leq k$. Every edge of $G'$ goes between consecutive layers, so the path must have length $k$ and follow each edge in the intended direction (as if the edge were directed toward the higher-index layer) since otherwise getting from layer 0 to layer $k$ would require more than $k$ edges. Thus this undirected path yields a directed path from $s$ to $t$, as in the proof of Lemma 1.21.

**Exercise 3.14.a:** Since NL = coNL, it suffices to prove $\overline{\text{2-SAT}} \in \text{NL}$. From Exercise 2.12 we know a 2-CNF $\varphi$ is unsatisfiable iff there exists a variable $x_i$ such that the nodes $x_i$ and $\overline{x_i}$ are reachable from each other in $\varphi$'s implication graph. Also, the implication graph can be computed in log space. Letting $V$ be a log-space streaming verifier for DIRECTED REACHABILITY, we have $\overline{\text{2-SAT}} \in \text{NL}$ by this log-space streaming verifier:

```
for each variable x_i:
    run V on input (φ's implication graph, x_i, x̄_i) (recomputing each word when V reads it)
    run V on input (φ's implication graph, x̄_i, x_i) (recomputing each word when V reads it)
    if both runs of V accepted: accept
    if either run rejected: continue to the next variable
reject
```

If $\varphi$ is unsatisfiable, then for some $x_i$, there exist witnesses that make both runs of $V$ accept. If $\varphi$ is satisfiable, then for each $x_i$, all purported witnesses make at least one run of $V$ reject.

**Exercise 3.14.b:** Since NL = coNL, it suffices to prove $\overline{\text{2-SAT}}$ is NL-hard. We show DIRECTED REACHABILITY $\leq_\ell^m \overline{\text{2-SAT}}$ by mapping $(G, s, t)$ to $\varphi$ as follows. For each node $v$ in $G$, $\varphi$ has a variable $x_v$. For each edge $(u, v)$ in $G$, $\varphi$ has a clause $(\overline{x_u} \vee x_v)$, which is equivalent to $(x_u \Rightarrow x_v)$. Also, $\varphi$ has clauses $(x_s) \wedge (\overline{x_t})$. To turn $\varphi$ into a 2-CNF, we could apply Claim 2.12 to those width-1 clauses, or just use $(\overline{x_s} \vee \overline{x_t}) \wedge (x_s \vee \overline{x_t}) \wedge (x_s \vee x_t)$ instead. To show this log-space mapping reduction is correct, we argue that $G$ has a path from $s$ to $t$ iff $\varphi$ is unsatisfiable:

$\Rightarrow$: If $G$ has a path $s \to v_1 \to v_2 \to \cdots \to v_k \to t$ then the clauses

$$(x_s) \wedge (x_s \Rightarrow x_{v_1}) \wedge (x_{v_1} \Rightarrow x_{v_2}) \wedge \cdots \wedge (x_{v_k} \Rightarrow x_t) \wedge (\overline{x_t})$$

are unsatisfiable, because satisfying $(x_s)$ requires $x_s = 1$, and then satisfying $(x_s \Rightarrow x_{v_1})$ requires $x_{v_1} = 1$, and then satisfying $(x_{v_1} \Rightarrow x_{v_2})$ requires $x_{v_2} = 1$, and so on until $x_t = 1$, which doesn't satisfy $(\overline{x_t})$.

$\Leftarrow$: If $t$ is not reachable from $s$ in $G$, then $\varphi$ is satisfied by assigning 1 to all variables corresponding to nodes reachable from $s$, and assigning 0 to all other variables: $(x_s)$ is satisfied by $x_s = 1$ (since $s$ is reachable from itself), and $(\overline{x_t})$ is satisfied by $x_t = 0$ (since $t$ is not reachable from $s$), and each clause $(x_u \Rightarrow x_v)$ is satisfied either by $x_u = 0$ (if $u$ is unreachable from $s$) or by $x_v = 1$ (if $v$ is reachable from $s$)—it can't happen that $u$ is reachable but $v$ isn't.

**Exercise 3.15:** To prove (2 OUT OF 3)-SAT $\in$ NL, it suffices to prove (2 OUT OF 3)-SAT $\leq_\ell^m$ 2-SAT (Lemma 3.20). Map $\varphi$ to $\varphi'$ by replacing each width-3 clause $(\ell_i \vee \ell_j \vee \ell_k)$ with three width-2 clauses $(\ell_i \vee \ell_j) \wedge (\ell_i \vee \ell_k) \wedge (\ell_j \vee \ell_k)$. This log-space mapping reduction is correct because for every assignment, at least 2 out of the 3 literals $(\ell_i \vee \ell_j \vee \ell_k)$ are satisfied iff for every 2 of those literals, at least 1 out of the 2 is satisfied—and thus $\varphi$ is satisfied iff $\varphi'$ is satisfied.

To prove (2 OUT OF 3)-SAT is NL-hard, we show 2-SAT $\leq_\ell^m$ (2 OUT OF 3)-SAT by mapping $\varphi$ to $\varphi'$ which is $\varphi$ but with a fresh variable $x_0$ inserted into all the clauses. To show this log-space mapping reduction is correct, we argue that $\varphi$ has an assignment that satisfies at least one literal in each clause iff $\varphi'$ has an assignment that satisfies at least two literals in each clause:

$\Rightarrow$: If $\varphi$ has such an assignment then the same assignment, together with $x_0 = 1$, satisfies at least two literals in each clause of $\varphi'$.

$\Leftarrow$: If $\varphi'$ has such an assignment then the same assignment, ignoring $x_0$, satisfies at least one literal in each clause of $\varphi$.

**Exercise 3.16.a:** Let $A$ stand for Graph 3-Coloring with Forbidden Colors. To prove $A \in \mathsf{NL}$, it suffices to prove $A \leq_\ell^m$ 2-Sat (Lemma 3.20). Map $(G, f)$ (where $f$ is the tuple of forbidden colors) to $\varphi$ as follows. For each node $v$ in $G$, we have a variable $x_v$ in $\varphi$. The two possible values of $x_v$ correspond to the two allowed colors of $v$. For concreteness:

- If $f_v =$ red then $x_v = 0$ means $c(v) =$ green and $x_v = 1$ means $c(v) =$ blue.
- If $f_v =$ green then $x_v = 0$ means $c(v) =$ red and $x_v = 1$ means $c(v) =$ blue.
- If $f_v =$ blue then $x_v = 0$ means $c(v) =$ red and $x_v = 1$ means $c(v) =$ green.

For each edge $e = \{u, v\}$ in $G$, the constraint $c(u) \neq c(v)$ can be expressed as a 2-CNF $\varphi_e$ on the two variables $x_u$ and $x_v$ (Lemma 2.9). We let $\varphi = \bigwedge_e \varphi_e$ be the conjunction of these 2-CNFs over all edges in $G$. For every 3-color assignment $c$ with $c(v) \neq f_v$ for each $v$, it is proper iff the corresponding assignment to $x$ satisfies $\varphi$. Thus $G$ has such a proper 3-coloring iff $\varphi$ is satisfiable, so this log-space mapping reduction is correct.

**Exercise 3.16.b:** BIPARTITE 2-SAT $\in$ NL since 2-SAT $\in$ NL. To prove BIPARTITE 2-SAT is NL-hard, we show 2-SAT $\leq_\ell^m$ BIPARTITE 2-SAT by mapping $\varphi(x_1 \cdots x_n)$ to $\varphi'(x_1 \cdots x_n, y_1 \cdots y_n)$ as follows. For each $i \in [n]$, $\varphi'$ has the pair of clauses $(x_i \vee \overline{y_i}) \wedge (\overline{x_i} \vee y_i)$, which is equivalent to $(x_i \Leftrightarrow y_i)$. For each clause of $\varphi$, $\varphi'$ has the same clause but where one of the literals (it doesn't matter which) uses the corresponding $y$ variable instead of the $x$ variable. To show this log-space mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $\varphi'$ is satisfiable:

$\Rightarrow$: If $\varphi$ has a satisfying assignment, then $\varphi'$ is satisfied by the same assignment to $x$ and duplicated to $y$: Each $(x_i \Leftrightarrow y_i)$ is satisfied since $x_i$ and $y_i$ are assigned the same bit, and each other clause of $\varphi'$ is satisfied as in $\varphi$.

$\Leftarrow$: If $\varphi'$ has a satisfying assignment, then $\varphi$ is satisfied by the same assignment to $x$ (ignoring $y$): Each clause of $\varphi$ is satisfied because the corresponding clause of $\varphi'$ is satisfied and each $y_i$ is assigned the same bit as $x_i$ since $(x_i \Leftrightarrow y_i)$ is satisfied.

**Exercise 3.16.c:** We show Bipartite 2-Sat $\leq_\ell^m$ Graph 3-Coloring with Forbidden Colors by mapping $\varphi(x_1 \cdots x_n, y_1 \cdots y_n)$ to $(G, f)$ as follows. We use $\{0, 1, 2\}$ as the colors. $G$'s nodes correspond to all possible literals from $\varphi$'s variables. We let $f_{x_i} = f_{\overline{x_i}} = 2$ and $f_{y_i} = f_{\overline{y_i}} = 1$ for all $i \in [n]$. $G$ has an edge between each pair of nodes corresponding to opposite literals ($\{x_i, \overline{x_i}\}$ and $\{y_i, \overline{y_i}\}$ for all $i \in [n]$). For each clause of $\varphi$, $G$ has an edge between the nodes corresponding to the clause's literals. To show this log-space mapping reduction is correct, we argue that $\varphi$ is satisfiable iff $G$ has a proper 3-coloring $c$ with $c(v) \neq f_v$ for each node $v$:

$\Rightarrow$: Consider any assignment that satisfies $\varphi$. Color each node of $G$ according to the associated literal's value under this assignment, except use 2 instead of 1 for $y_i$ and $\overline{y_i}$ nodes. That is:

- If $x_i = 0$ then $c(x_i) = 0$ and $c(\overline{x_i}) = 1$.
- If $x_i = 1$ then $c(x_i) = 1$ and $c(\overline{x_i}) = 0$.
- If $y_i = 0$ then $c(y_i) = 0$ and $c(\overline{y_i}) = 2$.
- If $y_i = 1$ then $c(y_i) = 2$ and $c(\overline{y_i}) = 0$.

This 3-color assignment has $c(v) \neq f_v$ for each node $v$. It is proper since $c(x_i) \neq c(\overline{x_i})$ and $c(y_i) \neq c(\overline{y_i})$ for all $i \in [n]$, and for each clause of $\varphi$, its two literals have values $1, 0$ or $0, 1$ or $1, 1$ which correspond to the colors $1, 0$ or $0, 2$ or $1, 2$ for the associated edge's endpoints.

$\Leftarrow$: Consider any such $c$. For each $i \in [n]$, assign variables $x_i$ and $y_i$ the colors of their associated nodes, except assign $y_i = 1$ if $c(y_i) = 2$. Since the edges $\{x_i, \overline{x_i}\}$ and $\{y_i, \overline{y_i}\}$ are properly colored, the values of the negative literals $\overline{x_i}$ and $\overline{y_i}$ (under this assignment) are the colors of their associated nodes, except that $\overline{y_i} = 1$ if $c(\overline{y_i}) = 2$. For each clause of $\varphi$, the corresponding edge is properly colored, so the endpoints have colors $1, 0$ or $0, 2$ or $1, 2$ and thus the clause is satisfied since its literals have values $1, 0$ or $0, 1$ or $1, 1$.

**Exercise 3.17.a:** We design a subroutine "wins" that has access to the input $\varphi(x_1 \cdots x_n)$, takes as an argument a partial assignment $a \in \{0,1,*\}^n$, and returns the bit indicating whether the player whose turn it is (P1 if $a$ has an even number of non-$*$s, P2 if $a$ has an odd number of non-$*$s) has a winning strategy for the rest of the game, assuming $a$ records what has already been played ($a_i \in \{0,1\}$ means some player picked $x_i$ and assigned it a bit already). To solve Unordered QSat, call wins($** \cdots *$).

The current player can win iff they can make a move so the other player cannot then win:

```
wins(a):
    if a has no *s: evaluate and return φ(a_1 ··· a_n)
    for each i ∈ [n] such that a_i = * and each b ∈ {0,1}:
        let a^{i,b} ∈ {0,1,*}^n be the same as a except a_i^{i,b} = b
        if wins(a^{i,b}) = 0: return 1
    return 0
```

This recursive subroutine maintains the invariant that wins$(a) = 1$ iff in the game on $\varphi|_a$, the player who goes first (which may be P1 or P2 from the overall game on $\varphi$) has a winning strategy. This holds for the base cases when $a$ has no $*$s: wins$(a) = 1$ iff $\varphi(a_1 \cdots a_n) = 1$ iff P1 would win $\varphi|_a$, because it would be P1's turn when the game is over since $n$ is even. For non-base cases, the invariant is maintained because wins$(a) = 1$ iff there exists a move $x_i = b$ for the current player such that wins$(a^{i,b}) = 0$, which means whoever played $x_i = b$ has a winning strategy on $\varphi|_{a^{i,b}}$ since their opponent (who has the next turn) doesn't.

The number of non-$*$s in $a$, plus 1, is the depth of a call, so the overall recursion depth is $n + 1 \leq N$. Each stack frame needs $O(N)$ bits for the argument and other local data (if we pack $O(\log N)$ bits of $a$ into an individual word). The space efficiency is:

$$\big(N \text{ stack frames at a time}\big) \cdot \big(O(N) \text{ bits per stack frame}\big) \;=\; O(N^2) \text{ bits}$$

**Exercise 3.17.b:** We show 2-SAT $\leq_\ell^m$ 4-UNORDERED QSAT by mapping $\varphi(x_1 \cdots x_n)$ to $\psi(x_1 y_1 \cdots x_n y_n)$ by replacing each occurrence of $x_i$ with $x_i \vee y_i$ and replacing each occurrence of $\overline{x_i}$ with $\overline{x_i} \vee \overline{y_i}$. Note that $\psi$ is indeed a 4-CNF with an even number of variables. To show this log-space mapping reduction is correct, we argue that $\varphi$ is satisfiable iff P1 has a winning strategy on $\psi$:

$\Leftarrow$: Assume $\varphi$ is unsatisfiable. Here is a winning strategy for P2 on $\psi$: When P1 picks $x_i$ or $y_i$ and assigns it a bit, P2 responds by picking the other one of $x_i$ or $y_i$ and assigning it the same bit. This guarantees that $x_i = y_i$ for all $i \in [n]$, and thus $\psi$ is equivalent to $\varphi$ where P1 gets to assign all variables. Since $\varphi$ is unsatisfiable, P2's strategy guarantees that $\psi$ won't be satisfied.

$\Rightarrow$: Assume $\varphi$ is satisfiable, and consider a satisfying assignment $a \in \{0,1\}^n$. Here is a winning strategy for P1 on $\psi$:

```
repeat until all variables have been assigned:
    P1 picks an unassigned x_i and assigns x_i = a_i
    repeat:
        if P2 picked y_i: break out of inner loop
        else if P2 picked some x_j or y_j with j ≠ i:
            P1 picks the other one of x_j or y_j and assigns it the opposite bit, so x_j ≠ y_j
```

This maintains the outer loop invariant that for all $k \in [n]$, $x_k$ and $y_k$ are either both assigned or both unassigned, and the inner loop invariant that $x_i$ is assigned and $y_i$ is unassigned and for all $k \neq i$, $x_k$ and $y_k$ are either both assigned or both unassigned. Thus P2 will eventually have to pick $y_i$ in the inner loop.

This strategy guarantees that for each $k \in [n]$, either $x_k = a_k$ or $y_k = a_k$, and thus $\psi$ is satisfied since $a$ satisfies $\varphi$. This is because either $x_k = a_k$ if P1 picks $i = k$ in some outer loop iteration, or $x_k \neq y_k$ if P2 picks $j = k$ in some inner loop iteration.

**Exercise 3.17.c:** We assume without loss of generality that $\varphi$ has no duplicate clauses such as $(x_1 \vee \overline{x_2}) \wedge (x_1 \vee \overline{x_2})$. When we say variable $x_i$ appears in clause $C$, we mean $C$ contains the literal $x_i$ or contains the literal $\overline{x_i}$. When we say two clauses have a variable in common, we mean at least one variable $x_i$ appears in both clauses (either both contain $x_i$, or both contain $\overline{x_i}$, or one contains $x_i$ and the other contains $\overline{x_i}$).

We claim that P1 has a winning strategy iff there exists a variable $x_i$ that appears either only positively or only negatively in $\varphi$, and such that among the clauses in which $x_i$ doesn't appear, no two of them have a variable in common. This property is straightforward to check in log space:

```
for each i ∈ [n]:
    if φ has a clause containing x_i and a clause containing ¬x_i:
        continue to next iteration of i loop
    for each clause C of φ such that x_i doesn't appear in C:
        for each clause D ≠ C of φ such that x_i doesn't appear in D:
            if C and D have a variable in common:
                continue to next iteration of i loop
    accept
reject
```

Now, we prove the claim:

$\Leftarrow$: Suppose there exists such an $x_i$. Here is a winning strategy for P1:

```
if x_i only appears positively in φ: P1 assigns x_i = 1
else x_i only appears negatively in φ: P1 assigns x_i = 0
repeat until only one variable remains:
    if P2 picked a variable that appears in some not-yet-satisfied clause:
        P1 picks the other variable appearing in that clause and assigns it to satisfy the clause
    else:
        P1 picks any remaining variable
        if that variable appears in a not-yet-satisfied clause: P1 assigns it to satisfy the clause
        else: P1 assigns it an arbitrary bit
P2 picks the only remaining variable
```

P1's first move satisfies all clauses of $\varphi$ in which $x_i$ appears. (If $x_i$ is a dummy variable, then it doesn't matter which bit P1 assigns to $x_i$.) P1's subsequent moves satisfy the remaining clauses. None of these remaining clauses contain $x_i$ or $\overline{x_i}$, and no two of these clauses have a variable in common. Thus this strategy maintains the loop invariant that for every not-yet-satisfied clause, both its variables are unassigned. Since the invariant holds at the end of the loop, all clauses must be satisfied because there's only one unassigned variable left, but a not-yet-satisfied clause would have two unassigned variables.

$\Rightarrow$: Suppose there does not exist such an $x_i$. That is, for every $x_i$, either $\varphi$ has a clause containing $x_i$ and a clause containing $\overline{x_i}$, or $\varphi$ has two different clauses such that $x_i$ appears in neither, and they have a variable in common. Here is a winning strategy for P2:

P1 picks some variable $x_i$ and assigns it a bit
if $\varphi$ has a clause containing $x_i$ and a clause containing $\overline{x_i}$:
    P2 picks the other variable in whichever of those two clauses wasn't satisfied by P1's move
    P2 assigns that variable so the clause is unsatisfied
else:
    let $C \neq D$ be clauses such that $x_i$ appears in neither, and $C, D$ have a variable in common
    if $C, D$ have both variables in common:
        if $C, D$ have neither literal in common:
            suppose $C = (x_1 \vee x_2)$ and $D = (\overline{x_1} \vee \overline{x_2})$
            P2 waits by picking other variables until P1 picks $x_1$ or $x_2$
            P2 responds by picking the other one of $x_1$ or $x_2$ and assigning it so $x_1 = x_2$
        else $C, D$ have one literal in common:
            suppose $C = (x_1 \vee x_2)$ and $D = (x_1 \vee \overline{x_2})$
            P2 assigns $x_1 = 0$
    else $C, D$ have only one variable in common:
        if the common variable is the same literal in $C, D$:
            suppose $C = (x_1 \vee x_2)$ and $D = (x_2 \vee x_3)$
            P2 assigns $x_2 = 0$
            after P1's move, at least one of $x_1$ or $x_3$ is available, and P2 assigns it 0
        else the common variable is opposite literals in $C, D$:
            suppose $C = (x_1 \vee x_2)$ and $D = (\overline{x_2} \vee x_3)$
            P2 assigns $x_1 = 0$
            if P1 assigns $x_2 = 1$: P2 assigns $x_3 = 0$
            else if P1 doesn't assign $x_2 = 0$: P2 assigns $x_2 = 0$

When $C, D$ have neither literal in common, P2 can indeed wait until P1 picks $x_1$ or $x_2$ because P2 has the last move since $n$ is even. P2 makes $x_1 = x_2$ in this case, so either $C$ is unsatisfied since $x_1 = x_2 = 0$, or $D$ is unsatisfied since $x_1 = x_2 = 1$. In the case that $C, D$ have one literal in common, it doesn't matter who picks $x_2$ or what they assign it—either $C$ or $D$ will be unsatisfied.

**Exercise 3.18:** Since PSPACE = NPSPACE (Theorem 3.18), it suffices to prove Pebble Game $\in$ NPSPACE. Here is a poly-space streaming verifier $V$ for Pebble Game. It views the purported witness $w$ as a sequence of moves, and simulates the gameplay accordingly, with an alarm clock to preempt a wild goose chase.

```
on input (G, k) where G has n nodes, and purported witness w:
    initially each node has no pebble
    repeat 2^n times:
        v ← w's next word
        if v is not a node of G: reject
        if v has no pebble:
            if some inneighbor of v has no pebble: reject
            if there are already k pebbles on G: reject
            put a pebble on v
            if v is the sink: accept
        if v has a pebble:
            remove the pebble from v
    reject
```

$V$ uses poly space since it needs $n$ bits for the loop counter, $n$ bits to remember which nodes have pebbles, and some registers of word size $O(\log N)$ for $k$, $v$, the number of pebbles currently used, and some pointers. Note that $k \leq n$ without loss of generality, since $n$ pebbles are always enough.

$V$ always terminates by definition. It checks that $w$ represents a sequence of moves that wins the pebble game on $G$ using only $k$ pebbles. If $V(G, k; w)$ accepts then $(G, k)$ is a yes-input of Pebble Game. Conversely, if $(G, k)$ is a yes-input of Pebble Game then there exists a sequence of at most $2^n$ moves that wins the game—because there are at most $2^n$ possibilities for which subset of nodes have pebbles and there's no reason to repeat a possibility—and thus $V(G, k; w)$ accepts for some $w$. Hence $V$ is correct.

**Exercise 3.19:** The proof is the same as for Exercise 2.42 (if $A \leq_p^o B$ and $B \in \mathsf{NP} \cap \mathsf{coNP}$ then $A \in \mathsf{NP} \cap \mathsf{coNP}$) but adapted to recompute each word of each query when needed (as in the proof of Lemma 1.18) and using $\mathsf{NL} \cap \mathsf{coNL} = \mathsf{NL}$.

**Exercise 3.20:** Suppose $A \in \mathsf{NSPACE}[S]$ by a streaming verifier $V$ with $L$ lines, $R$ registers, space efficiency $\leq aS$ (for an integer $a$) and $O(\log \max(S,N))$-bounded word size $W$. Consider any valid input $x$ of size $N$.

For each $M \in \{0,1,2,\ldots,\}$, define the configuration graph $G_{V,x,M}$ as follows: Nodes correspond to configurations in $\{0,1\}^{\log L} \times (\{0,1\}^W)^R \times (\{0,1\}^W)^M$ representing the program counter, all $R$ registers, and the work segment's first $M$ words. There's also an extra node $t$ (not corresponding to a configuration) with no outgoing edges. To define the outgoing edges of the node $v_C$ for configuration $C$, consider $V$'s instruction at line $C_{\mathrm{PC}}$:

- Data processing or control flow: $v_C$ has an edge to $v_{C'}$ where $C'$ is the configuration after one step of $V$ from $C$.
- "load Reg$[i]$ ← Mem[Reg$[j]$]" or "store Mem[Reg$[j]$] ← Reg$[i]$": If $C_{\mathrm{Reg}[j]} \geq M$ then $C$ and $v_C$ are deemed *bad* and $v_C$ has no outgoing edges. Otherwise, $v_C$ has an edge to $v_{C'}$ where $C'$ is the configuration after one step of $V$ from $C$.
- "read Reg$[i]$ ← In[Reg$[j]$]": $v_C$ has an edge to $v_{C'}$ where $C'$ is the configuration after one step of $V$ on input $x$ from $C$, as in the proof of Theorem 3.13.
- "read-witness Reg$[i]$": $v_C$ has $2^W$ edges (one for each possible next word of $V$'s witness segment) as in the proof of Theorem 3.13.
- "accept": $v_C$ has an edge to $t$.
- "reject": $v_C$ has no outgoing edges.

We always let $s$ be the node associated with the initial configuration. The memory space of $V$ on input $x$ is $(R+M)W$ for the smallest $M$ such that no bad node is reachable from $s$ in $G_{V,x,M}$.

The following algorithm shows $A \in \mathsf{TIME}[2^{O(S)}]$:

```
construct W(N)
for M ← 0, 1, 2, ...:
    build G_{V,x,M}
    run breadth-first search from s in G_{V,x,M}
    if t is reachable from s: accept
    else if no bad node is reachable from s: reject
```

This algorithm solves $A$ because: If $t$ is reachable from $s$ in $G_{V,x,M}$ then there exists $w$ such that $V(x;w)$ accepts, so the algorithm correctly accepts. If no bad node is reachable from $s$, then this $M$ must be the one that determines the memory space of $V$ on input $x$, so if $t$ is also not reachable from $s$ then $V(x;w)$ rejects for all $w$, and the algorithm correctly rejects.

Constructing $W(N)$ takes time $O(\log \max(S,N)) \leq O(S)$ since $S \geq \log N$. Since the algorithm terminates before $(R+M)W > aS$, we always have $M \leq aS/W - R \leq O(S)$, so the $M$ loop only incurs a $O(S)$ factor time overhead. The size of $G_{V,x,M}$ is at most:

$$\begin{aligned} & \left(2^{\log L + (R+M)W} + 1 \text{ nodes}\right) \cdot \left(2^W \text{ edges per node}\right) \\ \leq\ & \left(2^{\log L + aS} + 1 \text{ nodes}\right) \cdot \left(2^{O(S)} \text{ edges per node}\right) \\ \leq\ & 2^{O(S)} \end{aligned}$$

Since BFS takes linear time, this algorithm's overall time efficiency is $2^{O(S)}$. Also, some $O(S)$-bounded word size suffices.

The following algorithm shows $A \in \mathsf{SPACE}[S^2]$:

```
construct W(N)
for M ← 0, 1, 2, ...:
    run middle-first search to see whether t is reachable from s in G_{V,x,M}
    if so: accept
    for each bad configuration C:
        run middle-first search to see whether v_C is reachable from s in G_{V,x,M}
        if so: continue to the next iteration of the M loop
    reject
```

This algorithm solves $A$ for the same reason as the previous algorithm. Constructing $W(N)$ takes space $O(\log \max(S, N)) \leq O(S)$ since $S \geq \log N$. Since $G_{V,x,M}$ always has size $\leq 2^{O(S)}$, middle-first search always takes space $O(\log^2(2^{O(S)})) = O(S^2)$. Only $O(S)$ additional space is needed for the other data such as $M$ and $C$. Also, some $O(\log \max(S^2, N))$-bounded word size suffices.

The following streaming verifier shows $\overline{A} \in \mathsf{NSPACE}[S]$. The witness consists of:

- $M^*$, claimed to be the smallest value such that no bad node is reachable from $s$ in $G_{V,x,M^*}$
- for each $M \in \{0, 1, \ldots, M^* - 1\}$: a bad node in $G_{V,x,M}$ and a witness that it's reachable from $s$
- for each bad node in $G_{V,x,M^*}$: a witness that it's not reachable from $s$
- a witness that $t$ is not reachable from $s$ in $G_{V,x,M^*}$

Such a witness can be verified in $O(\log(2^{O(S)})) = O(S)$ space with $O(\log \max(S, N))$-bounded word size, as in the proof that $\mathsf{NL} = \mathsf{coNL}$.

**Exercise 3.21:** Given a valid input $(G, s, t)$ where $G$ has $n$ nodes, let $S_d$ be the set of all nodes with distance $\leq d$ from $s$. Thus $S_{n-1}$ is the set of all nodes reachable from $s$. Let $D(v)$ be the distance from $s$ to node $v$, and let $D(S_d) = \sum_{v \in S_d} D(v)$.

First, consider the following pseudocode fragment, which we dub "review". We claim if $prevcount = |S_{d-1}|$ and $prevsum = D(S_{d-1})$, then "review" rejects for all but exactly one possibility of the words it reads from the purported witness $w$, and when it doesn't reject, the nodes $u \in S_{d-1}$ are exactly those that cause an increment to *tempcount*.

```
"review":
    tempcount ← 0 and tempsum ← 0
    for each node u:
        if w's next word ∉ {0, 1}: reject
        if that word was 1:
            if w's next words aren't a walk from s to u of length ≤ d − 1: reject
            increment tempcount
            add that walk's length to tempsum
    if tempcount < prevcount or tempsum > prevsum: reject
```

We prove the above claim. First, note that "review" doesn't reject the following $w$: For each $u$: If $u \in S_{d-1}$, let $w$ have a 1 followed by the unique shortest path from $s$ to $u$. If $u \notin S_{d-1}$, let $w$ have a 0. We just need to show that "review" rejects all other $w$. If $u \notin S_{d-1}$ but $w$'s next word isn't 0 in the $u$ iteration, then "review" would reject the "walk check." If $u \in S_{d-1}$ but $w$'s next word isn't 1 in the $u$ iteration, then "review" would reject because $tempcount < prevcount = |S_{d-1}|$ (since *tempcount* only counts nodes in $S_{d-1}$ but misses this $u$). If every $u \in S_{d-1}$ causes an increment to *tempcount* but for some $u \in S_{d-1}$, $w$ has a walk from $s$ to $u$ that's not the unique shortest path from $s$ to $u$, then "review" would reject because $tempsum > prevsum = D(S_{d-1})$ (since every $u \in S_{d-1}$ contributes $\geq D(u)$ to *tempsum* but some $u \in S_{d-1}$ contributes $> D(u)$ to *tempsum*).

Here is pseudocode for an unambiguous log-space streaming verifier $V$ for Unambiguous Directed Reachability. In iteration $d$, *curcount* counts how many nodes $v$ are in $S_d$ and *cursum* sums $D(v)$ for these nodes, while *prevcount* and *prevsum* remember $|S_{d-1}|$ and $D(S_{d-1})$ (from iteration $d-1$). Before the loop, $curcount \leftarrow 1$ and $cursum \leftarrow 0$ because $S_0 = \{s\}$ has one node and $D(s) = 0$. Iteration $d$ starts with $curcount = |S_{d-1}|$ and $cursum = D(S_{d-1})$ (from iteration $d-1$), and then adds to *curcount* the number of nodes $v$ with $D(v) = d$, and adds to *cursum* that number of nodes times $d$. In each iteration of the $v$ loop, $V$ checks whether $D(v) = d$ and if so, increments *curcount* and adds $d$ to *cursum*.

```
on input (G, s, t) where G has n nodes, and purported witness w:
    if s = t: accept
    curcount ← 1 and cursum ← 0
    for d ← 1, 2, ..., n − 1:
        prevcount ← curcount and prevsum ← cursum
        for each node v:
            run "review" to enumerate S_{d−1}
            if v ∉ S_{d−1} but some u ∈ S_{d−1} has an edge to v:
                if v = t: accept
                increment curcount
                add d to cursum
    reject
```

First, we show that for every valid input, there's exactly one $w$ such that no run of "review" rejects, and that for this $w$, $V$ maintains the invariant that $curcount = |S_d|$ and $cursum = D(S_d)$ at the end of iteration $d$: Assuming $prevcount = |S_{d-1}|$ and $prevsum = D(S_{d-1})$ in iteration $d$, for each $v$ let $w$ have the unique words so that "review" doesn't reject (and thus "review" correctly enumerates $S_{d-1}$). Then the invariant is maintained because $D(v) = d$ iff $v \notin S_{d-1}$ but some $u \in S_{d-1}$ has an edge to $v$.

Thus for the unique $w$ such that no run of "review" rejects, and for each $d$ and $v$, the "if" condition is true iff $D(v) = d$. Now, we show $V$ is correct. Assume $s \neq t$. If $t$ is reachable from $s$, then $V$ will accept when $d = D(t)$ and $v = t$. Conversely, if $V$ accepts then $D(t) \in \{1, \ldots, n-1\}$ and thus $t$ is reachable from $s$.

$V$ is register-only, and word size $O(\log N)$ suffices, so $V$ has $O(\log N)$ space efficiency.

**Exercise 4.1.a:** Suppose $A \in \mathsf{SPACE}[N^2]$ by a program $\Pi$ with word size $e \log N^2$ for an integer $e \geq 1$. Let $\text{pad}(N) = N^2$. Then $A_{\text{pad}} \in \mathsf{SPACE}[M]$ by a program $\Pi_{\text{pad}}$ where $\Pi_{\text{pad}}(x_{\text{pad}})$ runs $\Pi(x)$ by telling $\Pi$ the input size is $N = \text{pad}^{-1}(M) = M^{1/2}$, with space efficiency $O(M) = O(N^2)$ and word size $e \log M = e \log N^2$.

Assuming $\mathsf{SPACE}[M] \subseteq \mathsf{TIME}[M^3]$, we have $A_{\text{pad}} \in \mathsf{TIME}[M^3]$ by a program $\Pi'_{\text{pad}}$ with word size $e' \log M$ for an integer $e' \geq 1$. Then $A \in \mathsf{TIME}[N^6]$ by a program $\Pi'$ where $\Pi'(x)$ runs $\Pi'_{\text{pad}}(x_{\text{pad}})$ by telling $\Pi'_{\text{pad}}$ the input size is $M = \text{pad}(N) = N^2$, with time efficiency $O(N^6) = O(M^3)$ and word size $e' \log N^2 = e' \log M$.

**Exercise 4.1.b:** Suppose $A \in \mathsf{TIME}[N^6]$ by a program $\Pi$ with word size $e \log N^3$ for an integer $e \geq 1$. Let $\mathrm{pad}(N) = N^3$. Then $A_{\mathrm{pad}} \in \mathsf{TIME}[M^2]$ by a program $\Pi_{\mathrm{pad}}$ where $\Pi_{\mathrm{pad}}(x_{\mathrm{pad}})$ runs $\Pi(x)$ by telling $\Pi$ the input size is $N = \mathrm{pad}^{-1}(M) = M^{1/3}$, with time efficiency $O(M^2) = O(N^6)$ and word size $e \log M = e \log N^3$.

Assuming $\mathsf{TIME}[M^2] \subseteq \mathsf{NTIME}[M]$, we have $A_{\mathrm{pad}} \in \mathsf{NTIME}[M]$ by a verifier $V_{\mathrm{pad}}$ with word size $e' \log M$ for an integer $e' \geq 1$. Then $A \in \mathsf{NTIME}[N^3]$ by a verifier $V$ where $V(x; w)$ runs $V_{\mathrm{pad}}(x_{\mathrm{pad}}; w)$ by telling $V_{\mathrm{pad}}$ the input size is $M = \mathrm{pad}(N) = N^3$, with time efficiency $O(N^3) = O(M)$ and word size $e' \log N^3$. This $V$ is a correct verifier for $A$ because:

$$A(x) = 1 \iff A_{\mathrm{pad}}(x_{\mathrm{pad}}) = 1 \iff \big(\exists w : V_{\mathrm{pad}}(x_{\mathrm{pad}}; w) \text{ accepts}\big) \iff \big(\exists w : V(x; w) \text{ accepts}\big)$$

**Exercise 4.2.a:** We prove the contrapositive, using essentially the same proof as Theorem 4.4.

Assuming NP $\subseteq$ L, we prove NEXP $\subseteq$ PSPACE. Suppose $A \in$ NEXP by a verifier $V$ with time efficiency $O(2^{N^d})$ and word size $eN^d$ for integers $d, e \geq 1$. Let $\text{pad}(N) = 2^{N^d}$. Then $A_{\text{pad}} \in$ NP by a verifier $V_{\text{pad}}$ where $V_{\text{pad}}(x_{\text{pad}}; w)$ runs $V(x; w)$ by telling $V$ the input size is $N = \text{pad}^{-1}(M)$, with time efficiency $O(M) = O(2^{N^d})$ and word size $e \log M = eN^d$.

Since $A_{\text{pad}} \in$ NP, we have $A_{\text{pad}} \in$ L by a program $\Pi_{\text{pad}}$ with space efficiency $O(\log M)$ and word size $e' \log M$ for an integer $e' \geq 1$. Then $A$ can be solved by a program $\Pi$ where $\Pi(x)$ runs $\Pi_{\text{pad}}(x_{\text{pad}})$ by telling $\Pi_{\text{pad}}$ the input size is $M = \text{pad}(N)$, with space efficiency $O(N^d) = O(\log M)$ and word size $W = e'N^d = e' \log M$. To see that $A \in$ PSPACE, apply Lemma 4.5 to turn $\Pi$ into a program with space efficiency $O(N^d)$ and some log-bounded word size $W'$. This is possible since $\log(O(N^d))$, $\log(N + 1)$, and $A$'s word size are all $O(\log N)$, and $W$ is $(O(1), W')$-constructible since $e'$ and $d$ are integers.

**Exercise 4.2.b:** We prove the contrapositive. Assuming $\mathsf{SPACE}[N^d] \subseteq \mathsf{NP}$ for some $d > 0$, we prove $\mathsf{PSPACE} \subseteq \mathsf{NP}$. By decreasing $d$ if necessary, we may assume $d$ is the reciprocal of an integer. Suppose $A \in \mathsf{PSPACE}$ by a program $\Pi$ with time efficiency $O(N^c)$ and word size $e \log N^c$ for integers $c \geq 1$ and $e \geq 1/d$. Let $\mathrm{pad}(N) = N^{c/d}$. Then $A_{\mathrm{pad}} \in \mathsf{SPACE}[M^d]$ by a program $\Pi_{\mathrm{pad}}$ where $\Pi_{\mathrm{pad}}(x_{\mathrm{pad}})$ runs $\Pi(x)$ by telling $\Pi$ the input size is $N = \mathrm{pad}^{-1}(M) = M^{d/c}$, with space efficiency $O(M^d) = O(N^c)$ and word size $e \log M^d = e \log N^c$.

Since $A_{\mathrm{pad}} \in \mathsf{SPACE}[M^d]$, we have $A_{\mathrm{pad}} \in \mathsf{NP}$ by a verifier $V_{\mathrm{pad}}$ with time efficiency $O(M^b)$ and word size $e' \log M$ for integers $b, e' \geq 1$. Then $A \in \mathsf{NP}$ by a verifier $V$ where $V(x; w)$ runs $V_{\mathrm{pad}}(x_{\mathrm{pad}}; w)$ by telling $V_{\mathrm{pad}}$ the input size is $M = \mathrm{pad}(N) = N^{c/d}$, with time efficiency $O(N^{bc/d}) = O(M^b)$ and word size $e' \log N^{c/d} = e' \log M$.

**Exercise 4.3:** We prove the contrapositive, following the same outline as the proof of Theorem 4.2.

Assuming $\mathsf{NTIME}[2^{\log^d N}] \subseteq \mathsf{TIME}[N^{\mathrm{polylog}\, N}]$ for some $d > 0$, we prove $\mathsf{NEXP} \subseteq \mathsf{EXP}$. By decreasing $d$ if necessary, we may assume $d$ is the reciprocal of an integer. Suppose $A \in \mathsf{NEXP}$ by a verifier $V$ with time efficiency $O(2^{N^c})$ and word size $eN^{c/d}$ for integers $c, e \geq 1$. Let $\mathrm{pad}(N) = 2^{N^{c/d}}$. Then $A_{\mathrm{pad}} \in \mathsf{NTIME}[2^{\log^d M}]$ by a verifier $V_{\mathrm{pad}}$ where $V_{\mathrm{pad}}(x_{\mathrm{pad}}; w)$ runs $V(x; w)$ by telling $V$ the input size is $N = \mathrm{pad}^{-1}(M) = \log^{d/c} M$, with time efficiency $O(2^{\log^d M}) = O(2^{N^c})$ and word size $e \log M = eN^{c/d}$. This $V_{\mathrm{pad}}$ is a correct verifier for $A_{\mathrm{pad}}$ because:

$$A_{\mathrm{pad}}(x_{\mathrm{pad}}) = 1 \iff A(x) = 1 \iff \big(\exists w : V(x; w) \text{ accepts}\big) \iff \big(\exists w : V_{\mathrm{pad}}(x_{\mathrm{pad}}; w) \text{ accepts}\big)$$

Since $A_{\mathrm{pad}} \in \mathsf{NTIME}[2^{\log^d M}]$, we have $A_{\mathrm{pad}} \in \mathsf{TIME}[M^{\mathrm{polylog}\, M}] = \mathsf{TIME}[2^{\mathrm{polylog}\, M}]$ by a program $\Pi_{\mathrm{pad}}$ with time efficiency $O(2^{\log^b M})$ and word size $e' \log^b M$ for integers $b, e' \geq 1$. Then $A \in \mathsf{EXP}$ by a program $\Pi$ where $\Pi(x)$ runs $\Pi_{\mathrm{pad}}(x_{\mathrm{pad}})$ by telling $\Pi_{\mathrm{pad}}$ the input size is $M = \mathrm{pad}(N) = 2^{N^{c/d}}$, with time efficiency $O(2^{N^{bc/d}}) = O(2^{\log^b M})$ and word size $e' N^{bc/d} = e' \log^b M$.

**Exercise 4.4:** Assuming $\mathsf{L} \subseteq \mathsf{TIME}[\text{polylog}\, N]$, we prove $\mathsf{PSPACE} \subseteq \mathsf{P}$. Suppose $A \in \mathsf{PSPACE}$ by a register-only program $\Pi$ with space efficiency $O(N^d)$ and word size $eN^d$ for integers $d, e \geq 1$. This can be achieved by packing all the "small words" of the work segment into one "big register" and modifying loads and stores to locate the desired "small words" using bit manipulation (like in the proof of Lemma 4.11). Let $\text{pad}(N) = 2^{N^d}$. Then $A_{\text{pad}} \in \mathsf{L}$ by a register-only program $\Pi_{\text{pad}}$ where $\Pi_{\text{pad}}(x_{\text{pad}})$ runs $\Pi(x)$ by telling $\Pi$ the input size is $N = \text{pad}^{-1}(M) = \log^{1/d} M$, with space efficiency $O(\log M) = O(N^d)$ and word size $e \log M = eN^d$.

Since $A_{\text{pad}} \in \mathsf{L}$, we have $A_{\text{pad}} \in \mathsf{TIME}[\text{polylog}\, M]$. For some integer $b \geq 1$, we have

$$A_{\text{pad}} \in \mathsf{TIME}[\log^b M] \subseteq \mathsf{TISP}[(\log^b M)(\log^2(\log^b M)), \log^{b+1} M] \subseteq \mathsf{TISP}[\log^{b+1} M, \log^{b+1} M]$$

(see the end of §1.7.3, about the sublinear-time version of Theorem 1.10) by a program $\Pi'_{\text{pad}}$ with word size $e' \log M$ for an integer $e' \geq 1$. Then $A$ can be solved by a program $\Pi'$ where $\Pi'(x)$ runs $\Pi'_{\text{pad}}(x_{\text{pad}})$ by telling $\Pi'_{\text{pad}}$ the input size is $M = \text{pad}(N) = 2^{N^d}$, with time and space efficiencies $O(N^{(b+1)d}) = O(\log^{b+1} M)$ and word size $W = e'N^d = e' \log M$. To see that $A \in \mathsf{P}$, apply Lemma 4.5 to turn $\Pi'$ into a program with time efficiency $O(N^{(b+1)d} \cdot (N^d)^2) = O(N^{(b+3)d})$ and some log-bounded word size $W'$. This is possible since $\log(O(N^{(b+1)d}))$, $\log(N+1)$, and $A$'s word size are all $O(\log N)$, and $W$ is $(O(1), W')$-constructible since $e'$ and $d$ are integers.

**Exercise 4.5.a:** $\Leftarrow$: If $\mathsf{L} = \mathsf{P}$ then $\mathsf{TIME}[N] \subseteq \mathsf{P} \subseteq \mathsf{L}$.

$\Rightarrow$: Assuming $\mathsf{TIME}[N] \subseteq \mathsf{L}$, we prove $\mathsf{P} \subseteq \mathsf{L}$. Suppose $A \in \mathsf{P}$, say $A \in \mathsf{TIME}[N^d]$ for some integer $d > 0$. By the usual padding argument with $\text{pad}(N) = N^d$:

$$A \in \mathsf{TIME}[N^d] \Rightarrow A_{\text{pad}} \in \mathsf{TIME}[M] \Rightarrow A_{\text{pad}} \in \mathsf{L} = \mathsf{SPACE}[\log M] \Rightarrow A \in \mathsf{SPACE}[\log N^d] = \mathsf{L}$$

**Exercise 4.5.b:** Suppose for contradiction $\mathsf{TIME}[N] = \mathsf{L}$. From $\mathsf{TIME}[N] \subseteq \mathsf{L}$ we get $\mathsf{P} \subseteq \mathsf{L} \subseteq \mathsf{TIME}[N]$ (Exercise 4.5.a), contradicting the time hierarchy (Theorem 4.7).

**Exercise 4.5.c:** $\Leftarrow$: If $\mathsf{P} = \mathsf{NP}$ then $\mathsf{NTIME}[N] \subseteq \mathsf{NP} \subseteq \mathsf{P}$.

$\Rightarrow$: Assuming $\mathsf{NTIME}[N] \subseteq \mathsf{P}$, we prove $\mathsf{NP} \subseteq \mathsf{P}$. Suppose $A \in \mathsf{NP}$, say $A \in \mathsf{NTIME}[N^d]$ for some integer $d > 0$. By the usual padding argument with $\text{pad}(N) = N^d$:

$$A \in \mathsf{NTIME}[N^d] \Rightarrow A_{\text{pad}} \in \mathsf{NTIME}[M] \Rightarrow A_{\text{pad}} \in \mathsf{P} = \mathsf{TIME}[\text{poly}\, M] \Rightarrow A \in \mathsf{TIME}[\text{poly}\, N^d] = \mathsf{P}$$

**Exercise 4.5.d:** Suppose for contradiction $\mathsf{NTIME}[N] = \mathsf{P}$. From $\mathsf{NTIME}[N] \subseteq \mathsf{P}$ we get $\mathsf{NP} \subseteq \mathsf{P} \subseteq \mathsf{NTIME}[N]$ (Exercise 4.5.c), contradicting the time hierarchy for verifiers (Theorem 4.13).

**Exercise 4.6:** We prove that there exist decision problems $A$ and $B$ such that $A \leq_p^m B$ and $B \in \mathsf{E}$ but $A \notin \mathsf{E}$. There exists $A \in \mathsf{TIME}[2^{N^2}] \setminus \mathsf{TIME}[2^{O(N)}]$ by the time hierarchy theorem (suitably adapted to these time bounds). Let $\mathrm{pad}(N) = N^2$ and $B = A_{\mathrm{pad}}$. Then $A \leq_p^m B$ (by a trivial quadratic-time reduction that just adds the appropriate amount of padding to the input) and $B \in \mathsf{E}$ (by a program that runs a $O(2^{N^2}) = O(2^M)$-time program for $A$ and tells it the input size is $N = \mathrm{pad}^{-1}(M)$) but $A \notin \mathsf{E}$.

**Exercise 4.7:** Here's the idea: The time hierarchy proof shows that a too-fast program fails to solve $B$ on input $(I, W)$ if the code $I$ is large enough. Beyond this "large enough" threshold, we can keep appending any number of dummy instructions to $I$. This shows that for all but finitely many $M$ (not merely infinitely many $M$), there exists an input of size $M$ on which the program fails to solve $B$.

Formally: Suppose for contradiction that $\Pi$ solves $B$, has log-bounded word size, and does not have $\omega(M^b)$ time efficiency. The latter means there exists a constant $d > 0$ such that for infinitely many $M$, $\Pi(x)$ takes at most $dM^b$ steps on every valid input $x$ of size $M$. ($O(M^b)$ would mean the same but for all large enough $M$, instead of just infinitely many.) Let $X$ be the set of valid inputs of $B$ of these infinitely many input sizes. Let $B'$ be the same as $B$ but where only inputs in $X$ are considered valid. Thus $\Pi$ shows that $B' \in \mathsf{TIME}[M^b]$. Let $\Pi'$ be the program with tight code $I$ after applying Claim 4.8 to $B'$. Let $W$ be $\Pi'$'s word size when $M$ is the size of input $(I, W)$. The rest of the time hierarchy proof shows that $\Pi'(I, W) \neq B'(I, W)$, so $\Pi'$ fails to solve $B'$ on input $(I, W)$, if $(I, W) \in X$ and $M$ is large enough. Since $X$ has infinitely many valid input sizes, we can ensure these conditions by appending a suitable number of dummy instructions to $I$.

This doesn't work for the time hierarchy for verifiers, because that proof doesn't let us control the size of an input where $V$ fails. It only shows that $V$ fails on at least one of a set of inputs of different sizes, so we can't guarantee $V$ will fail on an input whose size is in a particular infinite set.

**Exercise 4.8.a:** Let $c$ be a rational number with $a \geq c > b$.

| |
|---|
| (call this problem $B$, with input size $M$) |
| Input: Tight code $I$, $W$ such that $\log M + 10 \leq W \leq c \log M$ |
| Output: Does there exist $w \in (\{0,1\}^W)^{M^c}$ such that $I_W(I, W; w)$ accepts within $M^c$ steps? |

$B \in \mathsf{NTIME}[M^a]$: On input $(I, W)$, run a linear-time verifier for Verifier Interpretation on input $\big(I, W, (I, W), \text{padding of size } M^c\big)$. This takes $O(M^c) \leq O(M^a)$ time since the latter input has size $O(M^c)$.

Suppose for contradiction $\overline{B} \in \mathsf{NTIME}[M^b]$. Like in Claim 4.8, some verifier $V$ with tight code $I$ verifies $\overline{B}$ in time $O(M^b \log^4 M)$ with word size $b \log M + e$ for some constant $e$. Let $W = b \log M + e$ be $V$'s word size when $M$ is the size of input $(I, W)$. Since $b < c$, we can ensure $L$ is large enough (by appending dummy instructions to $I$) that $V$ runs in time $M^c$ on input $(I, W)$, and $W \leq c \log M$ so $(I, W)$ is a valid input. Quantifying over $w \in (\{0,1\}^W)^{M^c}$, we have a contradiction

$$\big(\exists w : \ V(I, W; w) \text{ accepts}\big) \iff \overline{B}(I, W) = 1 \iff \big(\forall w : \ V(I, W; w) \text{ rejects}\big)$$

where the first $\iff$ is because $V$ is correct for $\overline{B}$ and $(I, W)$ is valid, and the second $\iff$ is by $\overline{B}$'s definition and the fact that $V$ runs in time $M^c$ on input $(I, W)$. More succinctly, $V(I, W) = \overline{B}(I, W) = \overline{V(I, W)}$.

**Exercise 4.8.b:** That proof concluded with:

$$\mathsf{NTIME}[N^a] \subseteq \mathsf{TISP}[N^{ab}, N^{ac}] \subseteq \Sigma_2\mathsf{TIME}[N] \subseteq \mathsf{NTIME}[N^b]$$

We adjust this to

$$\mathsf{NTIME}[N^a] \subseteq \mathsf{TISP}[N^{ab}, N^{ac}] = \mathsf{coTISP}[N^{ab}, N^{ac}] \subseteq \mathsf{co}\Sigma_2\mathsf{TIME}[N] \subseteq \mathsf{coNTIME}[N^b]$$

which contradicts Exercise 4.8.a.

**Exercise 4.9:** The "for every node $v$, deleting $v$ would make it properly 3-colorable" part has the form $\forall\,\exists$, so we might worry about how to express it with $\exists\,\forall$ to show the problem is in $\Sigma_2\mathsf{P}$. But we don't need a quantifier for $v$, because we can try all possible $v$ in poly time. Thus this part only needs an $\exists$ quantifier, where the witness contains one 3-color assignment per $v$. The "$G$ is not properly 3-colorable" part is a $\forall$ quantifier. These quantifiers don't depend on each other, so we can put them in either order.

Formally, consider the 2-witness verifier $V$ where:

- $V$ views $w_1$ as a 3-color assignment $c$ to $G$'s $n$ nodes.
- $V$ views $w_2$ as a tuple $(c_1, \ldots, c_n)$ where $c_v$ is a 3-color assignment to $G$'s nodes except $v$.
- $V(G; c; c_1, \ldots, c_n)$ checks that $c$ is not proper (some edge of $G$ is not multicolored) and for each $v$, all edges of $G$ except those incident to $v$ are multicolored in $c_v$. This takes poly time.

Abbreviate Critical Graph 3-Coloring as $A$. We have $A \in \Pi_2\mathsf{P}$ since:

$$A(G) = 1 \iff \big(\forall c\ \exists c_1, \ldots, c_n :\ \ V(G; c; c_1, \ldots, c_n) \text{ accepts}\big)$$

We also have $A \in \Sigma_2\mathsf{P}$ by letting $V'$ be $V$ but with $w_1$ and $w_2$ swapped:

$$A(G) = 1 \iff \big(\exists c_1, \ldots, c_n\ \forall c :\ V'(G; c_1, \ldots, c_n; c) \text{ accepts}\big)$$

**Exercise 4.10:** Abbreviate Robust Independent Set as $A$. Assume Independent Set $\in \mathsf{P}$. Since Independent Set is NP-complete, we have $\mathsf{P} = \mathsf{NP}$. Thus $\mathsf{P} = \mathsf{PH}$ (Theorem 4.21). Hence, it suffices to show $A \in \mathsf{PH}$. We have $A \in \Pi_2\mathsf{P}$ by a 2-witness verifier $V$ that views $w_1$ and $w_2$ as subsets of $G$'s $n$ nodes, and $V(G, k; w_1; w_2)$ checks that $|w_1| \geq n/2$ and $w_2 \subseteq w_1$ and $|w_2| = k$ and no edge has both endpoints in $w_2$:

$$A(G,k) = 1 \iff \big(\forall w_1 \ \exists w_2 : \ V(G,k;w_1;w_2) \text{ accepts}\big)$$

**Exercise 4.11:** Assume NP = coNP. We argue that $\Sigma_{\ell+1}\mathsf{P} \subseteq \Sigma_\ell \mathsf{P}$ for every $\ell \geq 1$. This yields the theorem since then $\Sigma_\ell \mathsf{P} \subseteq \Sigma_{\ell-1}\mathsf{P} \subseteq \Sigma_{\ell-2}\mathsf{P} \subseteq \cdots \subseteq \Sigma_1 \mathsf{P} = \mathsf{NP}$ and thus $\mathsf{PH} = \bigcup_{\ell \geq 1} \Sigma_\ell \mathsf{P} = \mathsf{NP}$.

Suppose $A \in \Sigma_{\ell+1}\mathsf{P}$ by an $(\ell+1)$-witness verifier $V$ with time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$. First, assume $\ell$ is even. As in the proof of Theorem 4.21, we define $B = \textsc{Final} \; \exists \; \textsc{of} \; V$ and see that $B \in \mathsf{NP}$ by a $O(M)$-time verifier $U$ where $U(x, w_1, \ldots, w_\ell; w_{\ell+1})$ runs $V(x; w_1; \cdots; w_\ell; w_{\ell+1})$. From our assumption that $\mathsf{NP} \subseteq \mathsf{coNP}$, we have $B \in \Pi_1 \mathsf{P}$ by a verifier $U'$:

$$B(x, w_1, \ldots, w_\ell) = 1 \iff \big(\forall w'_\ell : \; U'(x, w_1, \ldots, w_\ell; w'_\ell) \text{ accepts}\big)$$

Thus $A \in \Sigma_\ell \mathsf{P}$ by an $\ell$-witness verifier $V'$ where $V'(x; w_1; \cdots; w_\ell, w'_\ell)$ runs $U'(x, w_1, \ldots, w_\ell; w'_\ell)$ but masks each word of each $w_i$ to $W$ bits—which doesn't affect the truth of any $\exists$ or $\forall$ quantifier. (We don't bother making separate notation for the masked and unmasked versions.) The time efficiency of $V'$ is $\text{poly}M = \text{poly}N^d = \text{poly}N$ (since $U'$ takes $\text{poly}M$ time), and $V'$ is correct because for every valid input $x$:

$$\begin{aligned} A(x) = 1 &\iff \big(\exists w_1 \cdots \forall w_\ell \; \exists w_{\ell+1} : \; V(x; w_1; \cdots; w_\ell; w_{\ell+1}) \text{ accepts}\big) \\ &\iff \big(\exists w_1 \cdots \forall w_\ell \; \forall w'_\ell : \; U'(x, w_1, \ldots, w_\ell; w'_\ell) \text{ accepts}\big) \\ &\iff \big(\exists w_1 \cdots \; \forall w_\ell, w'_\ell \; : \; V'(x; w_1; \cdots; w_\ell, w'_\ell) \text{ accepts}\big) \end{aligned}$$

Since $\ell \geq 1$, there is indeed a $\forall w_\ell$ quantifier to merge $\forall w'_\ell$ with, so the latter is not a "new" quantifier.

Now, assume $\ell$ is odd. Let $B = \textsc{Final} \; \forall \; \textsc{of} \; V$, which asks whether for all $w_{\ell+1}$, $V(x; w_1; \cdots; w_\ell; w_{\ell+1})$ accepts. Then $B \in \Pi_1 \mathsf{P} = \mathsf{coNP}$. From our assumption that $\mathsf{coNP} \subseteq \mathsf{NP}$, we have $B \in \mathsf{NP}$ by a verifier $U'$. Thus $A \in \Sigma_\ell \mathsf{P}$ by an $\ell$-witness verifier $V'$ where $V'(x; w_1; \cdots; w_\ell, w'_\ell)$ runs $U'(x, w_1, \ldots, w_\ell; w'_\ell)$.

**Exercise 4.12:** $\mathsf{NTIME}[N] \subseteq \mathsf{TIME}[N^b]$ implies that for all $\ell$, $\Pi_{\ell+1}\mathsf{TIME}[N] \subseteq \Pi_\ell \mathsf{TIME}[N^b]$ by the same proof as Theorem 4.22, but going from $\ell+1$ quantifiers, starting with $\forall$, to $\ell$ quantifiers (instead of from 2 quantifiers, starting with $\exists$, to 1 quantifier). Also, $\Pi_2\mathsf{TIME}[N] \subseteq \Pi_1\mathsf{TIME}[N^b]$ implies $\Pi_2\mathsf{TIME}[N^b] \subseteq \Pi_1\mathsf{TIME}[(N^b)^b]$ by a padding argument, using the fact that $b$ is rational to efficiently compute $\mathrm{pad}(N) = N^b$ and $\mathrm{pad}^{-1}(M) = M^{1/b}$. Putting things together, $\Pi_3\mathsf{TIME}[N] \subseteq \Pi_2\mathsf{TIME}[N^b] \subseteq \Pi_1\mathsf{TIME}[N^{b^2}] = \mathsf{coNTIME}[N^{b^2}]$.

Here's a slightly different version of the proof: Assume $\mathsf{NTIME}[N] \subseteq \mathsf{TIME}[N^b]$, which by a padding argument implies $\mathsf{NTIME}[N^b] \subseteq \mathsf{TIME}[N^{b^2}]$ since $b$ is rational. Suppose $A \in \Pi_3\mathsf{TIME}[N]$ by a 3-witness verifier $V$. Letting $B = \textsc{Final}\ \forall\ \textsc{of}\ V$ with input size $M = \Theta(N)$, we have $B \in \Pi_1\mathsf{TIME}[M]$ by a verifier $U$ where $U(x, w_1, w_2; w_3)$ runs $V(x; w_1; w_2; w_3)$. Since our assumption implies $\Pi_1\mathsf{TIME}[M] \subseteq \mathsf{TIME}[M^b]$, we have $B \in \mathsf{TIME}[M^b]$ by a program $\Pi$. Letting $C = \textsc{Final}\ \exists\ \forall\ \textsc{of}\ V$ with input size $L = \Theta(M)$, we have $C \in \Sigma_1\mathsf{TIME}[L^b]$ by a verifier $U'$ where $U'(x, w_1; w_2)$ runs $\Pi(x, w_1, w_2)$. By our assumption, $C \in \mathsf{TIME}[L^{b^2}]$ by a program $\Pi'$. Thus $A \in \Pi_1\mathsf{TIME}[N^{b^2}]$ by a verifier $V'$ where $V'(x; w_1)$ runs $\Pi'(x, w_1)$. We conclude that $\Pi_3\mathsf{TIME}[N] \subseteq \mathsf{coNTIME}[N^{b^2}]$.

**Exercise 4.13:** Assume $A \leq_p^o B$ and $B \in \mathsf{NP}$ by a verifier $V$. Then $A \in \Sigma_2\mathsf{P}$ by a 2-witness verifier $U$ that views the $\exists$ witness as $w = (q, a_1, a_2, \ldots, a_q, w_{j_1}, w_{j_2}, \ldots)$ and the $\forall$ witness as $w' = (w_{k_1}, w_{k_2}, \ldots)$ where $q$ is the number of oracle queries the reduction makes, $a_i \in \{0,1\}$ is the answer to the $i^{\text{th}}$ query, and $w_i$ is a potential witness for the $i^{\text{th}}$ query, and $j_1 < j_2 < \cdots$ are the indices where $a_{j_1} = a_{j_2} = \cdots = 1$, and $k_1 < k_2 < \cdots$ are the indices where $a_{k_1} = a_{k_2} = \cdots = 0$ (so $w_1, \ldots, w_q$ are partitioned across $w$ and $w'$, with $w_i$ being in $w$ if $a_i = 1$, and $w_i$ being in $w'$ if $a_i = 0$). The size of each $w_i$ is $cN^d$ for integers $c, d$ big enough that $cN^d$ is larger than the running time of $V$ on any query the reduction might make on an input of size $N$.

```
U(x; q, a_1, ..., a_q, w_{j_1}, ...; w_{k_1}, ...):
    for i ← 1, ..., q:
        continue the execution of the reduction on input x
        if it terminates without making another query: reject
        let y_i be the i^th query
        run V(y_i; w_i)
        if it rejected and a_i = 1, or it accepted and a_i = 0: reject
        tell the reduction that a_i is the answer to the query
    continue the execution of the reduction on input x
    if it makes another query: reject
    output the same bit the reduction does
```

$U$ also rejects if the word $a_i$ doesn't fit in a single bit. If some word of $w_i$ doesn't fit in the word size for $V(y_i)$, then $U$ doesn't run $V(y_i; w_i)$—it just assumes $V(y_i; w_i)$ rejected; if this happens with $a_i = 1$ then $U$ rejects, and if this happens with $a_i = 0$ then $U$ continues running the reduction.

$U$ runs in poly time. To see that $U$ is correct, we prove that for every valid input $x$:

$$A(x) = 1 \iff \big(\exists w\ \forall w' :\ U(x; w; w') \text{ accepts}\big)$$

$\Rightarrow$: Suppose $A(x) = 1$. Let $w = (q, a_1, \ldots, a_q, w_{j_1}, \ldots)$ be the actual number of queries $q$, the actual query answers $a_i = B(y_i)$, and actual witnesses $w_i$ for the queries with $a_i = 1$. Then for all $w'$, $U(x; w; w')$ runs the reduction exactly as if the answers came from a real oracle for $B$, and it doesn't reject before the reduction finishes: It doesn't reject due to the "number of queries" checks, and it doesn't reject due to $V(y_i; w_i)$ rejecting when $a_i = 1$ (because $B(y_i) = 1$ means there indeed exists $w_i$ such that $V(y_i; w_i)$ accepts, by completeness of $V$), and it doesn't reject due to $V(y_i; w_i)$ accepting when $a_i = 0$ (because $B(y_i) = 0$ means there is no $w_i$ such that $V(y_i; w_i)$ accepts, by soundness of $V$). Thus $U(x; w; w')$ accepts for all $w'$ because the reduction accepts when $A(x) = 1$ and the queries are answered according to $B$.

$\Leftarrow$: Suppose there exists $w$ such that for all $w'$, $U(x; w; w')$ accepts. Then $w$ must have the form $(q, a_1, \ldots, a_q, w_{j_1}, \ldots)$ where each $a_i$ is a single bit. We must have $a_i = B(y_i)$ for each $i$, because if $a_i = 1$ and $V(y_i; w_i)$ accepts then $B(y_i) = 1$ (by soundness of $V$) and if $a_i = 0$ and $V(y_i; w_i)$ rejects for all $w_i$ then $B(y_i) = 0$ (by completeness of $V$). Thus $a_1, \ldots, a_q$ are the actual answers to the queries the reduction makes, and $q$ is the actual number of queries (since otherwise $U(x; w; w')$ would reject if the reduction tried to make a $(q+1)^{\text{st}}$ query or terminated

after fewer than $q$ queries). Thus $U(x; w; w')$ runs the reduction to completion, exactly as the reduction would run if the answers came from a real oracle for $B$. Since $U(x; w; w')$ accepts, the reduction accepts input $x$, so $A(x) = 1$.

Since $A \leq_p^o B$ implies $\overline{A} \leq_p^o B$ by flipping the reduction's output, we also have $\overline{A} \in \Sigma_2\mathsf{P}$ and thus $A \in \Pi_2\mathsf{P}$.

Now, suppose $B \in \mathsf{coNP}$ and thus $\overline{B} \in \mathsf{NP}$. Since $A \leq_p^o B$ implies $A \leq_p^o \overline{B}$ by flipping the oracle's answer to every query, we also have $A \in \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$ in this case.

**Exercise 4.14.a:** $\Pi_2$SAT $\in \Pi_2$P by a 2-witness verifier $V$ where $V(\varphi(x,y);a;b)$ views $a$ as an assignment to the $x$ variables and $b$ as an assignment to the $y$ variables, and $V(\varphi(x,y);a;b)$ checks that $\varphi(a,b)=1$. (Assuming $V$ uses one word per bit of $a$ and $b$: $V$ should accept if some word of $a$ doesn't fit in one bit, and otherwise reject if some word of $b$ doesn't fit in one bit.) This $V$ runs in poly time and is correct by definition:

$$\Pi_2\text{SAT}(\varphi(x,y))=1 \iff \big(\forall a\ \exists b:\ V(\varphi(x,y);a;b) \text{ accepts}\big)$$

We argue that $\Pi_2$SAT is $\Pi_2$P-hard. Suppose $A \in \Pi_2$P by a 2-witness verifier $V$. Let $B = \text{FINAL}\ \exists\ \text{OF}\ V$ (defined using an appropriate bound $T$ on $V$'s time efficiency). Then $B \in$ NP by a verifier $U$ where $U(x,w_1;w_2)$ runs $V(x;w_1;w_2)$. By the proof that SAT is NP-complete, there is a poly-time program that, given $N$, produces a CNF $\varphi(x,w_1,w_2,y)$ such that for every assignment to the variables representing the input $(x,w_1)$ and purported witness $w_2$:

$$U(x,w_1;w_2) \text{ accepts} \iff \big(\exists \text{ assignment to the } y \text{ variables}:\ \varphi(x,w_1,w_2,y)=1\big)$$

To show $A \le_p^m \Pi_2$SAT, map $x$ to the CNF $\psi(w_1,w_2,y)$ which is the contraction of $\varphi(x,w_1,w_2,y)$ with the actual value of $x$ plugged in, and with $w_1$ designated as the $\forall$ variables and $w_2, y$ designated as the $\exists$ variables. This mapping reduction runs in poly time and is correct:

$$\begin{aligned} A(x)=1 &\iff \big(\forall w_1\ \exists w_2:\ V(x;w_1;w_2) \text{ accepts}\big) \\ &\iff \big(\forall w_1\ \exists w_2:\ U(x,w_1;w_2) \text{ accepts}\big) \\ &\iff \big(\forall w_1\ \exists w_2,y:\ \varphi(x,w_1,w_2,y)=1\big) \\ &\iff \big(\forall w_1\ \exists w_2,y:\ \psi(w_1,w_2,y)=1\big) \end{aligned}$$

**Exercise 4.14.b:** $\Sigma_2\text{SAT} \in \text{NP}$ by a verifier $V$ where $V(\varphi(x,y);a)$ views $a$ as an assignment to the $x$ variables, contracts $\varphi$ by plugging in $a$ for $x$ (and leaving the $y$ variables unassigned), and checks that the contraction $\varphi(a,y)$ has no surviving clauses (that is, all clauses of $\varphi$ already got satisfied by $x = a$). This $V$ runs in poly time and is correct because:

$$\Sigma_2\text{SAT}(\varphi(x,y)) = 1 \iff \big(\exists a : \ V(\varphi(x,y);a) \text{ accepts}\big)$$

$\Leftarrow$: If there exists $a$ such that $V(\varphi(x,y);a)$ accepts, then $\varphi(a,y)$ is satisfied by all assignments to $y$ since it has no clauses. Thus $\Sigma_2\text{SAT}(\varphi(x,y)) = 1$ since there exists an assignment to $x$ such that for every assignment to $y$, $\varphi$ is satisfied.

$\Rightarrow$: If $\Sigma_2\text{SAT}(\varphi(x,y)) = 1$ then there exists $a$ such that $\varphi(a,y)$ is satisfied by all assignments to $y$. The only CNF that's satisfied by all assignments is a CNF with no clauses (since for every possible clause, there exists an assignment that doesn't satisfy the clause). Thus $\varphi(a,y)$ has no clauses and $V(\varphi(x,y);a)$ accepts.

$\Sigma_2\text{SAT}$ is NP-hard because $\text{SAT} \leq_p^m \Sigma_2\text{SAT}$ by mapping $\varphi$ to itself with all variables designated as $\exists$, and no variables (or dummy variables) designated as $\forall$.

Define $\Sigma_2\text{SAT}'$ like $\Sigma_2\text{SAT}$ but where the input $\varphi$ is a DNF. Then $\Sigma_2\text{SAT}'$ is $\Sigma_2\text{P}$-complete because $\Sigma_2\text{SAT}'$ and $\overline{\Pi_2\text{SAT}}$ (which is $\text{co}\Pi_2\text{P} = \Sigma_2\text{P}$-complete) mapping reduce to each other by just negating the input formula. This works because $\varphi$ is a DNF iff $\overline{\varphi}$ is a CNF, and we have:

$$\begin{aligned}
\Sigma_2\text{SAT}'(\varphi(x,y)) = 1 &\iff \big(\exists x \ \forall y : \ \varphi(x,y) = 1\big) \\
&\iff \big(\overline{\forall x \ \exists y : \ \overline{\varphi}(x,y) = 1}\big) \\
&\iff \overline{\Pi_2\text{SAT}}(\overline{\varphi}(x,y)) = 1
\end{aligned}$$

**Exercise 4.14.c:** Suppose $B$ is PH-complete. This means $B \in \mathsf{PH}$ and $A \leq_p^m B$ for every $A \in \mathsf{PH}$. Since $B \in \mathsf{PH}$, we have $B \in \Sigma_\ell \mathsf{P}$ for some $\ell$. For every $A \in \mathsf{PH}$, we have $A \leq_p^m B$ and thus $A \in \Sigma_\ell \mathsf{P}$ by an $\ell$-witness verifier that runs the reduction and then runs an $\ell$-witness verifier for $B \in \Sigma_\ell \mathsf{P}$ on the query. Thus $\mathsf{PH} \subseteq \Sigma_\ell \mathsf{P}$, which means the polynomial hierarchy collapses to level $\ell$.

**Exercise 4.15.a:** $\varphi$ has exactly one satisfying assignment iff both:

- There exists an assignment that satisfies $\varphi$.
- For every two assignments, if they're distinct then at least one of them doesn't satisfy $\varphi$. (This means $\varphi$ doesn't have more than one satisfying assignment.)

Viewing $w_1, w_2, w_2'$ as three assignments to $\varphi$'s variables:

$$\textsc{Unique Sat}(\varphi) = 1 \iff \big(\exists w_1 \, \forall w_2, w_2' : \ \varphi(w_1) = 1 \text{ and } (w_2 = w_2' \text{ or } \varphi(w_2) = 0 \text{ or } \varphi(w_2') = 0)\big)$$

Since "$\varphi(w_1) = 1$ and ($w_2 = w_2'$ or $\varphi(w_2) = 0$ or $\varphi(w_2') = 0$)" can be computed by a poly-time 2-witness verifier $V(\varphi; w_1; w_2, w_2')$, we have $\textsc{Unique Sat} \in \Sigma_2\mathsf{P}$.

The order of the quantifiers doesn't matter here, so we can swap them to see $\textsc{Unique Sat} \in \Pi_2\mathsf{P}$.

**Exercise 4.15.b:** We show how to map any CNF $\varphi$ to a CNF $\psi$ that has exactly one more satisfying assignment than $\varphi$ does. This will be a correct mapping reduction from $\overline{\text{SAT}}$ to UNIQUE SAT since if $\varphi$ is unsatisfiable then $\psi$ has exactly one satisfying assignment, and if $\varphi$ is satisfiable then $\psi$ has more than one satisfying assignment.

If $\varphi$ has variables $x_1 x_2 \cdots x_n$, then $\psi$ has the same variables and a fresh variable $x_0$. For each clause of $\varphi$, we put the same clause in $\psi$ but include $x_0$ in the clause. We also add the clauses $(\overline{x_0} \vee x_1) \wedge (\overline{x_0} \vee x_2) \wedge \cdots \wedge (\overline{x_0} \vee x_n)$ to $\psi$. This works because: When $x_0 = 0$, $\psi$ contracts to $\varphi$ and thus has the same (number of) satisfying assignments as $\varphi$. When $x_0 = 1$, $\psi$ contracts to $(x_1) \wedge (x_2) \wedge \cdots \wedge (x_n)$ and thus has exactly one new satisfying assignment, which assigns 1 to every variable.

This reduction has linear query size and is simple enough to be realized as a focused reduction.

**Exercise 4.15.c:** Suppose for contradiction UNIQUE SAT $\in \mathsf{TISP}[N^d, \operatorname{polylog} N]$ for some $d < \sqrt{2}$. Combining this with Exercise 4.15.b and Lemma 4.26 shows that

$$\begin{aligned}\overline{\text{SAT}} &\in \mathsf{TISP}[O(N)^d \operatorname{polylog} N, \operatorname{polylog}(O(N)) + \log N] \\ &= \mathsf{TISP}[N^d \operatorname{polylog} N, \operatorname{polylog} N] \\ &\subseteq \mathsf{TISP}[N^{d'}, \operatorname{polylog} N]\end{aligned}$$

for $d < d' < \sqrt{2}$. The latter class is closed under complement, so SAT $\in \mathsf{TISP}[N^{d'}, \operatorname{polylog} N]$, contradicting Theorem 4.1.

**Exercise 4.16.a:** We show that for all $A$ and $B$ (both decision problems or both function problems), $A \leq_\ell^m B$ iff $A \leq_f^m B$ under the modified definition of $\leq_f^m$:

$\Leftarrow$: The proof that $A \leq_f^m B$ implies $A \leq_\ell^m B$ (Lemma 4.25) doesn't mention time, so it continues to work under the modified definition of $\leq_f^m$.

$\Rightarrow$: Suppose $A \leq_\ell^m B$ with mapping $F$ computed by a log-space program $\Pi$. To compute Size of $F$ in log space, we run $\Pi$ but instead of writing the query, we just use a register to keep track of its size. To compute Word of $F$ in log space, we run $\Pi$ but use a register to keep track of how many query words would have been written so far, and we halt and output the query word when the counter is at $i$.

**Exercise 4.16.b:** $\Rightarrow$: Assume $\mathsf{P} \neq \mathsf{L}$. Consider any $A \in \mathsf{P} \setminus \mathsf{L}$, and let $B$ be the silly problem where the input is $x \in \{0,1\}$ and the output is $x$ itself. Then $A \leq_p^m B$ by a mapping reduction that runs a poly-time program for $A$ and treats the output bit as the query (the input to $B$). But $A \leq_\ell^m B$ doesn't hold, because if it did, we'd have $A \in \mathsf{L}$ by Lemma 1.15 and the fact that $B \in \mathsf{L}$.

$\Leftarrow$: Assume there exist $A$ and $B$ such that $A \leq_p^m B$ but not $A \leq_\ell^m B$. Let $F$ be a poly-time computable mapping for $A \leq_p^m B$. Then $F$ is not log-space computable, since otherwise $F$ would demonstrate that $A \leq_\ell^m B$. By Exercise 4.16.a (the contrapositive of the $\Leftarrow$ direction), either SIZE OF $F$ or WORD OF $F$ is not log-space computable. Assume WORD OF $F$ is not log-space computable. (SIZE OF $F$ is analogous.) Let $C$ be a decision problem similar to WORD OF $F$ but where the input includes an extra index specifying which bit of the target word to output. Then $C \notin \mathsf{L}$ since otherwise WORD OF $F$ would have a log-space program that runs the log-space program for $C$ for each bit-index to build the target word bit-by-bit. But $C \in \mathsf{P}$ by a program that runs a poly-time program for $F$ and extracts the desired bit. Thus $\mathsf{P} \neq \mathsf{L}$.

**Exercise 4.17.a:** The proof of Theorem 4.1 showed that if $b$ and $c$ are rational and $(b+c)b < 2$, then $\mathsf{NTIME}[N] \not\subseteq \mathsf{TISP}[N^b, N^c]$. Suppose $(d+e)d < 2$. Then there exist rational $b > d$ and $c > e$ such that $(b+c)b < 2$. A tweak to Theorem 4.29 shows that if $\textsc{Sat} \in \mathsf{TISP}[N^d, N^e]$ then $\mathsf{NTIME}[N] \subseteq \mathsf{TISP}[N^b, N^c]$, and we know the latter is false.

**Exercise 4.17.b:** If $e < 1$ then there exists $d > 1$ such that $(d+e)d < 2$, so SAT $\notin$ TISP$[N^d, N^e] \supseteq$ TISP$[N \operatorname{polylog} N, N^e]$ (Exercise 4.17.a).

**Exercise 4.18.a:** This is like the proof of Theorem 4.24, but we use the same idea again to speed up checking that $C_{e-1}$ leads to $C_e$ in $D$ steps. We partition epoch $e$ into subepochs and use two more quantifiers. This would give $\exists\,\forall\,\exists\,\forall$. To bring the number of quantifiers down to three, we flip the last two from $\exists\,\forall$ to $\forall\,\exists$ (using a "closed under complement" idea), yielding $\exists\,\forall\,\forall\,\exists$ where the middle $\forall$s can be merged.

Suppose $A \in \mathsf{TISP}[N^b, N^c]$ by a program $\Pi$ with $R$ registers, time efficiency $\leq T = aN^b$, space efficiency $\leq S = aN^c$ (for an integer $a$), and log-bounded word size $W$. Consider any valid input $x$ of size $N$. Partition the $T$ time steps into $E = N^{(b-c)/3}$ many epochs, each of duration $D = T/E$ steps. Partition each epoch into $E$ many subepochs, each of duration $D' = D/E$ steps. Define $M = S/W - R$ so $\Pi(x)$ never accesses any work segment address $\geq M$. A configuration is $1 + R + M$ words representing the program counter, all $R$ registers, and the work segment's first $M$ words.

$A \in \Sigma_3\mathsf{TIME}[N^{(b+2c)/3}]$ by a 3-witness verifier $V$ that views $w_1 = (C_0, C_1, \ldots, C_E)$ as checkpoint configurations for the epochs, views $w_2 = (e, C'_0, C'_1, \ldots, C'_E)$ as an epoch's index and checkpoint configurations for the subepochs of epoch $e$, and views $w_3 = e'$ as the index of a subepoch of epoch $e$:

$V(x; C_0, C_1, \ldots, C_E; e, C'_0, C'_1, \ldots, C'_E; e')$:
  if some $C_i$ is illegitimate, or $C_0 \neq$ initial configuration, or $C_E$ doesn't accept: reject
  if $e \notin \{1, \ldots, E\}$: accept
  if some $C'_i$ is illegitimate, or $C'_0 \neq C_{e-1}$, or $C'_E = C_e$: accept
  if $e' \notin \{1, \ldots, E\}$: reject
  run $\Pi$ for $D'$ steps from configuration $C'_{e'-1}$ with $x$ in the input segment
  if the resulting configuration is $C'_{e'}$: reject
  else: accept

To see that $V$ is correct, we show that $A(x) = 1 \iff \big(\exists w_1\ \forall w_2\ \exists w_3 :\ V(x; w_1; w_2; w_3)$ accepts$\big)$.

$$\begin{aligned}\Pi(x) \text{ accepts} \iff\ & \exists \text{ configurations } C_0, C_1, \ldots, C_E \text{ where } C_0 = \text{initial and } C_E \text{ accepts}\\ & \forall \text{ index } e \in \{1, \ldots, E\}\\ & C_{e-1} \text{ leads to } C_e \text{ in } D \text{ steps}\end{aligned}$$

$$\begin{aligned}C_{e-1} \text{ leads to } C_e \text{ in } D \text{ steps} \iff\ & \forall \text{ configurations } C'_0, C'_1, \ldots, C'_E \text{ where } C'_0 = C_{e-1} \text{ and } C'_E \neq C_e\\ & \exists \text{ index } e' \in \{1, \ldots, E\}\\ & C'_{e'-1} \text{ does not lead to } C'_{e'} \text{ in } D' \text{ steps}\end{aligned}$$

Running $\Pi$ for one subepoch takes $O(D') = O(N^b/(N^{(b-c)/3})^2) = O(N^{(b+2c)/3})$ steps, and the size of all witnesses is $O(M \cdot E) = O((N^c/\log N) \cdot N^{(b-c)/3}) = O(N^{(b+2c)/3})$ words, so $V$ runs in time $O(N^{(b+2c)/3})$.

**Exercise 4.18.b:** Suppose for contradiction $\textsc{Sat} \in \mathsf{TISP}[N^d, \text{polylog}\, N]$ for some $1 \leq d < \sqrt[3]{3}$. Since $d^3 < 3$, there exist rational constants $b > d$ and $c > 0$ such that $(b + 2c)b^2 < 3$. Defining $a = 3/(b + 2c)$, we have $a > b^2$ and $(ab + 2ac)/3 = 1$. By the same argument as Theorem 4.1, we have $\mathsf{NTIME}[N] \subseteq \mathsf{TISP}[N^b, N^c]$ and so

$$\mathsf{NTIME}[N^a] \subseteq \mathsf{TISP}[N^{ab}, N^{ac}] \subseteq \Sigma_3\mathsf{TIME}[N] \subseteq \Sigma_2\mathsf{TIME}[N^b] \subseteq \mathsf{NTIME}[N^{b^2}]$$

where the last two inclusions are simple variants of the collapse property (from $\mathsf{NTIME}[N] \subseteq \mathsf{TIME}[N^b]$). This contradicts Theorem 4.13.

**Exercise 4.19.a:** This is like the proof of Theorem 4.24, but we use the same idea again to speed up checking that $C_{e-1}$ leads to $C_e$ in $D$ steps. We keep repeating this idea, using more quantifiers to achieve more speedup. That is, we partition each epoch into smaller epochs, and partition each of those into even smaller epochs, and so on. Since every log-space program runs in poly time, it only takes a constant number of quantifiers to get the epoch size down to $O(N)$.

We prove that for every positive integer $b$, $\mathsf{TISP}[N^b, \log N] \subseteq \Sigma_{2b-2}\mathsf{TIME}[N]$. (This can be improved to $\mathsf{TISP}[N^b, \log N] \subseteq \Sigma_b\mathsf{TIME}[N]$ using the idea from Exercise 4.18.a.) This implies the theorem since $\mathsf{L} = \bigcup_b \mathsf{TISP}[N^b, \log N] \subseteq \bigcup_b \Sigma_{2b-2}\mathsf{TIME}[N] = \bigcup_\ell \Sigma_\ell \mathsf{TIME}[N]$.

Suppose $A \in \mathsf{TISP}[N^b, \log N]$ by a register-only program $\Pi$ with $R$ registers, time efficiency $\leq T = aN^b$ (for an integer $a$), and log-bounded word size $W$. Consider any valid input $x$ of size $N$. For each $d \in \{0, 1, \ldots, b-1\}$, partition the $T$ time steps into $N^d$ epochs, each of duration $T/N^d$ steps—call these "depth-$d$" epochs. That is, regard all $T$ time steps as one big depth-0 epoch, and for each $d \in \{1, 2, \ldots, b-1\}$, partition each depth-$(d-1)$ epoch into $N$ many depth-$d$ epochs. Each depth-$(b-1)$ epoch has duration $T/N^{b-1} = aN$ steps. A configuration is $1 + R$ words representing the program counter and all $R$ registers. Assume there's a unique accepting configuration: Whenever $\Pi$ wants to accept, it branches to a unique section of code that puts 0 in every register and then accepts. Define $C_{0,0}$ = initial configuration, and $C_{0,1}$ = unique accepting configuration, and $e_0 = 1$ (the index of the depth-0 epoch).

$A \in \Sigma_{2b-2}\mathsf{TIME}[N]$ by this $(2b-2)$-witness verifier $V$:

$V(x\ ;\ C_{1,0}, \ldots, C_{1,N}\ ;\ e_1\ ;\ C_{2,0}, \ldots, C_{2,N}\ ;\ e_2\ ;\ \cdots\ ;\ C_{b-1,0}, \ldots, C_{b-1,N}\ ;\ e_{b-1})$:
  for $d \leftarrow 1, 2, \ldots, b-1$:
    if some $C_{d,i}$ is illegitimate, or $C_{d,0} \neq C_{d-1,e_{d-1}-1}$, or $C_{d,N} \neq C_{d-1,e_{d-1}}$: reject
    if $e_d \notin \{1, \ldots, N\}$: accept
  run $\Pi$ for $aN$ steps from configuration $C_{b-1,e_{b-1}-1}$ with $x$ in the input segment
  if the resulting configuration is $C_{b-1,e_{b-1}}$: accept
  else: reject

To see that $V$ is correct, we show that:

$$A(x) = 1 \iff \big(\exists w_1\ \forall w_2 \cdots \exists w_{2b-3}\ \forall w_{2b-2} :\ V(x; w_1; w_2; \cdots; w_{2b-3}; w_{2b-2}) \text{ accepts}\big)$$

This holds because:

$\Pi(x)$ accepts

$\iff \exists\ C_{1,0},\ldots,C_{1,N}$ where $C_{1,0} = C_{0,e_0-1} =$ initial and $C_{1,N} = C_{0,e_0} =$ accepting
$\forall\ e_1 \in \{1,\ldots,N\}$
$\underbrace{C_{1,e_1-1} \text{ leads to } C_{1,e_1} \text{ in } T/N^1 \text{ steps}}$

$\iff \exists\ C_{2,0},\ldots,C_{2,N}$ where $C_{2,0} = C_{1,e_1-1}$ and $C_{2,N} = C_{1,e_1}$
$\forall\ e_2 \in \{1,\ldots,N\}$
$\underbrace{C_{2,e_2-1} \text{ leads to } C_{2,e_2} \text{ in } T/N^2 \text{ steps}}$

$\iff \cdots$

$\iff \exists\ C_{b-1,0},\ldots,C_{b-1,N}$ where $C_{b-1,0} = C_{b-2,e_{b-2}-1}$ and $C_{b-1,N} = C_{b-2,e_{b-2}}$
$\forall\ e_{b-1} \in \{1,\ldots,N\}$
$C_{b-1,e_{b-1}-1}$ leads to $C_{b-1,e_{b-1}}$ in $T/N^{b-1} = aN$ steps

Running $\Pi$ for one depth-$(b-1)$ epoch takes $O(aN) = O(N)$ steps, and the size of each witness is $\leq (1+R)\cdot(N+1) = O(N)$ words, so $V$ runs in time $O(N)$.

**Exercise 4.19.b:** Suppose for contradiction $\textsc{Sat} \in \mathsf{TIME}[N] \cap \mathsf{L}$. Since $\textsc{Sat} \in \mathsf{TIME}[N]$, we have $\mathsf{NTIME}[N] \subseteq \mathsf{TIME}[N \operatorname{polylog} N]$ (Corollary 2.33). By padding, $\mathsf{NTIME}[N \operatorname{polylog} N] \subseteq \mathsf{TIME}[N \operatorname{polylog} N]$ and $\mathsf{coNTIME}[N \operatorname{polylog} N] \subseteq \mathsf{TIME}[N \operatorname{polylog} N]$ (because $\mathsf{TIME}[N \operatorname{polylog} N]$ is closed under complement). Using these to peel off quantifiers as in the proof of Theorem 4.21, for every $\ell$ we have:

$$\begin{aligned}
\Sigma_\ell \mathsf{TIME}[N] &\subseteq \Sigma_{\ell-1} \mathsf{TIME}[N \operatorname{polylog} N] \\
&\subseteq \Sigma_{\ell-2} \mathsf{TIME}[N \operatorname{polylog} N] \\
&\subseteq \cdots \\
&\subseteq \Sigma_1 \mathsf{TIME}[N \operatorname{polylog} N] \\
&\subseteq \mathsf{TIME}[N \operatorname{polylog} N]
\end{aligned}$$

Since $\textsc{Sat} \in \mathsf{L}$, we have $\mathsf{L} = \mathsf{NP}$ (Theorem 2.40). Using Exercise 4.19.a,

$$\begin{aligned}
\mathsf{TIME}[N^2] &\subseteq \mathsf{P} \\
&\subseteq \mathsf{NP} \\
&\subseteq \mathsf{L} \\
&\subseteq \bigcup_\ell \Sigma_\ell \mathsf{TIME}[N] \\
&= \mathsf{TIME}[N \operatorname{polylog} N]
\end{aligned}$$

which contradicts Theorem 4.7.

**Exercise 4.20.a:** This is like the proof of Theorem 4.24, but we use a third quantifier for the section of the witness that leads the time-space-efficient verifier from $C_{e-1}$ to $C_e$ in $D$ steps.

Suppose $A \in \mathsf{NTISP}[N^b, N^c]$ by a streaming verifier $V$ with $R$ registers, time efficiency $\leq T = aN^b$, space efficiency $\leq S = aN^c$ (for an integer $a$), and log-bounded word size $W$. Consider any valid input $x$ of size $N$. Partition the $T$ time steps into $E = N^{(b-c)/2}$ many epochs, each of duration $D = T/E$ steps, and assume $E$ and $D$ are integers. Define $M = S/W - R$ so $V(x;w)$ never accesses any work segment address $\geq M$. A configuration is $1+R+M$ words representing the program counter, all $R$ registers, and the work segment's first $M$ words.

$A \in \Sigma_3\mathsf{TIME}[N^{(b+c)/2}]$ by a 3-witness verifier $V'$ that views $w_1 = (C_0, C_1, \ldots, C_E)$ as a tuple of purported checkpoint configurations, views $w_2 = e$ as an epoch's index, and views $w_3 \in (\{0,1\}^W)^D$ as a section of a purported witness for $V$:

$V'(x; C_0, C_1, \ldots, C_E; e; w_3)$:
  if some $C_i$ is illegitimate, or $C_0 \neq$ initial configuration, or $C_E$ doesn't accept: reject
  if $e \notin \{1, \ldots, E\}$: accept
  if some word of $w_3$ doesn't fit in $W$ bits: reject
  run $V$ for $D$ steps from configuration $C_{e-1}$ with $x$ in the input segment and
    $w_3$ in the witness segment (simulating one-way sequential access with random access)
  if the resulting configuration is $C_e$: accept
  else: reject

To see that $V'$ is correct, we show that $A(x) = 1 \iff \big(\exists w_1\ \forall w_2\ \exists w_3 :\ V'(x; w_1; w_2; w_3)$ accepts$\big)$.

$\Rightarrow$: Suppose $A(x) = 1$ and let $w$ be such that $V(x;w)$ accepts. Let $C_0, \ldots, C_E$ be the actual checkpoints of $V(x;w)$—that is, $C_i$ is the configuration after epoch $i$. Then $C_E$ is an accept configuration since $V(x;w)$ accepts within $T$ steps. For all $e \in \{1, \ldots, E\}$, $C_{e-1}$ leads to $C_e$ in $D$ steps while reading some section $w_3^e$ of $w$, so $V'(x; C_0, \ldots, C_E; e; w_3^e)$ accepts.

$\Leftarrow$: Suppose there exists $C_0, \ldots, C_E$ such that for all $e \in \{1, \ldots, E\}$, there exists $w_3^e$ such that $V'(x; C_0, \ldots, C_E; e; w_3^e)$ accepts. Instead of each $w_3^e$ being exactly $D$ words, we let $w_3^e$ be only the words that are actually read by the corresponding run of $V$. Then we let $w$ be the concatenation $(w_3^1, w_3^2, \ldots, w_3^E)$. Since $C_0$ is the initial configuration and $V'$ accepts with $e = 1$, $C_1$ is the configuration of $V(x;w)$ after epoch 1. This implies that since $V'$ accepts with $e = 2$, $C_2$ is the configuration of $V(x;w)$ after epoch 2, and so on. Thus $V(x;w)$ reaches the accept configuration $C_E$, so $A(x) = 1$ since $V(x;w)$ accepts.

Running $V$ for one epoch takes $O(D) = O(N^{(b+c)/2})$ steps, and $w_1$ is $(1+R+M)(E+1) = O((N^c/\log N) \cdot N^{(b-c)/2}) = O(N^{(b+c)/2})$ words, and $w_2$ is one word, and $w_3$ is $D$ words, so $V'$ runs in time $O(N^{(b+c)/2})$.

**Exercise 4.20.b:** This is similar to the proof of Theorem 4.1.

Suppose for contradiction $\textsc{Sat} \in \mathsf{coNTISP}[N^d, \operatorname{polylog} N]$ for some $1 \leq d < \sqrt[3]{2}$. Since $d^3 < 2$, there exist rational constants $b > d$ and $c > 0$ such that $(b+c)b^2 < 2$. Defining $a = 2/(b+c)$, we have $a > b^2$ and $(ab + ac)/2 = 1$. We have $\mathsf{NTIME}[N] \subseteq \mathsf{coNTISP}[N^b, N^c]$ by a straightforward adaptation of Theorem 4.29 from solvers to verifiers, and so

$$\mathsf{NTIME}[N^a] \subseteq \mathsf{coNTISP}[N^{ab}, N^{ac}] \subseteq \Pi_3\mathsf{TIME}[N] \subseteq \Pi_2\mathsf{TIME}[N^b] \subseteq \mathsf{coNTIME}[N^{b^2}]$$

where the first inclusion is by padding, the second inclusion is by Exercise 4.20.a, and the last two inclusions are like Exercise 4.12 but merging adjacent quantifiers as in Exercise 4.11. This contradicts Exercise 4.8.a.

**Exercise 4.21.a:** For every positive integer $b$, we have $\mathsf{NTISP}[N^b, \log N] \subseteq \Sigma_{2b-1}\mathsf{TIME}[N]$ by the same proof as Exercise 4.19.a but using the $(2b-1)^{\text{st}}$ quantifier for a section of the time-space-efficient verifier's witness (one depth-$(b-1)$ epoch's worth), as in Exercise 4.20.a. By Lemma 3.8, $\mathsf{NL} = \bigcup_b \mathsf{NTISP}[N^b, \log N] \subseteq \bigcup_b \Sigma_{2b-1}\mathsf{TIME}[N] = \bigcup_\ell \Sigma_\ell \mathsf{TIME}[N]$.

**Exercise 4.21.b:** Suppose for contradiction $\textsc{Sat} \in \mathsf{coNTIME}[N] \cap \mathsf{NL}$. Since $\textsc{Sat} \in \mathsf{coNTIME}[N]$, we have $\mathsf{NTIME}[N] \subseteq \mathsf{coNTIME}[N \operatorname{polylog} N]$ by a straightforward adaptation of Corollary 2.33 from solvers to verifiers. By padding, $\mathsf{NTIME}[N \operatorname{polylog} N] = \mathsf{coNTIME}[N \operatorname{polylog} N]$. Using this to peel off quantifiers as in Exercise 4.11, for every $\ell$ we have:

$$\begin{aligned} \Sigma_\ell \mathsf{TIME}[N] &\subseteq \Sigma_{\ell-1} \mathsf{TIME}[N \operatorname{polylog} N] \\ &\subseteq \Sigma_{\ell-2} \mathsf{TIME}[N \operatorname{polylog} N] \\ &\subseteq \cdots \\ &\subseteq \Sigma_1 \mathsf{TIME}[N \operatorname{polylog} N] \end{aligned}$$

Since $\textsc{Sat} \in \mathsf{NL}$, we have $\mathsf{NL} = \mathsf{NP}$ (Theorem 2.40 and Lemma 3.20). Using Exercise 4.21.a,

$$\begin{aligned} \mathsf{NTIME}[N^2] &\subseteq \mathsf{NP} \\ &\subseteq \mathsf{NL} \\ &\subseteq \bigcup_\ell \Sigma_\ell \mathsf{TIME}[N] \\ &= \mathsf{NTIME}[N \operatorname{polylog} N] \end{aligned}$$

which contradicts Theorem 4.13.

**Exercise 5.1:** $\Leftarrow$: $\mathsf{P} \subseteq \mathsf{P/poly}$, so if $\mathsf{P/poly} = \mathsf{L/poly}$ then $\mathsf{P} \subseteq \mathsf{P/poly} = \mathsf{L/poly}$.

$\Rightarrow$: Assume $\mathsf{P} \subseteq \mathsf{L/poly}$. Since $\mathsf{L/poly} \subseteq \mathsf{P/poly}$, it suffices to show $\mathsf{P/poly} \subseteq \mathsf{L/poly}$. Suppose $A \in \mathsf{P/poly}$ by a poly-time program $\Pi$ with advice $a_1, a_2, \ldots$. Let $B$ be the problem $A$ WITH $\Pi$'S ADVICE defined in the proof of Theorem 5.12. Since $B \in \mathsf{P}$, we have $B \in \mathsf{L/poly}$ by a log-space program $\Pi'$ with advice $b_1, b_2, \ldots$. Then $A \in \mathsf{L/poly}$ by the log-space program $\Pi''$ with advice $c_1, c_2, \ldots$ where $c_N = (a_N, b_M)$ where $M$ is the size of $(x, a_N)$, and $\Pi''(x; a_N, b_N)$ runs $\Pi'(x, a_N; b_M)$ (adjusting the latter's "read" and "read-advice" instructions to find the desired words within $x; a_N, b_N$). This is correct since $\Pi''(x; a_N, b_N) = \Pi'(x, a_N; b_M) = B(x, a_N) = A(x)$, and $\Pi''$ runs in space $O(\log M) = O(\log N)$.

**Exercise 5.2:** Consider any decision problem $A$.

$\Leftarrow$: Assume there exists a sparse decision problem $B$ such that $A \leq_p^o B$. Let $p$ be a polynomial such that for all $M$, $B(y) = 1$ holds for $\leq p(M)$ many inputs $y$ of size $\leq M$ (not just size $M$). Let $q$ be a polynomial such that for every valid input to $A$ of size $N$, the reduction makes queries of size $\leq q(N)$. For all $N$, define advice $a_N$ that encodes the list of all $y$ of size $\leq q(N)$ such that $B(y) = 1$. There are $\leq p(q(N)) \in \text{poly}\, N$ many such $y$, so $a_N$ has size $\text{poly}\, N$. We define a poly-time nonuniform program for $A$ using the following algorithm:

```
on input x of size N and advice a_N:
    run the reduction on input x, but whenever it makes a query y:
        if y is in the list a_N: use 1 as the answer to the query
        else: use 0 as the answer to the query
    output whatever the reduction outputs
```

Each query gets answered correctly: If $y$ is in the list $a_N$ then $B(y) = 1$, and if $y$ is not in the list $a_N$ then $B(y) = 0$ because $y$ has size $\leq q(N)$ and $a_N$ contains all $y$ of size $\leq q(N)$ such that $B(y) = 1$. Since the reduction solves $A$ assuming each query gets answered correctly, this program with advice $a_1, a_2, \ldots$ solves $A$. Thus $A \in \mathsf{P/poly}$.

$\Rightarrow$: Assume $A \in \mathsf{P/poly}$ by a poly-time program $\Pi$ with advice $a_1, a_2, \ldots$. Define a decision problem $B$ with word size 1 as follows: For every $y \in \{0,1\}^N$, $B(y)$ is the $y^{\text{th}}$ bit of $a_N$ (after flattening the tuple of words into just a bit string, and viewing $y$ as a binary number). If the $y^{\text{th}}$ bit would be past the end of $a_N$, then $B(y) = 0$. This $B$ is sparse because $a_N$ has only $\text{poly}\, N$ many 1s, which means $B(y) = 1$ for only $\text{poly}\, N$ many $y \in \{0,1\}^N$. Let $p$ be a polynomial such that for all $N$, $a_N$ has $\leq p(N)$ bits. We define a poly-time oracle reduction from $A$ to $B$:

```
on input x of size N:
    if p(N) > 2^N:
        obtain A(x) from a lookup table and output it
    else:
        obtain a_N by querying the oracle on input y for all y ∈ {0,1}^N with y < p(N)
        run Π(x; a_N) and output the same answer
```

Since $p = o(2^N)$, there are only a constant number of inputs $x$ whose size $N$ is such that $p(N) > 2^N$, so the values of $A(x)$ for these $x$ can be hardcoded in the reduction. For all other $x$, the reduction gets the bits of $a_N$ from the oracle for $B$, so the reduction outputs $A(x)$ since $\Pi(x; a_N)$ does. Thus $A \leq_p^o B$.

**Exercise 5.3:** To show NEXP/poly = coNEXP/poly, it suffices to show NEXP/poly $\subseteq$ coNEXP/poly. Suppose $A \in$ NEXP/poly by a verifier $V$ with time efficiency $\leq T = 2^{eN^d}$ (for integers $d, e$) and poly-size advice $a_1, a_2, \ldots$. Then $\overline{A} \in$ NEXP/poly by the following exponential-time verifier $V'$ with poly-size advice $a'_1, a'_2, \ldots$. For each $N$, define $a'_N = (a_N, c_N)$ where $c_N$ is the number of not-necessarily-valid inputs $y$ of size $N$ such that there exists $w$ with $T$ words for which $V(y; w; a_N)$ accepts within $T$ steps. (If $y$ is valid, this just means $A(y) = 1$.) On input $x$ of size $N$, $V'$ views its purported witness $w'$ as having the following for each not-necessarily-valid input $y$ of size $N$: a bit $b_y$ indicating whether there exists $w_y$ with $T$ words for which $V(y; w_y; a_N)$ accepts within $T$ steps (that is, whether $y$ is included in the count $c_N$) and if $b_y = 1$ then $w'$ also has such a $w_y$. Then $V'(x; w'; a'_N)$ accepts iff $b_x = 0$ and $\sum_y b_y = c_N$ and $V(y; w_y; a_N)$ accepts within $T$ steps for each $y$ with $b_y = 1$.

This is correct because: If $A(x) = 0$ then $V'(x; w'; a'_N)$ accepts for the intended $w'$. If $A(x) = 1$ then $V'(x; w'; a'_N)$ rejects for every $w'$ since if $b_x = 0$ and $\sum_y b_y = c_N$ then there must exist $y$ such that: $b_y = 1$ but there does not exist $w_y$ with $T$ words for which $V(y; w_y; a_N)$ accepts within $T$ steps.

The time efficiency of $V'$ is only a factor $2^{O(N \log N)}$ greater than the time efficiency of $V$ (since there are $2^{O(N \log N)}$ many possibilities of $y$ since $A$ has word size $O(\log N)$) and so is still exponential. The advice size of $V'$ is only $+O(N \log N)$ bits greater than the advice size of $V$ and so is still polynomial.

**Exercise 5.4:** This is like the proof that if $\mathsf{NP} \subseteq \mathsf{P/poly}$ then $\mathsf{PH} = \Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$ (Theorem 5.5) but instead of using a quantifier for the advice, we try all possibilities of the advice in poly time. However, Lemma 5.4 doesn't preserve the logarithmic advice size, because it combines the advice for many input sizes. We adjust the search-to-decision reduction so all queries have the same size, so we only need the advice for that particular query size.

Formally, consider SAT with an input encoding that allows any amount of padding. SAT SEARCH $\leq_p^o$ SAT (Theorem 2.14) by a reduction that, on an input of size $N$, pads all queries up to a common size $M \in \text{poly}\, N$. Assuming $\mathsf{NP} \subseteq \mathsf{P/log}$, we have SAT $\in \mathsf{P/log}$ by a poly-time program $\Pi$ with log-size advice $a_1, a_2, \ldots$. Thus SAT SEARCH has a poly-time program $\Pi'$ that, on an input of size $N$, takes advice $a'_N = a_M$, runs the reduction to SAT, and answers each query (which has size $M$) using $\Pi$ with advice $a_M$. Assume $\Pi'$ clocks itself so it runs in poly time even with wrong advice. Assume $a'_N$ consists of $c$ words of size $W = O(\log M) = O(\log N)$ for a constant $c$.

To prove $\mathsf{P} = \mathsf{NP}$, it suffices to prove SAT $\in \mathsf{P}$ since SAT is NP-complete (Theorem 2.10). Here's an algorithm for SAT on input $\varphi$ of size $N$:

```
for every b consisting of c words of size W:
    run Π'(φ; b) to get an assignment y
    if y satisfies φ: accept
reject
```

This algorithm is poly-time since $\Pi'$ is poly-time even when $b \neq a'_N$, and there are $2^{cW} = \text{poly}\, N$ many possibilities of $b$. To show this algorithm is correct, we argue that it accepts iff $\varphi$ is satisfiable:

$\Rightarrow$: If the algorithm accepts, then it found a satisfying assignment for $\varphi$, so $\varphi$ is satisfiable.

$\Leftarrow$: If $\varphi$ is satisfiable, then when $b = a'_N$, $\Pi'(\varphi; b)$ finds a satisfying assignment for $\varphi$ by the correctness of $\Pi'$, so the algorithm accepts.

**Exercise 5.5:** Assume $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ and thus $\Pi_1\mathsf{P} = \mathsf{coNP} \subseteq \mathsf{NP/poly}$. To prove $\mathsf{PH} = \Sigma_3\mathsf{P} = \Pi_3\mathsf{P}$, it suffices to prove $\Pi_3\mathsf{P} \subseteq \Sigma_3\mathsf{P}$ (Theorem 4.23). Suppose $A \in \Pi_3\mathsf{P}$ by a 3-witness verifier $V$:

$$A(x) = 1 \iff \big(\forall w_1\ \exists w_2\ \forall w_3 :\ V(x; w_1; w_2; w_3) \text{ accepts}\big)$$

First attempt at showing $A \in \Sigma_3\mathsf{P}$: The assumption $\Pi_1\mathsf{P} \subseteq \mathsf{NP/poly}$ turns $\forall w_3$ into two things: a $\exists$ quantifier that can be merged with $\exists w_2$, and advice that doesn't depend on $w_1, w_2$ and can thus be specified with a $\exists$ quantifier preceding $\forall w_1\ \exists w_2$. In more detail: Final $\forall$ of $V \in \Pi_1\mathsf{P}$ by a verifier $U$ where $U(x, w_1, w_2; w_3)$ runs $V(x; w_1; w_2; w_3)$. Thus by assumption, Final $\forall$ of $V \in \mathsf{NP/poly}$ by a verifier $U'$ with advice $a_1, a_2, \ldots$. For every input $(x, w_1, w_2)$ of size $M$:

$$\big(\forall w_3 :\ V(x; w_1; w_2; w_3) \text{ accepts}\big) \iff \big(\exists z :\ U'(x, w_1, w_2; z; a_M) \text{ accepts}\big)$$

Assume $U'$ clocks itself so it runs in poly time even with wrong advice ($\neq a_M$). Then letting $V'(x; b; w_1; w_2, z)$ run $U'(x, w_1, w_2; z; b)$, we envision the correctness of $V'$ (for showing $A \in \Sigma_3\mathsf{P}$) as:

$$\begin{aligned} A(x) = 1 &\iff \big(\quad \forall w_1\ \exists w_2\ \forall w_3 :\ V(x; w_1; w_2; w_3) \text{ accepts}\big) \\ &\iff \big(\quad \forall w_1\ \exists w_2\ \exists z :\ U'(x, w_1, w_2; z; a_M) \text{ accepts}\big) \\ &\iff \big(\exists b\ \forall w_1\ \exists w_2\ \exists z :\ U'(x, w_1, w_2; z; b) \text{ accepts}\big) \\ &\iff \big(\exists b\ \forall w_1\ \ \exists w_2, z\ :\ V'(x; b; w_1; w_2, z) \text{ accepts}\big) \end{aligned}$$

$\Rightarrow$ holds with $b = a_M$ where $M$ is the common size of all $(x, w_1, w_2)$, but there's an issue with $\Leftarrow$:

$$\big(\exists b\ \forall w_1\ \exists w_2\ \exists z :\ U'(x, w_1, w_2; z; b) \text{ accepts}\big) \not\Rightarrow \big(\forall w_1\ \exists w_2\ \forall w_3 :\ V(x; w_1; w_2; w_3) \text{ accepts}\big)$$

because when $b \neq a_M$, we might have:

$$\big(\exists z :\ U'(x, w_1, w_2; z; b) \text{ accepts}\big) \text{ but } \big(\exists w_3 :\ V(x; w_1; w_2; w_3) \text{ rejects}\big)$$

Here's how to check that $U'$ never makes such a mistake, given any particular choice of $x, b, w_1$:

$$\forall w_2' :\ \Big(\big(\forall z' :\ U'(x, w_1, w_2'; z'; b) \text{ rejects}\big) \text{ or } \big(\forall w_3' :\ V(x; w_1; w_2'; w_3') \text{ accepts}\big)\Big)$$

This is equivalent to the following statement, which we name $C(x, b, w_1)$:

$$\forall w_2', z', w_3' :\ \big(U'(x, w_1, w_2'; z'; b) \text{ rejects or } V(x; w_1; w_2'; w_3') \text{ accepts}\big)$$

We merge this $\forall$ quantifier with $\forall w_1$. Thus, to fix our first attempt, we redefine $V'$ such that:

$$A(x) = 1 \iff \big(\exists b\ \ \forall w_1, w_2', z', w_3'\ \ \exists w_2, z :\ V'(x; b; w_1, w_2', z', w_3'; w_2, z) \text{ accepts}\big)$$

$V'(x; b; w_1, w_2', z', w_3'; w_2, z)$:
  if $U'(x, w_1, w_2; z; b)$ accepts and $\big(U'(x, w_1, w_2'; z'; b)$ rejects or $V(x; w_1; w_2'; w_3')$ accepts$\big)$:
    accept
  else: reject

$\Rightarrow$: Suppose $A(x) = 1$. Let $b = a_M$ where $M$ is the common size of all $(x, w_1, w_2)$. Consider any $w_1, w_2', z', w_3'$. By correctness of $V$, we can let $w_2$ be such that $\forall w_3 : V(x; w_1; w_2; w_3)$ accepts. By correctness of $U'$, we can let $z$ be such that $U'(x, w_1, w_2; z; b)$ accepts. By correctness of $U'$, if $U'(x, w_1, w_2'; z'; b)$ accepts then $V(x; w_1; w_2'; w_3')$ accepts. Thus $V'(x; b; w_1, w_2', z', w_3'; w_2, z)$ accepts.

$\Leftarrow$: Suppose $A(x) = 0$. Consider any $b$. By correctness of $V$, we can let $w_1$ be such that $\forall w_2 \; \exists w_3 : V(x; w_1; w_2; w_3)$ rejects. Consider two cases for whether or not $C(x, b, w_1)$ is true: If it is true, then for all $w_2$, since $\exists w_3 : V(x; w_1; w_2; w_3)$ rejects, we must have $\forall z : U'(x, w_1, w_2; z; b)$ rejects and thus $V'(x; b; w_1, w_2', z', w_3'; w_2, z)$ rejects. If it is false, then we can let $w_2', z', w_3'$ be such that $U'(x, w_1, w_2'; z'; b)$ accepts and $V(x; w_1; w_2'; w_3')$ rejects, and thus $V'(x; b; w_1, w_2', z', w_3'; w_2, z)$ rejects.

**Exercise 5.6:** Assume PSPACE $\subseteq$ P/poly. Since QSAT $\in$ PSPACE $\subseteq$ P/poly (Lemma 3.1) and PLAY QSAT $\leq_p^o$ QSAT (Exercise 3.8), PLAY QSAT has a poly-time program $\Pi$ with advice $a_1, a_2, \ldots$ (Lemma 5.4). Assume $\Pi$ clocks itself so it runs in poly time even with wrong advice. We use an input encoding of PLAY QSAT that allows any amount of padding, so for every $\varphi$ we can pad all inputs $(\varphi, y_1 y_2 \cdots y_{i-1})$ up to a common size $M$.

To prove PSPACE $= \Sigma_2$P $= \Pi_2$P, it suffices to prove QSAT $\in \Sigma_2$P since QSAT is PSPACE-complete (Theorem 3.4): This implies PSPACE $\subseteq \Sigma_2$P since for every $A \in$ PSPACE, we have $A \leq_p^m$ QSAT $\in \Sigma_2$P and thus $A \in \Sigma_2$P (like the analogous fact for NP in Exercise 2.4), and PSPACE is closed under complement.

To show QSAT $\in \Sigma_2$P, we design a 2-witness verifier $V$ such that for every CNF $\varphi(x_1 \cdots x_n)$ where $n$ is even:

$$\text{QSAT}(\varphi) = 1 \iff \big(\exists b \ \forall y_2 y_4 \cdots y_n : \ V(\varphi; b; y_2 y_4 \cdots y_n) \text{ accepts}\big)$$

The idea is that $\Pi$ with advice $b$ gives a strategy for P1, and $y_2 y_4 \cdots y_n$ are the bits P2 assigns to $x_2 x_4 \cdots x_n$, and $V$ simulates the gameplay and accepts iff P1 wins.

```
V(φ; b; y_2 y_4 ⋯ y_n):
    for i ← 1, 3, 5, …, n − 1:
        run Π(φ, y_1 y_2 ⋯ y_{i−1}; b)
        if it reported that no solution exists: reject
        else: let y_i be the bit output by Π
    if y_1 y_2 ⋯ y_n satisfies φ: accept
    else: reject
```

$\Rightarrow$: Suppose QSAT$(\varphi) = 1$ and thus P1 has a winning strategy. Let $b = a_M$. Then $\Pi$ with advice $b$ gives a winning strategy for P1, which means for all $y_2 y_4 \cdots y_n$, P1 wins and thus $V(\varphi; b; y_2 y_4 \cdots y_n)$ accepts.

$\Leftarrow$: Suppose QSAT$(\varphi) = 0$ and thus P1 does not have a winning strategy. Then for all $b$, $\Pi$ with advice $b$ does not give a winning strategy for P1, which means there exists $y_2 y_4 \cdots y_n$ such that P2 wins and thus $V(\varphi; b; y_2 y_4 \cdots y_n)$ rejects.

$V$ is poly-time since $b$ and $y_2 y_4 \cdots y_n$ have poly$N$ size, and $\Pi(\varphi, y_1 y_2 \cdots y_{i-1}; b)$ runs in poly$M$ = poly$N$ time even when $b \neq a_M$, and evaluating $\varphi(y_1 y_2 \cdots y_n)$ takes poly$N$ time.

**Exercise 5.7:** This is like the proof that if PSPACE $\subseteq$ P/poly then PSPACE $= \Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$ (Theorem 5.6), except now the 2-witness verifier doesn't have time to look at an entire exponentially long configuration. Instead, it uses another $\forall$ to quantify over all words of the configuration. Each word can be checked in terms of a constant number of words of the previous configuration, and each word has poly size, so the checking takes poly time.

Assume EXP $\subseteq$ P/poly. To prove EXP $= \Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$, it suffices to prove EXP $\subseteq \Sigma_2\mathsf{P}$ since $\Sigma_2\mathsf{P} \subseteq$ EXP and EXP is closed under complement.

Suppose $A \in$ EXP by a program $\Pi$ with $L$ lines, $R$ registers, time efficiency $\leq T = 2^{cN^d}$, space efficiency $\leq S = 2^{cN^d}$ (for integers $c, d \geq 1$), and poly-bounded word size $W$. Consider any valid input $x$ of size $N$. Define $M = S/W - R$ so $\Pi(x)$ never accesses any work segment address $\geq M$. A configuration in $\{0,1\}^{\log L} \times (\{0,1\}^W)^R \times (\{0,1\}^W)^M$ represents the program counter, all $R$ registers, and the work segment's first $M$ words. When $\Pi(x)$ terminates, imagine it continues taking steps but with the configuration frozen. For convenience, assume $W \geq \log L$ for all $N$ so the PC can have the same size as all other words.

Consider the following function problem and its decision version, in which $s$ is a location from {PC, Reg[0], ..., Reg[$R-1$], Mem[0], ..., Mem[$M-1$]} represented with poly$N$ bits:

| Word of Configuration of $\Pi$ | |
|---|---|
| Input: | Valid input $x$ of size $N$ to $A$, $t \in \{0,1,\ldots,T\}$, and location $s$ |
| Output: | The word at location $s$ of the configuration after $t$ steps of $\Pi(x)$ |

| Bit of Word of Configuration of $\Pi$ | |
|---|---|
| Input: | Valid input $x$ of size $N$ to $A$, $t \in \{0,1,\ldots,T\}$, location $s$, and $j \in \{0,\ldots,W-1\}$ |
| Output: | The $j^{\text{th}}$ bit of the word at location $s$ of the configuration after $t$ steps of $\Pi(x)$ |

Bit of Word of Configuration of $\Pi \in$ EXP by a program that runs $\Pi(x)$ for $t$ steps (using extra work memory for the clock) and then extracts the $j^{\text{th}}$ bit of the word at location $s$ of the configuration. By assumption, Bit of Word of Configuration of $\Pi \in$ P/poly. We have Word of Configuration of $\Pi \leq_p^o$ Bit of Word of Configuration of $\Pi$ by a reduction that queries the oracle for all $j \in \{0,\ldots,W-1\}$. Thus Word of Configuration of $\Pi$ has a poly-time program $\Pi'$ with advice $a_1, a_2, \ldots$ (Lemma 5.4). Assume $\Pi'$ clocks itself so it runs in poly time even with wrong advice.

To show $A \in \Sigma_2\mathsf{P}$, we design a 2-witness verifier $V$ such that:

$$A(x) = 1 \iff \big(\exists b\ \forall t,s :\ V(x;b;t,s) \text{ accepts}\big)$$

$V(x; b; t, s)$:
    run $\Pi'(x, t, s; b)$ to get a word $y$
    if $t = 0$ and $y$ isn't the word at location $s$ of the initial configuration: reject
    if $t = T$ and $s =$ PC and line $y$ of $\Pi$ isn't an accept instruction: reject
    if $t > 0$:
        run $\Pi'(x, t-1, s'; b)$ for every location $s'$ that $y$ depends on
        if $y$ doesn't follow from these words after one step of $\Pi(x)$: reject
    accept

$V$ has $O(\log N)$ word size, so each word of $\Pi$ takes many words of $V$.

The check when $t > 0$ is like the $\varphi_{t,s}$ check in the proof that SAT is NP-complete (§2.8.2). In more detail: $V$ first runs $\Pi'(x, t-1, \text{PC}; b)$ to see which instruction $\Pi(x)$ would execute in step $t$ (according to $\Pi'$).

- If this instruction is "load Reg$[i] \leftarrow$ Mem[Reg$[j]$]" and $s =$ Reg$[i]$, then $V$ runs $\Pi'(x, t-1, \text{Reg}[j]; b)$ to get the address $m$, and then $V$ runs $\Pi'(x, t-1, \text{Mem}[m]; b)$ and rejects if this word doesn't equal $y$.
- If this instruction is "read Reg$[i] \leftarrow$ In[Reg$[j]$]" and $s =$ Reg$[i]$, then $V$ runs $\Pi'(x, t-1, \text{Reg}[j]; b)$ to get the address $m$, and then $V$ reads $x_m$ and rejects if $x_m \neq y$.

In all other cases, $V$ can determine which locations $s'$ are relevant to $s$ upon seeing which instruction would be executed. Now, we argue the correctness of $V$:

$\Rightarrow$: Suppose $A(x) = 1$ and thus $\Pi(x)$ accepts. Let $b = a_{N'}$ where $N'$ is the common size of all $(x, t, s)$. The words output by $\Pi'(x, t, s; b)$ for all $t, s$ are the transcript of $\Pi(x)$, which starts with the initial configuration and ends with an accept configuration, and each configuration (except the $0^{\text{th}}$) follows from the previous. Thus $V(x; b; t, s)$ accepts for all $t, s$.

$\Leftarrow$: Suppose $b$ is such that $V(x; b; t, s)$ accepts for all $t, s$. This means the words output by $\Pi'(x, t, s; b)$ for all $t, s$ form a sequence of configurations that starts with the initial configuration and ends with an accept configuration, and each configuration (except the $0^{\text{th}}$) follows from the previous. Thus this is an accepting transcript of $\Pi(x)$, so $A(x) = 1$.

$V$ is poly-time since $b$ and $t, s$ have poly$N$ size, and $\Pi'(x, t, s; b)$ and all $\Pi'(x, t-1, s'; b)$ run in poly$N' =$ poly$N$ time even when $b \neq a_{N'}$, and checking that $y$ follows from the relevant $\Pi'(x, t-1, s'; b)$ words (of which there are only a constant number) takes poly$N$ time.

**Exercise 5.8.a:** In every directed graph, the number of edges equals $\sum_u \text{outdeg}(u)$. Suppose for contradiction that in some $n$-node round-robin tournament, every node has outdegree $< (n-1)/2$. Then $\sum_u \text{outdeg}(u) < n \cdot (n-1)/2 = \binom{n}{2}$, which contradicts the fact that the graph has $\binom{n}{2}$ edges.

**Exercise 5.8.b:** Here's an algorithm that finds such an $S$ in such a graph $G$:

initialize $H \leftarrow G$ and $S \leftarrow \emptyset$
while $H$ is nonempty:
    find a node $u$ in $H$ with edges to at least half of $H$'s other nodes
    add $u$ to $S$
    remove $u$ and $u$'s outneighbors (and their edges) from $H$

Since $H$ is always a round-robin tournament, Exercise 5.8.a ensures that such a node $u$ always exists. In each iteration, over half of $H$'s nodes are removed, so there are $\leq \lfloor \log n \rfloor$ iterations and thus $|S| \leq \lfloor \log n \rfloor$. For every node $v$ in $G$, in the iteration when $v$ is removed from $H$, $v$ is either the node $u$ that's added to $S$, or an outneighbor of $u$.

**Exercise 5.8.c:** Assume $A$ SELECT has a poly-time program $\Pi$. For every input size $N$, define the graph $G_N$ as in the hint, and note that $G_N$ has $2^{O(N \log N)}$ nodes since every input of size $N$ has $O(N \log N)$ bits. By Exercise 5.8.b, there exists a set $S_N$ of $O(N \log N)$ nodes in $G_N$ such that for every node $y$ in $G_N$, either $y \in S_N$ or there exists a node $x \in S_N$ such that $(x, y)$ is an edge in $G_N$. Define advice $a_N$ that encodes the list of all elements of $S_N$. Here's a nonuniform program $\Pi'$ for $A$:

```
on input y of size N and advice a_N:
    for each x ∈ S_N:
        if x = y: accept
        if Π(min(x, y), max(x, y)) = y: accept
    reject
```

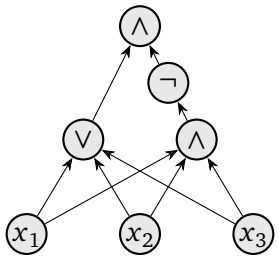
To show that $\Pi'$ is correct, we argue that $A(y) = 1$ iff $\Pi'(y; a_N)$ accepts:

$\Leftarrow$: Suppose $\Pi'(y; a_N)$ accepts in the $x$ iteration. Note that $A(x) = 1$ since $x$ is a node in $G_N$. Either $x = y$ in which case $A(y) = A(x) = 1$, or $\Pi(\min(x, y), \max(x, y)) = y$ in which case $A(y) = 1$ by correctness of $\Pi$.

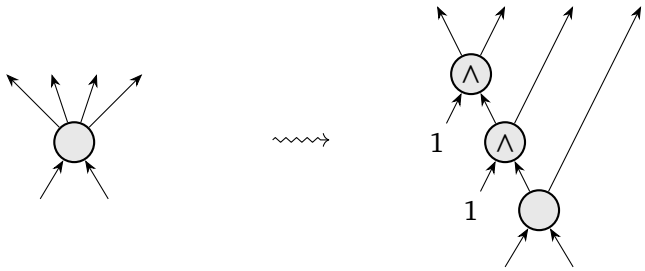
$\Rightarrow$: Suppose $A(y) = 1$, so $y$ is a node in $G_N$. Either $y \in S_N$ in which case $\Pi'(y; a_N)$ accepts in the $x = y$ iteration (if not sooner), or $(x, y)$ is an edge for some $x \in S_N$ in which case $\Pi'(y; a_N)$ accepts because $\Pi(\min(x, y), \max(x, y)) = y$ by definition of $G_N$.

$\Pi'$ is poly-time since $a_N$ and $S_N$ are poly-size and $\Pi$ is poly-time.

**Exercise 5.9:**



**Exercise 5.10:** Replace each gate of fan-out $> 2$ with a group of gates of fan-out 2 as shown below. To propagate whatever bit would be on the outgoing wires, use $\wedge$ gates each having an incoming constant 1 (or $\vee$ gates each having an incoming constant 0).




Instead of contracting the 1s away (which would just undo the modification), replace each 1 with a copy of $x_i \vee \neg x_i$ for some input node $x_i$. The size increases by only a constant factor.

**Exercise 5.11.a:** Start with the formula from the proof of Theorem 5.7, consisting of a tree of $2^N - 1$ many 2-to-1 muxes, where all $2^{N-i}$ muxes at level $i \in \{1, \ldots, N\}$ ($i = 1$ being the bottom, $i = N$ being the top) have $x_{i-1}$ as their selector, and with constants being selected by the level 1 muxes. For each $i$ and each function $f : \{0,1\}^i \to \{0,1\}$, if more than one level $i$ mux's subtree computes $f(x_0 \cdots x_{i-1})$, then delete all but one such mux. If a wire came from a now-deleted mux, then instead have it come from the equivalent surviving mux. This circuit is equivalent to the original formula, so it computes $F_N$. We just need to analyze the size, which is big-$O$ of the number of surviving muxes. (The $2^N$ many constant bits feeding into level 1 are actually just a single 0 and a single 1, so they don't contribute $2^N$ to the size.)

Let $k = \lceil \log(N/3) \rceil$, which is $\geq 1$ if $N$ is large enough. The number of surviving muxes at levels $\geq k$ is at most the number of original muxes at those levels, which is:

$$2^{N-k} + 2^{N-(k+1)} + \cdots + 2^0 \;=\; 2^{N-k+1} - 1 \;\leq\; 6 \cdot 2^N / N$$

The number of surviving muxes at level $i < k$ is at most $2^{2^i}$ since there are only that many functions $f : \{0,1\}^i \to \{0,1\}$. Thus the number of surviving muxes at levels $< k$ is at most

$$2^{2^1} + 2^{2^2} + \cdots + 2^{2^{k-1}} \;\leq\; 2^{2^k} \;\leq\; 2^{2^{\log(N/3)+1}} \;=\; 2^{2N/3} \;\leq\; 2^N / N$$

assuming $N$ is large enough. Thus the total number of surviving muxes is $O(2^N/N)$.

**Exercise 5.11.b:** This is essentially the same as Exercise 5.11.a: Each mux in the circuit corresponds to a node in the branching program, as in Theorem 5.22.

Start with the decision tree from the proof of Theorem 5.19, with $2^N - 1$ many internal nodes, where all $2^{N-i}$ nodes at level $i \in \{1, \ldots, N\}$ ($i = 1$ being the bottom, $i = N$ being the top) read $x_{N-i+1}$. For each $i$ and each function $f : \{0,1\}^i \to \{0,1\}$, if more than one level $i$ node's subtree computes $f(x_{N-i+1} \cdots x_N)$, then delete all but one such node. If an edge went to a now-deleted node, then instead have it go to the equivalent surviving node. This branching program is equivalent to the original decision tree, so it computes $F_N$. We just need to analyze the size, which is big-$O$ of the number of surviving internal nodes. (The $2^N$ many constant bits at the leaves are actually just a single 0 and a single 1, so they don't contribute $2^N$ to the size.)

Let $k = \lceil \log(N/3) \rceil$, which is $\geq 1$ if $N$ is large enough. The number of surviving nodes at levels $\geq k$ is at most the number of original nodes at those levels, which is:

$$2^{N-k} + 2^{N-(k+1)} + \cdots + 2^0 \;=\; 2^{N-k+1} - 1 \;\leq\; 6 \cdot 2^N / N$$

The number of surviving nodes at level $i < k$ is at most $2^{2^i}$ since there are only that many functions $f : \{0,1\}^i \to \{0,1\}$. Thus the number of surviving nodes at levels $< k$ is at most

$$2^{2^1} + 2^{2^2} + \cdots + 2^{2^{k-1}} \;\leq\; 2^{2^k} \;\leq\; 2^{2^{\log(N/3)+1}} \;=\; 2^{2N/3} \;\leq\; 2^N / N$$

assuming $N$ is large enough. Thus the total number of surviving nodes is $O(2^N/N)$.

**Exercise 5.12:** A formula of size $s$ on $N$ variables can be written as a string with $O(s \log N)$ bits: Each (, ), $\wedge$, $\vee$, and $\neg$ (of which there are $O(s)$ many) takes $O(1)$ bits, and each variable occurrence (of which there are $O(s)$ many) takes $O(\log N)$ bits to specify the variable's index. Thus, for some constant $c \geq 1$ and all large enough $N$, at most $2^{c \cdot s \log N}$ of the $2^{2^N}$ possible functions $F_N : \{0,1\}^N \to \{0,1\}$ have a formula of size $s$. If $s = \left\lceil \frac{1}{c} \cdot 2^N / \log N \right\rceil - 1$ then $2^{c \cdot s \log N} < 2^{2^N}$ so not every function $F_N$ can be computed by a formula of size $s$.

**Exercise 5.13.a:** To shrink the circuit for $x < y$ from §5.2.5, the "$x_{W-1} \cdots x_{i+1} = y_{W-1} \cdots y_{i+1}$" subcircuits can share a lot of their work. For each $i \in \{1, \ldots, W-1\}$, Have a $O(1)$-size subcircuit with "output gate" $g_i$ that computes whether $x_i = y_i$. Combine them with this formula:

$$(\cdots(((g_{W-1} \wedge g_{W-2}) \wedge g_{W-3}) \wedge g_{W-4}) \cdots \wedge g_1)$$

For each $i \in \{0, \ldots, W-2\}$, "$x_{W-1} \cdots x_{i+1} = y_{W-1} \cdots y_{i+1}$" is available as an $\wedge$ gate in that formula, or as $g_{W-1}$ if $i = W-2$. The output is an $\vee$ gate of fan-in $W$ whose $i^{\text{th}}$ subcircuit is $\overline{x_i} \wedge y_i \wedge$ "$x_{W-1} \cdots x_{i+1} = y_{W-1} \cdots y_{i+1}$", which has size $O(1)$ not counting the "$x_{W-1} \cdots x_{i+1} = y_{W-1} \cdots y_{i+1}$" which was already computed. Thus the circuit's overall size is $O(W)$. This circuit is not a formula (unlike the circuit from §5.2.5) since the $g_i$ gates have fan-out $> 1$.

**Exercise 5.13.b:** To shrink the circuit for $\lfloor \log x \rfloor$ from §5.2.5, the "$z_i = \overline{x_{W-1}} \wedge \cdots \wedge \overline{x_{i+1}} \wedge x_i$" subcircuits can share a lot of their work. We have this formula:

$$(\cdots(((\overline{x_{W-1}} \wedge \overline{x_{W-2}}) \wedge \overline{x_{W-3}}) \wedge \overline{x_{W-4}}) \cdots \wedge \overline{x_1})$$

Then each $z_i$ with $i < W-2$ is the $\wedge$ of $x_i$ and a gate from this formula, and we have $z_{W-1} = x_{W-1}$ and $z_{W-2} = \overline{x_{W-1}} \wedge x_{W-2}$. Thus it takes $O(W)$ size to compute all $z_i$. It takes $O(W \log W)$ size to compute the output from all $z_i$, since there are $O(\log W)$ output bits (besides the constant 0s), each of which is an $\vee$ gate with fan-in $\leq W$.

Alternative construction: For each $i \leftarrow 0, \ldots, W-1$, have a 2-bundles-to-1-bundle mux that uses $x_i$ to select either $i$ (a constant binary number) if $x_i = 1$, or the previous mux's output if $x_i = 0$. To make log-of-0 output 0, use $0 \cdots 0$ in lieu of the "previous mux" when $i = 0$. The total size is $O(W \log W)$ since each bundle only needs $O(\log W)$ wires to carry a value of $i$.

**Exercise 5.14.a:** In the following, all circuits have $N$ input nodes and one output node, and we use the notation $<$ for lexicographic comparison of circuits.

Idea: On input $x \in \{0,1\}^N$, we have $\exists B$ where $B$ is a circuit of size $\leq N^{d+1}$ purported to be $C_N$. We check that $B(x) = 1$, and we use additional quantifiers to check that $B = C_N$. To check that $B \in S_N$ (no circuit of size $\leq N^d$ is equivalent to $B$):

$$\left(\forall \text{ circuit } D \text{ of size } \leq N^d\right)\left(\exists y \in \{0,1\}^N\right) :\ B(y) \neq D(y)$$

To check that $S_N$ contains no circuit that's $< B$:

$$\left(\forall \text{ circuit } E < B \text{ of size } \leq N^{d+1}\right)\left(\exists \text{ circuit } F \text{ of size } \leq N^d\right)\left(\forall z \in \{0,1\}^N\right) :\ E(z) = F(z)$$

Since these two checks are independent of each other, we can combine their quantifiers.

Formally, we define a 4-witness verifier $V$ such that:

$$C_N(x) = 1 \iff \left(\exists B\ \ \forall D, E\ \ \exists y, F\ \ \forall z :\ V(x; B; D, E; y, F; z) \text{ accepts}\right)$$

$V(x; B; D, E; y, F; z)$:
  if $B$ isn't a circuit of size $\leq N^{d+1}$: reject
  if $D$ isn't a circuit of size $\leq N^d$ or $E$ isn't a circuit of size $\leq N^{d+1}$ or $E \geq B$: accept
  if $y \notin \{0,1\}^N$ or $F$ isn't a circuit of size $\leq N^d$: reject
  if $z \notin \{0,1\}^N$: accept
  if $B(x) = 1$ and $B(y) \neq D(y)$ and $E(z) = F(z)$: accept
  reject

$\Rightarrow$: Suppose $C_N(x) = 1$. Let $B = C_N$, and consider any $D$ of size $\leq N^d$ and any $E < B$ of size $\leq N^{d+1}$. Since $B \in S_N$, $B$ and $D$ are not equivalent and thus there exists $y$ such that $B(y) \neq D(y)$. Since $E \notin S_N$ (because $E < B$ and $B$ is the lexicographically first circuit in $S_N$), there exists $F$ of size $\leq N^d$ that's equivalent to $E$, which means for all $z$, $E(z) = F(z)$. Thus $V$ accepts.

$\Leftarrow$: Suppose $\exists B\ \ \forall D, E\ \ \exists y, F\ \ \forall z :\ V(x; B; D, E; y, F; z)$ accepts. We claim that $B = C_N$, in which case $C_N(x) = B(x) = 1$ since $V$ accepts. We have $B \in S_N$ since $B$ has size $\leq N^{d+1}$ and for every $D$ of size $\leq N^d$, there exists $y$ such that $B(y) \neq D(y)$ and thus $B$ and $D$ are not equivalent. Also, $B$ is the lexicographically first circuit in $S_N$ since for every $E < B$ of size $\leq N^{d+1}$, there exists $F$ of size $\leq N^d$ such that for all $z$, $E(z) = F(z)$, which means $E$ and $F$ are equivalent and thus $E \notin S_N$.

The pseudocode's last "if" statements use a poly-time program for Circuit Evaluation (Lemma 5.10). The encodings of $B, D, E, y, F, z$ are poly-size. Thus $V$ is poly-time.

**Exercise 5.14.b:** Consider two cases: If $\mathsf{NP} \subseteq \mathsf{P/poly}$ then $\Sigma_4\mathsf{P} \subseteq \Sigma_2\mathsf{P}$ (Theorem 5.5) and thus by Exercise 5.14.a there exists a problem in $\Sigma_4\mathsf{P} \subseteq \Sigma_2\mathsf{P}$ whose circuit size complexity is $> N^d$ for all large enough $N$, and thus not $\leq N^d$. If $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ then there exists a problem in $\mathsf{NP} \subseteq \Sigma_2\mathsf{P}$ whose circuit size complexity is not poly$N$ (for any degree) and thus not $\leq N^d$.

**Exercise 5.15:** By Theorem 5.7, there exists a constant $c$ such that every $f : \{0,1\}^N \to \{0,1\}$ has a circuit of size $\leq c2^N$.

| Max Circuit Complexity Decision | |
|---|---|
| Input: | Positive integer $s \leq c2^N$, with padding so the input size is $M = 2^N$ |
| Output: | Does every $f : \{0,1\}^N \to \{0,1\}$ have a circuit of size $\leq s$? |

Max Circuit Complexity Decision $\in \Pi_2\mathsf{P}$ by a 2-witness verifier $V_1$ where $V_1(s; f; C)$ accepts iff $C$ is a circuit of size $\leq s$ and $f(x) = C(x)$ for all $x \in \{0,1\}^N$. This $V_1$ has time efficiency $O(M^2)$ since $f$ has size $2^N = M$, $C$ has size $\leq c2^N = cM$, the loop over all $x$ has $2^N = M$ iterations, and evaluating $C(x)$ takes time $O(s) = O(M)$ (Lemma 5.10).

| High Complexity Truth Table Completion | |
|---|---|
| Input: | Partial truth table $f : \{0,1\}^N \to \{0,1,*\}$, positive integer $s \leq c2^N$ |
| Output: | Does there exist a total truth table $g : \{0,1\}^N \to \{0,1\}$ consistent with $f$ such that every circuit for $g$ has size $\geq s$? |

High Complexity Truth Table Completion $\in \Sigma_2\mathsf{P}$ by a 2-witness verifier $V_2$ where $V_2(f, s; g; C)$ accepts iff $g$ is consistent with $f$ and either $C$ isn't a circuit of size $< s$ or $g(x) \neq C(x)$ for at least one $x \in \{0,1\}^N$. Letting $M$ denote the input size, this $V_2$ has time efficiency $O(M^2)$ since $g$ has size $2^N \leq M$, $C$ has size $< c2^N \leq cM$, the loop over all $x$ has $2^N \leq M$ iterations, and evaluating $C(x)$ takes time $O(s) = O(M)$ (Lemma 5.10).

Assuming $\mathsf{P} = \mathsf{NP}$, we have $\mathsf{P} = \Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$ (Theorem 4.21). Thus, Max Circuit Complexity Decision $\in \mathsf{P}$ by a poly-time program $\Pi_1$, and High Complexity Truth Table Completion $\in \mathsf{P}$ by a poly-time program $\Pi_2$.

Define $F : \{0,1\}^+ \to \{0,1\}$ as follows: Let $s_N$ be the maximum over all $f : \{0,1\}^N \to \{0,1\}$ of the size of a smallest circuit for $f$, and let $F_N : \{0,1\}^N \to \{0,1\}$ be the lexicographically first function such that a smallest circuit for $F_N$ has size $s_N$. Here, the lexicographic order is by viewing a truth table as a bit string of length $2^N$ indexed by inputs $x \in \{0,1\}^N$ (ordered by their values as binary numbers in $\{0,1,\ldots,2^N-1\}$).

By definition, $F$ has the maximum possible circuit size complexity for all $N$. We prove $F \in \mathsf{E}$ by describing an algorithm to compute $F_N(x)$:

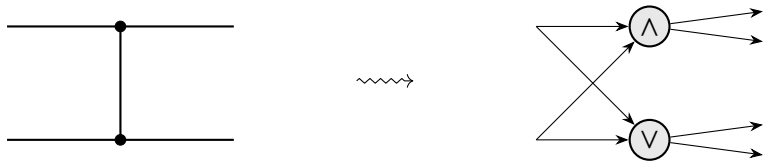
```
find s_N by doing binary search over s ≤ c2^N to find the least s such that Π_1(s) accepts
initialize f : {0,1}^N → {0,1,*} by f(y) ← * for all y
for all y ∈ {0,1}^N in increasing numerical order:
    update f(y) ← 0
    run Π_2(f, s_N)
    if it accepted: keep f(y) ← 0
    if it rejected: update f(y) ← 1
output f(x)
```

The loop over $y$ finds $F_N$'s truth table bit by bit, maintaining the invariant that $F_N$ is consistent with the partial truth table $f$: The invariant trivially holds before the loop. We argue that each

iteration maintains the invariant: If $F_N(y) = 0$ then $\Pi_2(f, s_N)$ accepts for the version of $f$ with $f(y) = 0$ (since $F_N$ is such a $g$) and so the algorithm fixes $f(y) = 0$. If $F_N(y) = 1$ then $\Pi_2(f, s_N)$ rejects for the version of $f$ with $f(y) = 0$ and so the algorithm fixes $f(y) = 1$—this is because if there existed a total $g$ consistent with $f$ with $g(y) = 0$ and whose smallest circuit has size $s_N$, then $g$ would come lexicographically before $F_N$, contradicting the definition of $F_N$. Upon finishing the loop over all $y$, we must have $f = F_N$ since $f$ is total and $F_N$ is consistent with $f$. Thus the algorithm correctly outputs $f(x) = F_N(x)$.

For finding $s_N$, each iteration takes time $\text{poly}(2^N) = 2^{O(N)}$, and there are $O(N)$ iterations (though it would also be fine to try all $c2^N$ possibilities of $s$ rather than do binary search), so this phase takes time $O(N) \cdot 2^{O(N)} = 2^{O(N)}$. For finding $F_N$, each iteration takes time $\text{poly}(2^N) = 2^{O(N)}$, and there are $2^N$ iterations, so this phase takes time $2^N \cdot 2^{O(N)} = 2^{O(N)}$.

**Exercise 5.16:** For all $x \in \{0,1\}^N$, $\textsc{Majority}(x) = y_{\lceil N/2 \rceil}$ where $y = y_1 \cdots y_N$ is the sorted version of $x$ (all 0s precede all 1s in $y$). We turn the $O(N \log^2 N)$-size sorting network from Theorem 2.34 into a $O(N \log^2 N)$-size monotone circuit where each wire in the sorting network becomes a pair of wires in the circuit, and each comparator becomes a pair of gates:




The sorting network's output wires carry $y_1 \cdots y_N$. We delete the corresponding "wires" from the circuit (since they have no gates to go to), and we designate the gate corresponding to $y_{\lceil N/2 \rceil}$ as the output.

**Exercise 5.17.a:** Call this problem $A$. We have $A \in \mathsf{P}$ since Circuit Evaluation $\in \mathsf{P}$. It suffices to show Circuit Evaluation $\leq_\ell^m A$ since Circuit Evaluation is P-complete (Theorem 5.15). Map $(C, a)$ to $(C', a')$ as follows. $C'$ has one input node, which $a'$ assigns 0, and which feeds into a $\neg$ gate. Other than that, $C'$ has a copy of $C$ except for $C$'s input nodes. For each wire from an input node in $C$, if $a$ assigns that node 0 then $C'$ has the wire come from its only input node, and if $a$ assigns that node 1 then $C'$ has the wire come from the $\neg$ gate attached to the input node. This log-space mapping reduction is correct since $C(a) = C'(a')$.

**Exercise 5.17.b:** Call this problem $A$. Note that $C(a) = 0$ iff the output node is reachable from an input node that $a$ assigns 0. We have $\overline{A} \in \mathsf{NL} = \mathsf{coNL}$ (Theorem 3.19) since $\overline{A} \leq_\ell^m$ Directed Reachability $\in \mathsf{NL}$ (Lemma 3.20 and Lemma 3.10) by mapping $(C, a)$ to $(G, s, t)$ where $G$ is $C$ with a new node $s$ having edges to all input nodes that $a$ assigns 0, and where $t$ is the output node. This log-space mapping reduction is correct since $C(a) = 0$ iff $G$ has a path from $s$ to $t$.

Next, it suffices to show Directed Reachability on a Dag $\leq_\ell^m \overline{A}$ since Directed Reachability on a Dag is NL-complete (Exercise 3.13.b). Map $(G, s, t)$ to $(C, a)$ where $C$ is $G$ except we make $s$ an input node (by deleting any incoming edges) that $a$ assigns 0, and all other sources are input nodes that $a$ assigns 1, and all other nodes are $\wedge$ gates (bypassing any dummy gates that have fan-in 1), and $t$ is the output node. $C$ is a dag since $G$ is a dag. This log-space mapping reduction is correct since $G$ has a path from $s$ to $t$ iff $C(a) = 0$.

**Exercise 5.18:** We show Geography on a Dag with Targets $\in$ P by modifying the algorithm for Geography on a Dag from Theorem 5.17. We have arrays *P1wins* and *P2wins*, where *P1wins*$[i]$ gets the bit indicating whether P1 has a winning strategy for the rest of the game, assuming it's P1's turn and they're at node $v_i$, and similarly for *P2wins*$[i]$.

topologically order $G$'s nodes $v_1, v_2, \ldots, v_n$
for $i \leftarrow n, n-1, \ldots, k$ where $v_k = s$:
    if $v_i$ is "P1 loses": *P1wins*$[i] \leftarrow 0$ and *P2wins*$[i] \leftarrow 1$
    if $v_i$ is "P2 loses": *P1wins*$[i] \leftarrow 1$ and *P2wins*$[i] \leftarrow 0$
    else if $v_i$ has edges to $v_{j_1}, \ldots, v_{j_d}$:
        *P1wins*$[i] \leftarrow \neg$*P2wins*$[j_1] \vee \cdots \vee \neg$*P2wins*$[j_d]$
        *P2wins*$[i] \leftarrow \neg$*P1wins*$[j_1] \vee \cdots \vee \neg$*P1wins*$[j_d]$
output *P1wins*$[k]$

This algorithm runs in linear time.

It suffices to show Monotone Circuit Evaluation $\leq_\ell^m$ Geography on a Dag with Targets since Monotone Circuit Evaluation is P-complete (Theorem 5.16). We use almost the same reduction we used to show Monotone Circuit Evaluation $\leq_\ell^m$ Geography on a Dag. In the latter reduction, for every sink $v$ in $G$, either every path from $s$ to $v$ has even length (if $v$ is a 0 node) in which case P1 loses if they reach $v$, or every path from $s$ to $v$ has odd length (if $v$ is an $\wedge$ node) in which case P2 loses if they reach $v$. Thus we can modify $G$ by combining all sinks at even distance from $s$ into a single "P1 loses" sink (which has all those sinks' incoming edges), and combining all sinks at odd distance from $s$ into a single "P2 loses" sink.

Or, we could show Geography on a Dag $\leq_\ell^m$ Geography on a Dag with Targets by mapping $(G, s)$ to $(G', s')$ as follows. First, define a graph $H$ with two copies of $G$'s nodes: a left copy and a right copy. Each edge $(u, v)$ of $G$ becomes two edges in $H$: an edge from the left $u$ to the right $v$, and an edge from the right $u$ to the left $v$. Node $s'$ is the left $s$. Gameplay in $(H, s')$ proceeds as in $(G, s)$, but the left/right sides keep track of whose turn it is (left = P1's turn, right = P2's turn). Obtain $G'$ from $H$ by combining all left sinks into a single "P1 loses" sink, and combining all right sinks into a single "P2 loses" sink. This log-space mapping reduction is correct since P1 has a winning strategy in the path forming game on $(G, s)$ iff P1 has a winning strategy in the "path forming game with targets" on $(G', s')$.

**Exercise 5.19:** Say that an input node is *unsafe* if it feeds directly into at least one ¬ gate, and *safe* otherwise.

if $C$ has a ¬ gate whose incoming wire comes from a gate: reject
if $C$ has an ∧ gate with at least one incoming wire from an unsafe input node: reject
if $C$ has an ∨ gate with all its incoming wires from unsafe input nodes: reject
accept

To show this log-space algorithm is correct, we argue that it accepts iff there exists an assignment that makes every gate of $C$ evaluate to 1:

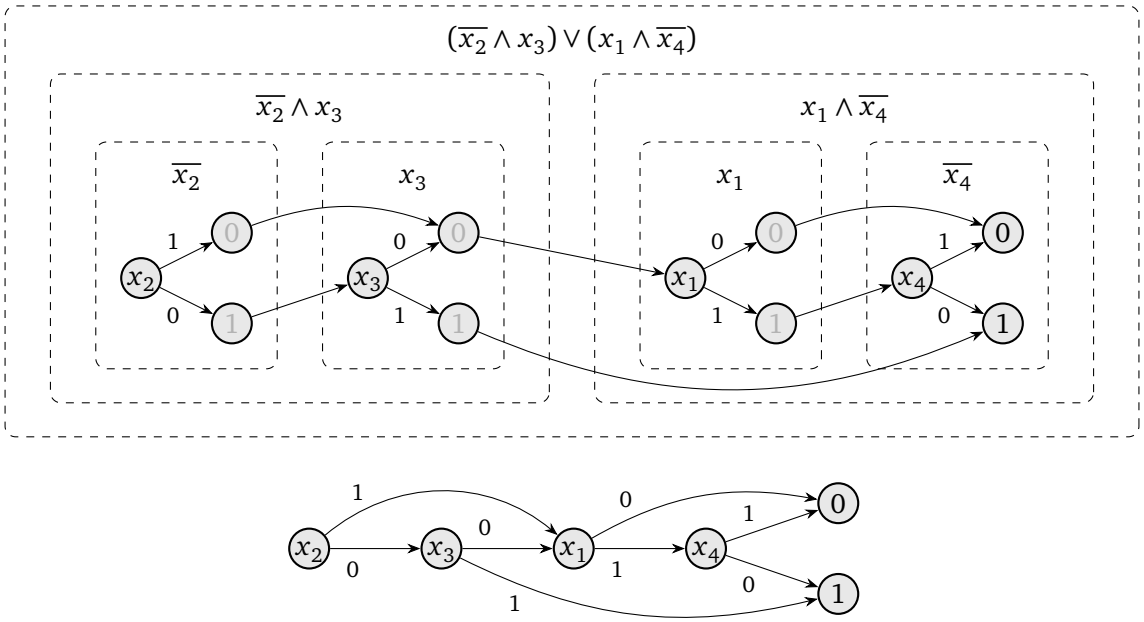
⇒: Suppose the algorithm accepts. Assign 1 to each safe input node and 0 to each unsafe input node. Each ¬ gate evaluates to 1 since its incoming wire comes from an unsafe input node. Each ∧ gate evaluates to 1 since all its incoming wires come from safe input nodes and/or other gates, which evaluate to 1. Each ∨ gate evaluates to 1 since at least one of its incoming wires comes from a safe input node or another gate, which evaluates to 1.

⇐: Suppose the algorithm rejects. We show that for every assignment, some gate evaluates to 0. If $C$ has a ¬ gate whose incoming wire comes from a gate, then for every assignment, either the ¬ gate or its predecessor gate evaluates to 0. Suppose $C$ has an ∧ gate with at least one incoming wire from an unsafe input node. If that input node is assigned 0, then the ∧ gate evaluates to 0. If that input node is assigned 1, then the ¬ gate it feeds into evaluates to 0. Suppose $C$ has an ∨ gate with all its incoming wires from unsafe input nodes. If all those input nodes are assigned 0, then the ∨ gate evaluates to 0. If at least one of those input nodes is assigned 1, then the ¬ gate it feeds into evaluates to 0.

**Exercise 5.20.a:**
$(((((0 \wedge \overline{x_3}) \vee (1 \wedge x_3)) \wedge \overline{x_1}) \vee (0 \wedge x_1)) \wedge \overline{x_2}) \vee (((1 \wedge \overline{x_3}) \vee (((1 \wedge \overline{x_1}) \vee (0 \wedge x_1)) \wedge x_3)) \wedge x_2)$

After contracting: $((x_3 \wedge \overline{x_1}) \wedge \overline{x_2}) \vee ((\overline{x_3} \vee (\overline{x_1} \wedge x_3)) \wedge x_2)$

**Exercise 5.20.b:**



**Exercise 5.21:** Branching Program Sat is NP-complete via $\leq_\ell^m$ and thus not in NL unless NP = NL: Branching Program Sat $\in$ NP by a verifier that, given input $B$ and an assignment $a$ as a purported witness, runs a log-space (and poly-time) program for Branching Program Evaluation (Lemma 5.23) on input $(B, a)$ to check whether $a$ satisfies $B$. Branching Program Sat is NP-hard via $\leq_\ell^m$ because Sat is NP-hard via $\leq_\ell^m$ (Theorem 2.40) and Sat $\leq_\ell^m$ Branching Program Sat by mapping a CNF formula to an equivalent branching program as in Theorem 5.22.*(ii)*, which only needs log space.

Branching Program Sat would be NL-complete if we defined NL by allowing random access to the witness segment and didn't count the witness segment toward the space (treating the witness segment like the input segment), but we saw in §3.3.1 that NL would equal NP in that case.

**Exercise 5.22.a:**

**Lemma.** Directed Switching Network Evaluation $\in$ NL.

**Proof.** We show Directed Switching Network Evaluation $\leq_\ell^m$ Directed Reachability and use Directed Reachability $\in$ NL (Lemma 3.10 and Lemma 3.20). Map $((G,s,t),a)$ to $(G_a,s,t)$ where $G_a$ is $G$ but without edges labeled by literals that evaluate to 0 under the assignment $a$. This log-space mapping reduction is correct because (there exists a path from $s$ to $t$ in $G$ only using edges that are unlabeled or labeled by literals that evaluate to 1 under $a$) iff (there exists a path from $s$ to $t$ in $G_a$). ■

**Theorem.** *Every problem in* NSPACE$[S]$ *has directed switching network size complexity at most* $2^{O(S)}$.

**Proof.** Consider any $A \in \text{NSPACE}[S]$. We design a family of directed switching networks of size $2^{O(S)}$ solving $A$. Let $V$ be a streaming verifier for $A$ with $L$ lines, $R$ registers, space efficiency $S' \leq O(S)$, and word size $W \leq O(\log \max(S,N))$. Consider any valid input $x$ of size $N$. Define $M = S'/W - R$ so $V(x;w)$ never accesses any work segment address $\geq M$. A configuration in $\{0,1\}^{\log L} \times (\{0,1\}^W)^R \times (\{0,1\}^W)^M$ represents the program counter, all $R$ registers, and the work segment's first $M$ words. Let $K = L2^{S'} \leq 2^{O(S)}$ be the number of configurations. For all $w$, $V(x;w)$ takes fewer than $K$ steps (Lemma 3.8). When $V(x;w)$ terminates, imagine it continues taking steps but with the configuration frozen. Assume $V$ has only one possible accept configuration.

We describe a directed switching network for $A$ on inputs of size $N$. Letting $W'$ be $A$'s word size, there are $n = NW'$ variables for the bits of the input $x$. There are $K^2$ nodes in $G$, corresponding to pairs $(C,k)$ where $C$ is a configuration and $k \in \{0,1,\ldots,K-1\}$ is a time step. Let $s =$ (initial configuration, 0) and $t =$ (accept configuration, $K-1$), and all other nodes with $k = 0$ or $k = K-1$ are unnecessary. On a valid input $x$, (there will exist a path from $s$ to $(C,k)$ in $G$ using only edges consistent with $x$) iff (there exists $w$ such that $V(x;w)$ is in configuration $C$ after step $k$). This directed switching network will be correct since it outputs 1 iff (there exists a path from $s$ to $t$ in $G$ using only edges consistent with $x$) iff (there exists $w$ such that $V(x;w)$ accepts) iff $A(x) = 1$. To define the outgoing edges of node $(C,k)$ when $k < K-1$, consider $V$'s instruction at line $C_{\text{PC}}$:

- Data processing, control flow, or "load"/"store": $(C,k)$ has an unlabeled edge to $(C',k+1)$ where $C'$ is the configuration after one step of $V$ from $C$. Caveat: $C$ might load or store at an address $\geq M$, in which case $(C,k)$ won't be visited on any valid input, but for simplicity we keep the node and give it no outgoing edges.
- "read Reg$[i]$ $\leftarrow$ In[Reg$[j]$]": Let $m = C_{\text{Reg}[j]}$ and first assume $m < N$. The node $(C,k)$ has $2^{W'}$ outgoing edges labeled by the terms "$x_m = a_m$" for all possible words $a_m \in \{0,1\}^{W'}$. The $a_m$ edge goes to $(C',k+1)$ where $C'$ is $C$ except $C'_{\text{Reg}[i]} = a_m$ (left padded with 0s if $W' < W$) and $C'_{\text{PC}} = C_{\text{PC}} + 1$. These edges can be turned into a binary tree of depth $W'$ whose edges are labeled with individual literals. If $m \geq N$ (reading past the input) then $(C,k)$ has an unlabeled edge to $(C',k+1)$ where $C'$ is $C$ except $C'_{\text{Reg}[i]} = 0$ and $C'_{\text{PC}} = C_{\text{PC}} + 1$.
- "read-witness Reg$[i]$": $(C,k)$ has $2^W$ unlabeled edges to the nodes $(C',k+1)$ where $C'$ is $C$ except $C'_{\text{Reg}[i]}$ is anything and $C'_{\text{PC}} = C_{\text{PC}} + 1$.

- "accept" or "reject": $(C, k)$ has an unlabeled edge to $(C, k+1)$.

$G$ is a dag since each edge goes from a layer $k$ to layer $k+1$—the layers provide a topological order. The total number of nodes and edges is $O(2^W \cdot K^2) \leq 2^{O(S)} \cdot (2^{O(S)})^2 \leq 2^{O(S)}$. ■

Finally, we prove that for every decision problem $A$, we have $A \in \mathsf{NL/poly}$ iff $A$ has a poly-size directed switching network family.

$\Leftarrow$: Suppose a poly-size directed switching network family $(G_1, s_1, t_1), (G_2, s_2, t_2), \ldots$ solves $A$. For each $N$, let the advice $a_N$ be the encoding of $(G_N, s_N, t_N)$. (Now, $a$ stands for "advice," not "assignment.") Define a streaming verifier $V'$ such that on input $x$ of size $N$ and purported witness $w$, $V'(x; w; a_N)$ runs a log-space streaming verifier $V$ for Directed Switching Network Evaluation on input $((G_N, s_N, t_N), x)$ and purported witness $w$, and outputs the same bit. This is correct since the directed switching network is correct:

$$\begin{aligned} \big(\exists w : \ V'(x; w; a_N) \text{ accepts}\big) &\iff \big(\exists w : \ V((G_N, s_N, t_N), x; w) \text{ accepts}\big) \\ &\iff (G_N, s_N, t_N) \text{ outputs 1 on input } x \\ &\iff A(x) = 1 \end{aligned}$$

The space efficiency of $V'$ is logarithmic in the size of $((G_N, s_N, t_N), x)$, which is $\operatorname{poly} N$. Thus $A \in \mathsf{NL/poly}$.

$\Rightarrow$: Suppose $A \in \mathsf{NL/poly}$ by a log-space streaming verifier $V$ with advice $a_1, a_2, \ldots$. Let $B$ (with input size $M$) be the promise problem $A$ with $V$'s Advice: The input is $(x, a_N)$ and the output is $A(x)$.

$B \in \mathsf{NL}$ by a streaming verifier $V'$ where $V'(x, a_N; w)$ runs $V(x; w; a_N)$. This is correct for $B$:

$$\big(\exists w : \ V'(x, a_N; w) \text{ accepts}\big) \iff \big(\exists w : \ V(x; w; a_N) \text{ accepts}\big) \iff A(x) = 1 \iff B(x, a_N) = 1$$

The space efficiency of $V'$ is $O(\log N) = O(\log M)$.

Thus a poly-size directed switching network family $(G_1, s_1, t_1), (G_2, s_2, t_2), \ldots$ solves $B$. We turn this into a poly-size directed switching network family $(G'_1, s'_1, t'_1), (G'_2, s'_2, t'_2), \ldots$ that solves $A$. For each $N$, let $M$ be the size of inputs to $B$ of the form $(x, a_N)$, and obtain $(G'_N, s'_N, t'_N)$ from $(G_M, s_M, t_M)$ by plugging in the bits of $N$ and $a_N$ as constants (edges labeled with literals that evaluate to 0 are removed, and edges labeled with literals that evaluate to 1 become unlabeled), leaving only $x$ for the input variables. This solves $A$ since the output of $(G'_N, s'_N, t'_N)$ on input $x$ equals the output of $(G_M, s_M, t_M)$ on input $(x, a_N)$, which equals $B(x, a_N) = A(x)$. The size of $(G_M, s_M, t_M)$, and thus the size of $(G'_N, s'_N, t'_N)$, is $\operatorname{poly} M = \operatorname{poly} N$.

**Exercise 5.22.b:**

**Lemma.** Skew Circuit Evaluation $\in$ NL.

**Proof.** Here's a log-space streaming verifier $V$ for Skew Circuit Evaluation on input $(C, a)$ and purported witness $w$. A "1 literal" is a literal that evaluates to 1 under $a$, and similarly for a "0 literal."

```
initialize v ← C's output node
repeat:
    if v is a 1 literal: accept
    if v is a 0 literal: reject
    if v is an ∧ gate for which all inneighbors are 1 literals: accept
    if v is an ∧ gate for which at least one inneighbor is a 0 literal: reject
    if v is an ∧ gate:
        update v ← v's unique inneighbor that isn't a literal
    else if v is an ∨ gate:
        u ← w's next word
        if u isn't an inneighbor of v: reject
        update v ← u
```

In the fifth case in the loop, $v$ indeed has exactly one inneighbor that isn't a literal, because $C$ is a skew circuit.

By design, $V$ always terminates, and $V$ uses log space since it just remembers $v$ and $u$ and some pointers into the input. To show that $V$ is correct, we argue that $C(a) = 1 \iff \big(\exists w : V(C, a; w) \text{ accepts}\big)$:

$\Rightarrow$: Assume $C(a) = 1$. We claim that some $w$ maintains the invariant that $v$ evaluates to 1 on input $a$: This holds initially since $C(a) = 1$. The fifth case in the loop maintains the invariant because if $v$ evaluates to 1 then each of its inneighbors evaluates to 1. The sixth case in the loop maintains the invariant because if $v$ evaluates to 1 then at least one of its inneighbors evaluates to 1, and $u$ can be such an inneighbor. Since the invariant holds when the loop terminates with one of the first four cases, it must be the first or third case (since in the second and fourth cases, $v$ evaluates to 0), so $V$ accepts.

$\Leftarrow$: Assume $C(a) = 0$. We claim that every $w$ maintains the invariant that $v$ evaluates to 0 on input $a$: This holds initially since $C(a) = 0$. The fifth case in the loop maintains the invariant because all of $v$'s literal inneighbors evaluate to 1 (otherwise $V$ would have rejected in the fourth case) and so if $v$ evaluates to 0 then so does its other inneighbor. The sixth case in the loop maintains the invariant because if $v$ evaluates to 0 then so do all its inneighbors. Since the invariant holds when the loop terminates with one of the first four cases, it must be the second or fourth case (since in the first and third cases, $v$ evaluates to 1), so $V$ rejects.

Another perspective on this proof that Skew Circuit Evaluation $\in$ NL: We show Skew Circuit Evaluation $\leq_\ell^m$ Directed Reachability $\in$ NL (Lemma 3.10 and Lemma 3.20) by mapping $(C, a)$ to $(G, s, t)$ as follows:

- For each $\wedge$ gate for which all inneighbors are 1 literals, we replace the $\wedge$ gate with constant 1.
- For each $\wedge$ gate for which at least one inneighbor is a 0 literal, we replace the $\wedge$ gate with constant 0.
- For each other $\wedge$ gate, we remove all the incoming wires from literals, so the $\wedge$ gate only retains its incoming wire from a non-literal. An "$\wedge$ gate with fan-in 1" is equivalent to an "$\vee$ gate with fan-in 1," so now the circuit only has $\vee$ gates.

We define $G$ to be this modified circuit with a new node $s$ having edges to all constant 1s (including 1s we just created and any remaining literals that evaluate to 1 under $a$), and where $t$ is the output node. This log-space mapping reduction is correct since $C(a) = 1$ iff the modified circuit outputs 1 iff $G$ has a path from $s$ to $t$. ■

**Theorem.** *Every directed switching network is equivalent to some skew circuit that's only a constant factor bigger.*

**Proof.** We turn any directed switching network $(G, s, t)$ into a skew circuit $C$:

- For every node $v$ in $G$, $C$ has a node $g_v$ that will evaluate to 1 on input $x$ iff $v$ is reachable from $s$ using only edges consistent with $x$.
- For every edge $e = (u, v)$ in $G$, $C$ has a node $g_e$ that will evaluate to 1 on input $x$ iff $e$ is consistent with $x$ and $u$ is reachable from $s$ using only edges consistent with $x$.

To achieve this:

- $g_s$ is constant 1.
- For every node $v \neq s$ in $G$, $g_v = \bigvee_{\text{edges } e \text{ into } v} g_e$.
- For every unlabeled edge $e = (u, v)$ in $G$, $g_e$ is an alias for $g_u$.
- For every edge $e = (u, v)$ labeled $x_i$ in $G$, $g_e = x_i \wedge g_u$.
- For every edge $e = (u, v)$ labeled $\overline{x_i}$ in $G$, $g_e = \overline{x_i} \wedge g_u$.

$C$'s output node is $g_t$, and so $C$ is equivalent to $(G, s, t)$. Note that $C$ is a skew circuit that's only a constant factor bigger than $(G, s, t)$. ■

Finally, we prove that for every decision problem $A$, we have $A \in \mathsf{NL/poly}$ iff $A$ has a poly-size skew circuit family.

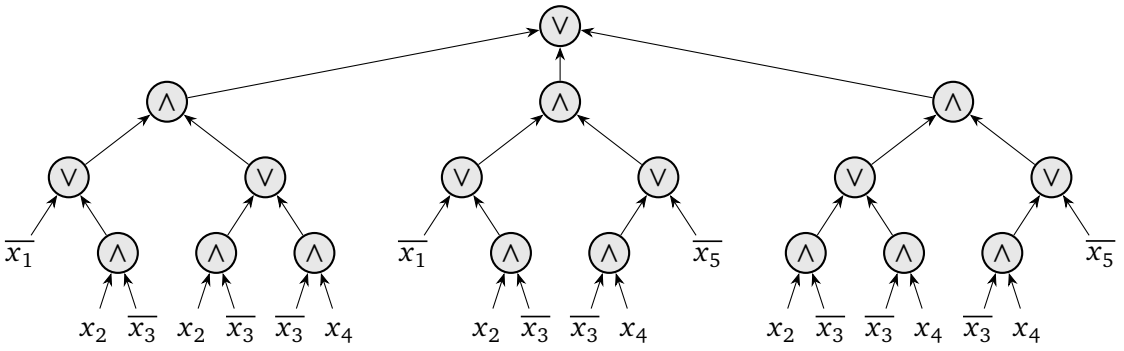
$\Leftarrow$: Suppose a poly-size skew circuit family $C_1, C_2, \ldots$ solves $A$. For each $N$, let the advice $a_N$ be the encoding of $C_N$. (Now, $a$ stands for "advice," not "assignment.") Define a streaming verifier $V'$ such that on input $x$ of size $N$ and purported witness $w$, $V'(x; w; a_N)$ runs a log-space streaming verifier $V$ for Skew Circuit Evaluation on input $(C_N, x)$ and purported witness $w$, and outputs the same bit. This is correct since the skew circuit is correct:

$$\big(\exists w:\ V'(x; w; a_N) \text{ accepts}\big) \iff \big(\exists w:\ V(C_N, x; w) \text{ accepts}\big) \iff C_N(x) = 1 \iff A(x) = 1$$

The space efficiency of $V'$ is logarithmic in the size of $(C_N, x)$, which is poly$N$. Thus $A \in \mathsf{NL/poly}$.

$\Rightarrow$: If $A \in \mathsf{NL/poly}$ then $A$ has a poly-size directed switching network family (Exercise 5.22.a), so $A$ has a poly-size skew circuit family as we proved above.

**Exercise 6.1:**



**Exercise 6.2:** We prove this for NC. The same argument works for AC since the fan-in is irrelevant.

Assume $\mathsf{NC}^{i+1} \subseteq \mathsf{NC}^i$. We argue that for all $\ell > i$, $\mathsf{NC}^{\ell+1} \subseteq \mathsf{NC}^\ell$. This implies that for all $\ell > i$,

$$\mathsf{NC}^\ell \subseteq \mathsf{NC}^{\ell-1} \subseteq \mathsf{NC}^{\ell-2} \subseteq \cdots \subseteq \mathsf{NC}^{i+1} \subseteq \mathsf{NC}^i$$

and thus $\mathsf{NC} = \bigcup_{\ell > i} \mathsf{NC}^\ell \subseteq \mathsf{NC}^i$.

Suppose $A \in \mathsf{NC}^{\ell+1}$ by a log-space-uniform circuit family $C_1, C_2, \ldots$ of depth $O(\log^{\ell+1} N)$. Group $C_N$'s layers into "blocks": The $0^{\text{th}}$ block is the literals, the $1^{\text{st}}$ block is the first $\log^{i+1} N$ layers, the $2^{\text{nd}}$ block is the next $\log^{i+1} N$ layers, and so on. There are $J = O(\log^{\ell+1} N)/\log^{i+1} N = O(\log^{\ell-i} N)$ many blocks. The idea is that because of our assumption $\mathsf{NC}^{i+1} \subseteq \mathsf{NC}^i$, each block can be modified to have depth $O(\log^i N)$ instead of $\log^{i+1} N$, so the whole modified circuit will have:

$$\big(O(\log^i N) \text{ depth per block}\big) \cdot \big(O(\log^{\ell-i} N) \text{ blocks}\big) \;=\; O(\log^\ell N) \text{ depth}$$

Formally: If $(u, v)$ is a wire and $u$ is in some block $< j$ and $v$ is in some block $\geq j$, we say $(u, v)$ leaves block $j-1$ and enters block $j$. For each $j \in [J]$, let $W_j$ denote the set of wires entering block $j$. Assume $C_N$'s output node is in block $J$ and has an outgoing "wire" in $W_{J+1}$. Note that $W_1, \ldots, W_{J+1}$ might not be pairwise disjoint, since wires may pass through blocks. For each $j \in [J]$, let $C_{N,j}$ be the circuit consisting of block $j$ of $C_N$, with the wires of $W_j$ as input variables and the wires of $W_{j+1}$ as output bits. For each $j \in [J]$ and $k \in [|W_{j+1}|]$, let $C_{N,j,k}$ be $C_{N,j}$ but with the $k^{\text{th}}$ outgoing wire designated as the 1-bit output.

(call this problem $B$, with input size $M$)

Input: $N$, $j$, $k$, an assignment to the wires of $W_j$, and padding to ensure $M \geq N$

Output: $C_{N,j,k}$'s output on this assignment

$B \in \mathsf{NC}^{i+1}$ by having parallel copies of all relevant $C_{N,j,k}$ and selecting the right one for the output using a mux whose selector is the triple $(N, j, k)$. Thus by assumption, $B \in \mathsf{NC}^i$ by some log-space-uniform circuit family $E_1, E_2, \ldots$ of depth $O(\log^i M)$. Let $D_{N,j,k}$ be $E_M$ for some relevant $M$ but with the specific values of $N$, $j$, $k$ plugged in, so the input is just an assignment to $W_j$. Let $D_{N,j}$ be parallel copies of $D_{N,j,k}$ for all $k \in [|W_{j+1}|]$ on a shared input, and note that $D_{N,j}$ has depth $O(\log^i M) = O(\log^i N)$. Let $D_N$ be $C_N$ but with $D_{N,j}$ instead of $C_{N,j}$ for each $j$. That is, $D_N$ consists of $C_N$'s literals feeding into $D_{N,1}$ feeding into $D_{N,2}$ feeding into $D_{N,3}$ and so on, up through $D_{N,J}$. Then $D_1, D_2, \ldots$ solves $A$ (since each $D_N$ is equivalent to $C_N$) and is log-space-uniform and has depth $O(\log^\ell N)$. Therefore $A \in \mathsf{NC}^\ell$.

**Exercise 6.3:** Suppose SQUARING has an $\mathsf{AC}^0$/poly-type circuit family. We design an $\mathsf{AC}^0$/poly-type circuit family for MULTIPLICATION, contradicting Theorem 6.7. To compute $x \cdot y$: First, an $\mathsf{AC}^0$/poly-type ADDITION circuit computes $x + y$. Then, three copies of the hypothesized $\mathsf{AC}^0$/poly-type SQUARING circuit compute $x^2$, $y^2$, and $(x+y)^2$. Then, two copies of an $\mathsf{AC}^0$/poly-type SUBTRACTION circuit compute $(x+y)^2 - x^2 - y^2 = 2xy$. Then, the least significant bit is dropped to get $xy$.

**Exercise 6.4.a:** Otherwise, every unbounded-fan-in circuit could be turned into an equivalent semi-unbounded-fan-in circuit that's only a constant factor bigger, by replacing each $\wedge$ gate with an $\vee$ gate surrounded by $\neg$ gates (like Figure 5.5 but interchanging $\wedge$ and $\vee$).

**Exercise 6.4.b:** $\mathsf{SAC}^1 \subseteq \mathsf{AC}^1$ is trivial since semi-unbounded fan-in is a special case of unbounded fan-in.

To prove $\mathsf{NL} \subseteq \mathsf{SAC}^1$, we adjust the proof that $\mathsf{NL} \subseteq \mathsf{AC}^1$ (Theorem 6.9). The circuit is already almost semi-unbounded-fan-in: The $\neg$ gates are already on the input variables. When $k > 0$, each $v_{B,D,k} = \bigvee_C \left(v_{B,C,k-1} \wedge v_{C,D,k-1}\right)$ contributes bounded-fan-in $\wedge$ gates and an unbounded-fan-in $\vee$ gate. For certain $B$ and $D$, $v_{B,D,0}$ is a width-$O(\log N)$ term, whose unbounded-fan-in $\wedge$ gate can be replaced by a tree of bounded-fan-in $\wedge$ gates of depth $O(\log \log N)$.

**Exercise 6.5:** Let $D_1, D_2, \ldots$ be an $\mathsf{NC}^1$-type circuit family computing a mapping reduction from $A$ to $B$. Let $E_1, E_2, \ldots$ be an $\mathsf{NC}^1$-type circuit family solving $B$. We design an $\mathsf{NC}^1$-type circuit family $C_1, C_2, \ldots$ solving $A$. Say $D_N$ outputs a "query size word" and $K$ "query output words" for some $K \in \text{poly}N$. Let $C_N$ contain $D_N, E_1, E_2, \ldots, E_K$. For each $M \in [K]$, the input to $E_M$ is $D_N$'s first $M$ query output words. Then $C_N$ feeds the outputs of $E_1, E_2, \ldots, E_K$ into a mux whose selector is $D_N$'s query size word. So $C_N(x) = E_M(y)$ where $y$ is the actual query of size $M$ on a valid input $x$ to $A$. Thus $C_1, C_2, \ldots$ solves $A$ since $A(x) = B(y) = E_M(y) = C_N(x)$. Also, $C_N$ has bounded fan-in and depth $O(\log N)$ (for $D_N$) plus $O(\log K) = O(\log N)$ (for $E_1, E_2, \ldots, E_K$ in parallel) plus $O(\log N)$ for the mux, and is log-space-uniform.

**Exercise 6.6:** It suffices to show Branching Program Evaluation $\leq^{m}_{nc^1}$ Forest Reachability since Branching Program Evaluation is L-complete (Theorem 6.15). Map $(B, a)$ to $(G, s, t)$ as follows. As in the proof of Theorem 6.16, define the dag $B_a$, which consists of two disjoint in-trees rooted at the accept and reject nodes, and note that $B(a) = 1$ iff the start node is in the accept node's in-tree. Let $G$ be $B_a$ but ignoring the edge directions—so $G$ is a forest with two trees—and let $s$ be the start node and $t$ be the accept node. This mapping reduction is correct since $B(a) = 1$ iff $s$ and $t$ are in the same connected component in $G$.

To compute this reduction with an $\mathsf{NC}^1$-type circuit family—or more generally, to convert any directed graph to its undirected version—we first convert from the adjacency list encoding to the adjacency matrix encoding (Lemma 6.18), then change the $(u, v)$ entry to the $\vee$ of the $(u, v)$ entry and the $(v, u)$ entry for each pair of nodes $(u, v)$, and then convert the resulting adjacency matrix to its adjacency list encoding (Lemma 6.19).

**Exercise 6.7:** Since CIRCUIT EVALUATION is P-complete (Theorem 5.15), we know that CIRCUIT EVALUATION $\in \mathsf{AC}^1$ iff $\mathsf{AC}^1 = \mathsf{P}$ (Lemma 6.11). To change $\mathsf{AC}^1$ to $\mathsf{NC}^1$, log-space reductions no longer suffice—we need $\mathsf{NC}^1$-type reductions. Specifically: For every problem $B$ that's P-complete via $\leq^m_{nc^1}$, we have $B \in \mathsf{NC}^1$ iff $\mathsf{NC}^1 = \mathsf{P}$. This is analogous to the corresponding fact about L-completeness (Lemma 6.14). So we just need to prove CIRCUIT EVALUATION is P-complete via $\leq^m_{nc^1}$. We already know CIRCUIT EVALUATION $\in \mathsf{P}$ (Lemma 5.10). Every $A \in \mathsf{P}$ has a log-space-uniform circuit family $C_1, C_2, \ldots$ (Theorem 5.13), so $A \leq^m_{nc^1}$ CIRCUIT EVALUATION by mapping $x$ of size $N$ to $(C_N, x)$. This is correct since $A(x) = 1 \iff C_N(x) = 1 \iff (C_N, x)$ is a 1-input of CIRCUIT EVALUATION. It is $\mathsf{NC}^1$-type (indeed, $\mathsf{NC}^0$-type) because each query bit is either hardwired (depending only on $N$) or a bit of $x$, and a log-space generator for the reduction circuit can determine the hardwired bits by running the log-space generator to get $C_N$.

**Exercise 6.8:** For any rooted tree with $\ell \geq 2$ leaves and where each node has at most two children, the following algorithm finds a node $v$ with $> \ell/2$ leaves such that $v$'s children $v_{\text{left}}$ and $v_{\text{right}}$ both have $\leq \ell/2$ leaves.

```
initialize v ← the root
while v has a child with > ℓ/2 leaves:
    update v ← some child of v with > ℓ/2 leaves
```

This trivially maintains the invariant that $v$ has $> \ell/2$ leaves. It stops at some $v$ whose children each have $\leq \ell/2$ leaves. This $v$ must have two children: If $v$ had one child, then the algorithm wouldn't stop at $v$. If $v$ had no children, then $v$ would have $1 \leq \ell/2$ leaf (namely itself) and so the algorithm wouldn't reach $v$.

It suffices to find a bounded-fan-in circuit $C$ that's equivalent to $\varphi$ and has depth $\leq 3\log_2(\ell)$ (Theorem 6.1.*(i)*).

```
balance(φ):
    if φ has only one leaf: return φ
    if φ has ℓ ≥ 2 leaves:
        find a node v that has > ℓ/2 leaves and v_left and v_right both have ≤ ℓ/2 leaves in φ
        let C_{v_left} ← balance(φ_{v_left}) and C_{v_right} ← balance(φ_{v_right})
        let C_{v=0} ← balance(φ_{v=0}) and C_{v=1} ← balance(φ_{v=1})
        return a circuit C with a mux
            whose selector is the same gate type as v, applied to the outputs of C_{v_left} and C_{v_right}
            and whose other two inputs are the outputs of C_{v=0} and C_{v=1}
```

This recursive subroutine maintains the invariant that balance($\varphi$) is equivalent to $\varphi$. This holds by definition for the base cases when $\varphi$ has only one leaf. For non-base cases, the invariant is maintained because if $C_{v_{\text{left}}}$, $C_{v_{\text{right}}}$, $C_{v=0}$, $C_{v=1}$ are equivalent to $\varphi_{v_{\text{left}}}$, $\varphi_{v_{\text{right}}}$, $\varphi_{v=0}$, $\varphi_{v=1}$, then for every input $x$, assuming $v$ is an $\wedge$ gate (the argument is the same if $v$ is an $\vee$ gate) and letting $b = C_{v_{\text{left}}}(x) \wedge C_{v_{\text{right}}}(x) = \varphi_{v_{\text{left}}}(x) \wedge \varphi_{v_{\text{right}}}(x) = \varphi_v(x)$ we have $C(x) = C_{v=b}(x) = \varphi_{v=b}(x) = \varphi(x)$, so $C$ is equivalent to $\varphi$.

Base cases correspond to literals, which contribute nothing to the depth. Each level of recursion corresponds to a level of muxes, contributing two layers of $\wedge$ and $\vee$ gates, plus one more layer for the gates corresponding to the various $v$ gates. If $\varphi$ has $\ell \geq 2$ leaves, then $\varphi_{v_{\text{left}}}$, $\varphi_{v_{\text{right}}}$ each have $\leq \ell/2$ leaves and $\varphi_{v=0}$, $\varphi_{v=1}$ each have $< \ell - \ell/2 = \ell/2$ leaves since $\varphi_v$ has $> \ell/2$ leaves. Since $\ell$ gets multiplied by $\leq 1/2$ at each level, each formula passed to a call at level $i$ has $\leq \ell/2^{i-1}$ leaves. Thus the recursion always reaches a base case after $\leq \log_2(\ell)$ levels, so $C$ has depth $\leq 3\log_2(\ell)$.

**Exercise 6.9.a:** $F_N(x) = 1$ iff there exists an odd $i$ such that $x_i = 1$ and $x_j = 1$ for all even $j < i$. Thus we can compute $F_N$ with an unbounded-fan-in depth-2 monotone formula whose output is an $\vee$ gate of fan-in $\lceil N/2 \rceil$ where each incoming wire corresponds to an odd $i$ and comes from an $\wedge$ gate of fan-in $\lceil i/2 \rceil$ with incoming wires from $x_i$ and $x_j$ for all even $j < i$. We convert each gate to a tree of bounded-fan-in gates of depth $\leq \lceil \log(\lceil N/2 \rceil) \rceil \leq \log N + O(1)$ as in §6.1.2. The total depth is $\leq 2 \log N + O(1)$.

**Exercise 6.9.b:** Let the constant $c \geq 2$ be the maximum over all $N \leq 1000$ of the minimum depth of any bounded-fan-in monotone formula for $F_N$. The following recursive subroutine maintains the invariant that if $N \geq 3$ is odd, then to-formula($N$) returns a bounded-fan-in monotone formula of depth $\leq 1.45 \log N + c$ for $F_N$. Here, $L$ and $R$ stand for "left" and "right."

```
to-formula(N):
    if N ≤ 1000: return any minimum-depth bounded-fan-in monotone formula for F_N
    R ← 0.382N rounded up to an odd integer
    L ← N − R + 1
    φ ← to-formula(L)
    ψ ← to-formula(R)
    return φ(x_1 ⋯ x_{L−1} 0) ∨ ( ψ(x_L ⋯ x_N) ∧ ⋀_{even j < L} x_j )
        where ⋀_{even j < L} x_j is a balanced tree of bounded-fan-in ∧ gates
```

The invariant holds when $N \leq 1000$, by definition. To see that the invariant is maintained when $N > 1000$, assume the invariant holds for to-formula($L$) and to-formula($R$). Note that $R \leq 0.382N + 2 \leq 0.384N$ and $L \leq 0.618N + 1 \leq 0.619N$ since $N > 1000$. Thus $\varphi$ is a bounded-fan-in monotone formula of depth

$$\begin{aligned} &\leq 1.45 \log L + c \\ &\leq 1.45 \log(0.619N) + c \\ &\leq 1.45 \log 0.619 + 1.45 \log N + c \\ &\leq -1 + 1.45 \log N + c \end{aligned}$$

for $F_L$, and $\psi$ is a bounded-fan-in monotone formula of depth

$$\begin{aligned} &\leq 1.45 \log R + c \\ &\leq 1.45 \log(0.384N) + c \\ &= 1.45 \log 0.384 + 1.45 \log N + c \\ &\leq -2 + 1.45 \log N + c \end{aligned}$$

for $F_R$. Note that $\bigwedge_{\text{even } j < L} x_j$ has depth:

$$\lceil \log(\lfloor L/2 \rfloor) \rceil \leq \lceil \log L - 1 \rceil \leq \log L \leq \log N$$

Thus the formula returned by to-formula($N$) has depth:

$$\begin{aligned} &\max\Big(1 + \text{depth of } \varphi,\ \ 2 + \text{depth of } \psi,\ \ 2 + \text{depth of } \textstyle\bigwedge_{\text{even } j < L} x_j\Big) \\ \leq\ &\max\big(1 - 1 + 1.45 \log N + c,\ \ 2 - 2 + 1.45 \log N + c,\ \ 2 + \log N\big) \\ =\ &1.45 \log N + c \end{aligned}$$

We claim that for all $x \in \{0,1\}^N$, $F_N(x) = 1$ iff the formula returned by to-formula($N$) evaluates to 1:

$\Rightarrow$: Assume $F_N(x) = 1$, and consider two cases. If there exists an odd $i < L$ such that $x_i = 1$ and $x_j = 1$ for all even $j < i$, then $F_L(x_1 \cdots x_{L-1}0) = 1$ and thus $\varphi(x_1 \cdots x_{L-1}0) = 1$. Otherwise, there exists an odd $i \geq L$ such that $x_i = 1$ and $x_j = 1$ for all even $j < i$, in which case $F_R(x_L \cdots x_N) = 1$ (since relabeling $x_L \cdots x_N$ as $y_1 \cdots y_R$, we have that $i - L + 1$ is odd and $y_{i-L+1} = 1$ and $y_j = 1$ for all even $j < i - L + 1$) and thus $\psi(x_L \cdots x_N) = 1$, and $\bigwedge_{\text{even } j < L} x_j = 1$ (since $i \geq L$).

$\Leftarrow$: Assume the formula returned by to-formula($N$) evaluates to 1, and consider two cases. If $\varphi(x_1 \cdots x_{L-1}0) = 1$, then $F_L(x_1 \cdots x_{L-1}0) = 1$ and thus there exists an odd $i < L \leq N$ such that $x_i = 1$ and $x_j = 1$ for all even $j < i$, so $F_N(x) = 1$. Otherwise, $\psi(x_L \cdots x_N) = 1$ and $\bigwedge_{\text{even } j < L} x_j = 1$, in which case $F_R(x_L \cdots x_N) = 1$ and thus there exists an odd $i \geq L$ such that $x_i = 1$ and $x_j = 1$ for all even $j$ with $L < j < i$, and so $F_N(x) = 1$ since $x_i = 1$ and $x_j = 1$ for all even $j < i$.

**Exercise 6.9.c:** To compute $x_{n-1}\cdots x_0 + y_{n-1}\cdots y_0$, first compute $g_i = x_i \wedge y_i$ ("generate") and $p_i = x_i \oplus y_i$ ("propagate") for each $i$. Then, the carry-in to coordinate 0 is $c_0 = 0$, and the carry-in to coordinate $i \in \{1,\ldots,n\}$ is

$$c_i \;=\; g_{i-1} \vee (p_{i-1} \wedge c_{i-1}) \;=\; g_{i-1} \vee \big(p_{i-1} \wedge \big(g_{i-2} \vee \big(p_{i-2} \wedge \big(\cdots g_0 \vee \big(p_0 \wedge c_0\big)\cdots\big)\big)\big)\big)$$

computed by plugging $g_{i-1}p_{i-1}g_{i-2}p_{i-2}\cdots g_0p_0c_0$ into a formula for $F_{2i+1}$. Finally, compute the sum $z_n\cdots z_0$ by $z_n = c_n$ and $z_i = x_i \oplus y_i \oplus c_i$ for each $i < n$. Since $2i+1 \le 2n+1 \le N+1$ where $N = 2n$ is the input size, the formula for $F_{2i+1}$ has depth $\le d\log(2i+1) + O(1) \le d\log(N+1) + O(1) \le d\log N + O(1)$. Computing all $g_i$, $p_i$, and $z_i$ just adds $O(1)$ more layers, so our circuit's total depth is $d\log N + O(1)$.

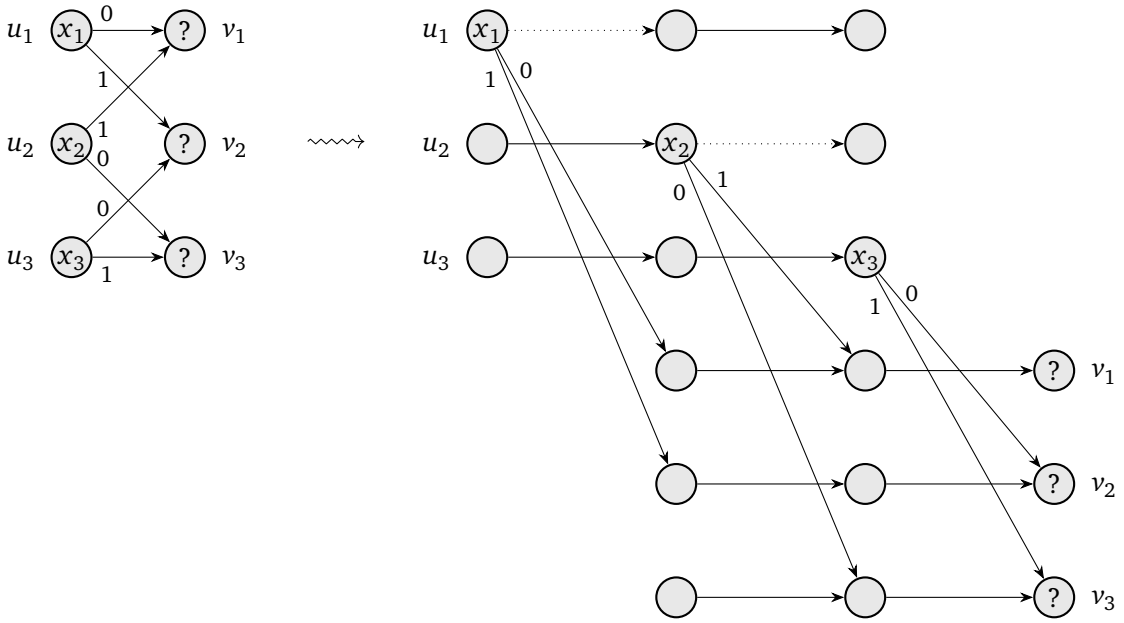
**Exercise 6.10.a:** For any width-$w$ length-$\ell$ layered branching program $B$, we design an equivalent width-$2w$ length-$\ell w$ oblivious layered branching program $B'$. Assume each layer of $B$ has exactly $w$ nodes, by adding dummy nodes if necessary.

Consider any non-output layer of $B$. Let $u_1, u_2, \ldots, u_w$ be this layer's nodes. Let $v_1, v_2, \ldots, v_w$ be $B$'s next layer's nodes. For notational convenience, assume $u_1, u_2, \ldots, u_w$ read variables $x_1, x_2, \ldots, x_w$ respectively. (The same argument works for any choice of variables.) To obtain $B'$, replace this layer of $B$ with a group of $w$ many layers, thus ensuring $B'$ has length $\ell w$. Each node in this group's $i^{\text{th}}$ layer reads $x_i$ (or is an unconditional branch, which can just read $x_i$ and have both edges go to the same node as each other), thus ensuring $B'$ is oblivious.

This group of layers consists of $2w$ many "lanes" of unconditional branch nodes, thus ensuring $B'$ has width $2w$: For each $i \in [w]$, $B'$ has a lane leading to $v_i$ in the next group. For each $i \in [w]$, $B'$ has a lane leading from $u_i$ in this group, except that when the lane reaches the layer that reads $x_i$, then it's not an unconditional branch: Assuming $u_i$ has a 0-edge to $v_j$ and a 1-edge to $v_k$ in $B$, the node that reads $x_i$ in $B'$ has a 0-edge into the lane leading to $v_j$, and a 1-edge into the lane leading to $v_k$. Nodes with no outgoing edges (the end of $u_i$'s lane) should be deleted.

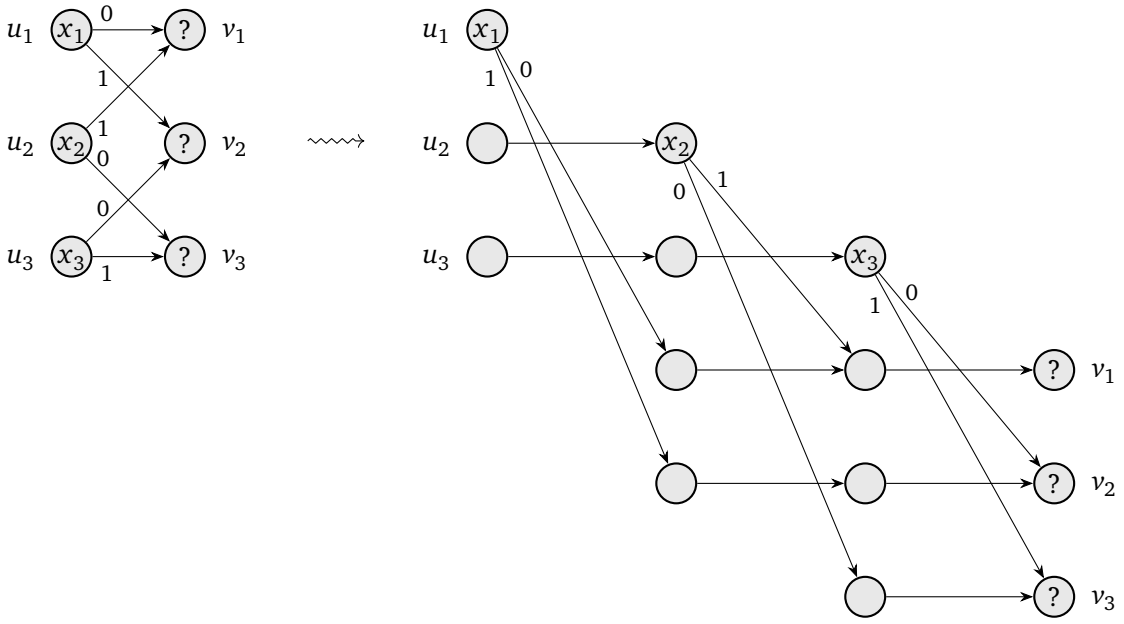
$B'$ is equivalent to $B$: For every input $x$, if the computation of $B(x)$ reaches $u_i$ and steps to $v_j$, then the computation of $B'(x)$ reaches $u_i$, waits until the layer that reads $x_i$, and then steps into the lane leading to $v_j$. Since the computation of $B'(x)$ visits all nodes corresponding to the nodes that the computation of $B(x)$ visits, $B'(x)$ outputs the same thing as $B(x)$.

Here's an example with $w = 3$ and $2w = 6$. The dotted edges are where the unconditional branches in each lane would be, when $B'$ actually has a conditional branch instead.



**Exercise 6.10.b:** We use the same construction from Exercise 6.10.a but delete all unreachable nodes. In the group's $i^{\text{th}}$ layer, we only keep the nodes in the lanes of $u_i, u_{i+1}, \ldots, u_w$, of which there are $w-i+1$, and the nodes in the lanes of all $v_j$ that have an edge from any of $u_1, u_2, \ldots, u_{i-1}$ in $B$, of which there are $\leq \min(2(i-1), w)$ since each of $u_1, u_2, \ldots, u_{i-1}$ has only 2 outgoing edges in $B$. Thus the group's $i^{\text{th}}$ layer has $\leq w-i+1+\min(2(i-1), w)$ many nodes. When $2(i-1) \leq w$, this is $w-i+1+2(i-1) = w+i-1 \leq w+(w/2+1)-1 = 1.5w$. When $2(i-1) \geq w$, this is $w-i+1+w = 2w-i+1 \leq 2w-(w/2+1)+1 = 1.5w$. Thus $B'$ has width $\leq 1.5w$. Since the width is an integer, it is $\leq \lfloor 1.5w \rfloor$.

Here's the revised example from Exercise 6.10.a with $w = 3$ and $\lfloor 1.5w \rfloor = 4$.



**Exercise 6.11.a:** For starters, MAJORITY $\in$ L by a program that sums the $N$ input bits and compares the sum to $N/2$, so MAJORITY has a poly-size branching program family (Theorem 5.24). These branching programs are already almost layered—except for unconditional branches to output nodes—but we can turn them into poly-width poly-length layered branching programs anyway (Theorem 6.25).

To confirm that this only needs linear width and linear length, we explicitly describe a layered branching program family $B_1, B_2, \ldots$ for MAJORITY. For simplicity, we let $B_N$ have width $N+1$, where the nodes in each layer are numbered $0, 1, \ldots, N$. For each $i \in \{1, \ldots, N\}$, all nodes in layer $i$ read $x_i$, and for each $j \in \{0, \ldots, N-1\}$, node $j$ in layer $i$ has a 0-edge to node $j$ in layer $i+1$ and a 1-edge to node $j+1$ in layer $i+1$. The start node is node 0 in layer 1. Thus the computation of $B_N(x)$ visits node $j$ in layer $i$ iff $\sum_{h=1}^{i-1} x_h = j$. Since $\text{MAJORITY}_N(x) = 1$ iff $\sum_{h=1}^{N} x_h > \lfloor N/2 \rfloor$, nodes $0, 1, \ldots, \lfloor N/2 \rfloor$ in layer $N+1$ unconditionally branch to the reject node, and the rest unconditionally branch to the accept node. Thus $B_N$ computes $\text{MAJORITY}_N$.

Actually, we only need width $\lfloor N/2 \rfloor + 2$ because we can turn nodes $\lfloor N/2 \rfloor + 1$ in all layers into a lane of nodes (each unconditionally branching to the next) leading to the accept node, and discard nodes $\lfloor N/2 \rfloor + 2, \ldots, N$ in all layers. This is because once $\lfloor N/2 \rfloor + 1$ many 1s have been seen in $x$ so far, the output will be 1 regardless of the rest of the input.

**Exercise 6.11.b:** MAJORITY $\in \mathsf{NC}^1$ by using an $\mathsf{NC}^1$-type circuit for ITERATED ADDITION (Theorem 6.6) to sum the input bits, and then an $\mathsf{NC}^1$-type circuit to check whether the sum is $> \lfloor N/2 \rfloor$ (§5.2.5). So MAJORITY has a bounded-fan-in log-depth formula family (Theorem 6.1.*(i)*) and thus also a constant-width poly-length layered branching program family (Theorem 6.28).

**Exercise 7.1:** $\overline{\text{EZE}} \in \text{NP}$ because given $a \in \mathbb{Z}_2^n$ as a purported witness, a poly-time verifier can evaluate $\varphi(a)$ and accept iff it equals 1.

It suffices to show 3-SAT $\leq_p^m \overline{\text{EZE}}$ since 3-SAT is NP-complete (Theorem 2.11). Map an $n$-variate 3-CNF $\psi$ to an $n$-variate arithmetic formula $\varphi$ as follows: For any variable $x_i$, let $\overline{x_i}$ also denote the arithmetic formula $1 - x_i$. For each clause $(\ell_i \vee \ell_j \vee \ell_k) = \neg(\overline{\ell_i} \wedge \overline{\ell_j} \wedge \overline{\ell_k})$ in $\psi$, have a factor $\left(1 - (\overline{\ell_i} \cdot \overline{\ell_j} \cdot \overline{\ell_k})\right)$ in $\varphi$. For example:

$$\psi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5}) \;\; \Rightarrow \;\; \varphi = \Big(1 - (1 - x_1)x_2(1 - x_3)\Big)\Big(1 - x_3(1 - x_4)x_5\Big)$$

This poly-time mapping reduction is correct since $\psi(a) = \varphi(a)$ for all $a \in \{0,1\}^n$ and thus $\psi$ is satisfiable iff there exists $a$ such that $\varphi(a) = 1$.

**Exercise 7.2.a:** As in the formula setting (§ 7.1.2), we define Circuit Polynomial Zero Testing (CPZT) and reduce from CPIT to CPZT in linear time by mapping $(C, D)$ to $C - D$. We show CPZT $\in$ coRTIME$[N]$, which implies CPIT $\in$ coRTIME$[N]$. On input $(p, C)$, the algorithm picks an assignment uniformly at random, evaluates $C$ on it, and accepts iff $C$ evaluates to 0. If $C$ is identically zero then the algorithm accepts with probability 1. Otherwise, the polynomial expressed by $C$ has $\leq (p/3)p^{n-1}$ many roots (Lemma 7.1), so the algorithm accepts (errs) with probability $\leq (p/3)p^{n-1}/p^n = 1/3$.

We just need to show that Arithmetic Circuit Evaluation has a linear-time algorithm. (This is a function problem, not a decision problem, since the output is an element of $\mathbb{Z}_p$.)

Arithmetic Circuit Evaluation
Input: Prime $p \leq N$, $n$-variate arithmetic circuit $C$ over $\mathbb{Z}_p$, and assignment $a \in \mathbb{Z}_p^n$
Output: $C(a)$

Our algorithm is like the one for Circuit Evaluation (Lemma 5.10):

topologically order $C$'s nodes $v_1, v_2, \ldots, v_m$
for $i \leftarrow 1, 2, \ldots, k$ where $v_k$ is the output node:
    if $v_i$ is an input node $x_j$: $value[i] \leftarrow a_j$
    if $v_i$ is a constant node $c$: $value[i] \leftarrow c$
    if $v_i$ is a $+$ gate with wires from $v_h$ and $v_j$: $value[i] \leftarrow value[h] + value[j]$
    similarly if $v_i$ is a $-$ or $\cdot$ gate
output $value[k]$

**Exercise 7.2.b:** Suppose for contradiction that $P$ and $Q$ are distinct multilinear canonical-form polynomials that both agree with $f\colon \{0,1\}^n \to \{0,1\}$. Then $P-Q$ is a nonzero multilinear canonical-form polynomial and $(P-Q)(a) = P(a) - Q(a) = f(a) - f(a) = 0$ for all $a \in \{0,1\}^n$. Consider any smallest set $S \subseteq [n]$ such that $\prod_{i \in S} x_i$ has a nonzero coefficient $c$ in $P-Q$. Define $a \in \{0,1\}^n$ by $a_i = 1$ for all $i \in S$ and $a_i = 0$ for all $i \notin S$. The monomial $c \prod_{i \in S} x_i$ evaluates to $c \prod_{i \in S} a_i = c$, and each other monomial $b \prod_{i \in T} x_i$ with $b \neq 0$ evaluates to $b \prod_{i \in T} a_i = 0$ since $T$ contains some $i \notin S$ (by the minimality of $S$). Thus $(P-Q)(a) = c \neq 0$, a contradiction.

**Exercise 7.2.c:** We show ROBP EQ $\leq_p^m$ CPIT, which implies ROBP EQ $\in$ coRP since CPIT $\in$ coRP (Exercise 7.2.a). Map $(A, B)$ to $(p, C, D)$ where $p$ is a prime between $3n$ and $6n$ (which exists and can be found in poly $n$ time by trying every possibility and testing whether it's prime by trying all possible divisors) and $C$ is defined from $A$ (and $D$ is defined from $B$) like the conversion of branching programs to logic circuits (Theorem 5.22.*(i)*):

Every node $v$ in $A$ has an associated node $g_v$ in $C$. If $v$ is $A$'s start node, then $g_v$ is $C$'s output node. If $v$ is $A$'s accept node, then $g_v = 1$. If $v$ is $A$'s reject node, then $g_v = 0$. If $v$ is a node of $A$ labeled by variable $x_i$ and with 0-edge to $u$ and 1-edge to $w$, then $g_v = (1 - x_i) \cdot g_u + x_i \cdot g_w$. Since $A$ is a dag, $C$ is a dag.

We claim that $C$ agrees with $A$ (and $D$ agrees with $B$): For every $a \in \{0,1\}^n$ and every node $v$ in $A$, the value of $g_v$ in $C$ is what $A$ would output if $v$ were the start node, so $C(a) = A(a)$.

Since $A$ and $B$ are read-once, the canonical-form polynomials of $C$ and $D$ are multilinear and have total degree $\leq n \leq p/3$, so $(p, C, D)$ is a valid input to CPIT. To show this poly-time mapping reduction is correct, we argue that $A$ and $B$ are equivalent iff $C$ and $D$ are identical:

$\Rightarrow$: If $A$ and $B$ both compute $f : \{0,1\}^n \to \{0,1\}$, then $C$ and $D$ both agree with $f$. Since the canonical-form polynomials of $C$ and $D$ are multilinear, they are the same (Exercise 7.2.b), which means $C$ and $D$ are identical.

$\Leftarrow$: If $A$ and $B$ are not equivalent then $C$ and $D$ are not identical.

Technicality: Our definition of CPIT requires $p \leq N$. In case $6n > N$, we could tweak CPIT to allow padding or allow $p \leq N^{O(1)}$.

**Exercise 7.3:** The algorithm still picks $x \in \{0,1\}^n$ uniformly at random and accepts iff $A(Bx) = Cx$. With some $O(\log N)$ word size, each intermediate calculation's result fits in one word. To analyze the algorithm, we first adapt the random subsum principle (Lemma 7.2):

**Lemma.** *Assuming $x \in \{0,1\}^n$ is sampled uniformly at random:*

*(i)* $\mathbf{Pr}[v \cdot x = 0] \leq 1/2$ *for all* $v \in \mathbb{Z}^n$ *such that* $v \neq 0^n$.

*(ii)* $\mathbf{Pr}[v \cdot x = w \cdot x] \leq 1/2$ *for all* $v, w \in \mathbb{Z}^n$ *such that* $v \neq w$.

**Proof.** *(i)*: Let $i$ be an index such that $v_i \neq 0$ (it doesn't matter which one). Consider any *partners* $y, z \in \{0,1\}^n$, meaning $y_i = 1$ and $z_i = 0$ and $y_j = z_j$ for all $j \neq i$. Then $y - z$ has a 1 at index $i$ and 0s everywhere else. By linearity, $(v \cdot y) - (v \cdot z) = v \cdot (y - z) = v_i \neq 0$. This means $v \cdot y \neq v \cdot z$, so at most one (perhaps neither) of them is 0. Partitioning $\{0,1\}^n$ into partner pairs, we see that $v \cdot x = 0$ for at most half of all $x \in \{0,1\}^n$.

*(ii)*: If $v \neq w$ then $v - w \neq 0^n$, so by linearity and *(i)*:

$$\mathbf{Pr}[v \cdot x = w \cdot x] \;=\; \mathbf{Pr}\Big[(v \cdot x) - (w \cdot x) = 0\Big] \;=\; \mathbf{Pr}\Big[(v - w) \cdot x = 0\Big] \;\leq\; 1/2 \qquad \blacksquare$$

Now, we analyze the algorithm's correctness. If $AB = C$ then the algorithm accepts with probability 1. If $AB \neq C$ then some row $i$ of $AB$ is different from row $i$ of $C$, so by *(ii)* applied to the vectors $(AB)_i \neq C_i$:

$$\begin{aligned}
\mathbf{Pr}[\text{algorithm accepts}] \;&=\; \mathbf{Pr}\Big[(AB)x = Cx\Big] \\
&\leq\; \mathbf{Pr}\Big[((AB)x)_i = (Cx)_i\Big] \\
&=\; \mathbf{Pr}\Big[(AB)_i \cdot x = C_i \cdot x\Big] \\
&\leq\; 1/2
\end{aligned}$$

**Exercise 7.4:** Suppose $A \in \mathsf{NP}$ by a verifier $V$ with time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$. By the definition of running time for verifiers, $V$ accesses at most the first $T$ words of its witness segment. Then $A \in \mathsf{PP}$ by this poly-time randomized algorithm:

on input $x$:
    run $V(x;w)$ for a uniformly random $w \in (\{0,1\}^W)^T$
    if it accepted: accept
    toss $TW+1$ coins and view the outcome as a binary number $r \in \{0,1,\ldots,2^{TW+1}-1\}$
    if $r > 2^{TW}$: accept
    reject

If $A(x) = 0$ then $V(x;w)$ accepts with probability 0, so this algorithm accepts iff $r > 2^{TW}$, which happens with probability $(2^{TW}-1)/2^{TW+1} = 1/2 - 2^{-TW-1} < 1/2$.

If $A(x) = 1$ then $V(x;w)$ accepts with probability $\geq 1/2^{TW}$, so by the law of total probability, this algorithm accepts with probability:

$$\begin{aligned}
& \mathbf{Pr}\big[V(x;w) \text{ accepts}\big] \cdot 1 + \mathbf{Pr}\big[V(x;w) \text{ rejects}\big] \cdot \mathbf{Pr}[r > 2^{TW}] \\
\geq\ & 2^{-TW} \cdot 1 + (1 - 2^{-TW}) \cdot (2^{-1} - 2^{-TW-1}) \\
=\ & 2^{-TW} + (2^{-1} - 2^{-TW-1} - 2^{-TW-1} + 2^{-2TW-1}) \\
=\ & 1/2 + 2^{-2TW-1} \\
>\ & 1/2
\end{aligned}$$

**Exercise 7.5:** This is trivial for $n = 1$ since $1e^0 = 1$, so assume $n > 1$. Let $A$ be the event that each element of $[n]$ appears exactly once. For each $j \in [n]$, let $A_j$ be the event that $j$ appears exactly once. By the chain rule:

$$\begin{aligned}\mathbf{Pr}[A] &= \mathbf{Pr}[A_n \cap A_{n-1} \cap A_{n-2} \cap \cdots \cap A_2] \\ &= \mathbf{Pr}[A_n] \cdot \mathbf{Pr}[A_{n-1} \mid A_n] \cdot \mathbf{Pr}[A_{n-2} \mid A_n \cap A_{n-1}] \cdots \mathbf{Pr}[A_2 \mid A_n \cap A_{n-1} \cap \cdots \cap A_3]\end{aligned}$$

For each $j \in \{2, \ldots, n\}$, $\mathbf{Pr}[A_j \mid A_n \cap A_{n-1} \cap \cdots \cap A_{j+1}]$ equals the probability that $j$ appears exactly once in a uniformly random element of $[j]^j$, because after conditioning on $A_n \cap A_{n-1} \cap \cdots \cap A_{j+1}$, we can omit the coordinates having the unique occurrences of $n, n-1, \ldots, j+1$ and re-index the remaining coordinates with $[j]$, and the tuple of values in those coordinates is uniformly distributed over $[j]^j$. The probability that $j$ appears exactly once in a uniformly random element of $[j]^j$ is

$$j \cdot (1/j) \cdot (1-1/j)^{j-1} \;=\; \tfrac{j}{j-1} \cdot (1-1/j)^j \;\leq\; \tfrac{j}{j-1} e^{(-1/j)j} \;=\; \tfrac{j}{j-1} e^{-1}$$

since there are $j$ possible coordinates for the unique occurrence of $j$, and probability $1/j$ that $j$ occurs there, and each of the other $j-1$ coordinates has probability $1-1/j$ of not being $j$ (and using Fact 7.6 with $x = -1/j$). Thus:

$$\mathbf{Pr}[A] \;\leq\; \tfrac{n}{n-1} e^{-1} \cdot \tfrac{n-1}{n-2} e^{-1} \cdot \tfrac{n-2}{n-3} e^{-1} \cdots \tfrac{2}{1} e^{-1} \;=\; n e^{-(n-1)}$$

**Exercise 7.6:** $\Leftarrow$: Assume there exists such a $\Pi$, say with time efficiency $T$ and word size $W'$. For any input size $N$, let $R \subseteq (\{0,1\}^{W'})^T$ be the set of all $r$ such that $\forall$ valid $x$ of size $N$ : $\Pi(x;r) = A(x)$. By assumption, $\mathbf{Pr}_r[r \in R] \geq 2/3$. For every valid $x$ of size $N$, since $\Pi(x;r) = A(x)$ for all $r \in R$ (and possibly for some other $r$), we have $\mathbf{Pr}_r\big[\Pi(x;r) = A(x)\big] \geq \mathbf{Pr}_r[r \in R] \geq 2/3$. Thus $\Pi$ shows that $A \in \mathsf{BPP}$.

$\Rightarrow$: This is like the proof of $\mathsf{BPP} \subseteq \mathsf{P/poly}$ (Theorem 7.9). Suppose $A \in \mathsf{BPP}$ and $A$ has word size $W = O(\log N)$, so inputs of size $N$ are in $(\{0,1\}^W)^N$. By amplification (§7.3.2), $A$ has a poly-time randomized program $\Pi$ with error probability $\leq 2^{-NW}/3$. Say $\Pi$ has time efficiency $T$ and word size $W'$. For any input size $N$, the randomness $r$ is uniformly distributed over $(\{0,1\}^{W'})^T$. For each valid input $x$ of size $N$, let $B_x = \{r : \Pi(x;r) \neq A(x)\}$ be the "bad" event that $\Pi$ errs on input $x$. There are at most $2^{NW}$ valid inputs $x$ of size $N$, and $\mathbf{Pr}[B_x] \leq 2^{-NW}/3$ for each such $x$, so by a union bound:

$$\begin{aligned}
\mathbf{Pr}_r\big[\neg\forall \text{ valid } x \text{ of size } N : \Pi(x;r) = A(x)\big] &= \mathbf{Pr}\big[\textstyle\bigcup_{\text{valid } x \text{ of size } N} B_x\big] \\
&\leq \textstyle\sum_{\text{valid } x \text{ of size } N} \mathbf{Pr}[B_x] \\
&\leq 2^{NW} \cdot 2^{-NW}/3 \\
&= 1/3
\end{aligned}$$

**Exercise 7.7.a:** $\mathsf{S}_2\mathsf{P} \subseteq \Sigma_2\mathsf{P}$ because if $A \in \mathsf{S}_2\mathsf{P}$ by a 2-witness verifier $V$, then $A \in \Sigma_2\mathsf{P}$ by the same $V$:

$$A(x)=1 \Rightarrow \big(\exists w_1\ \forall w_2 :\ V(x;w_1;w_2) \text{ accepts}\big)$$
$$A(x)=0 \Rightarrow \big(\exists w_2\ \forall w_1 :\ V(x;w_1;w_2) \text{ rejects }\big) \Rightarrow \big(\forall w_1\ \exists w_2 :\ \ V(x;w_1;w_2) \text{ rejects }\big)$$

$\mathsf{S}_2\mathsf{P} \subseteq \Pi_2\mathsf{P}$ because $\mathsf{S}_2\mathsf{P}$ is closed under complement: If $A \in \mathsf{S}_2\mathsf{P}$ by a 2-witness verifier $V$, then $\overline{A} \in \mathsf{S}_2\mathsf{P}$ by the 2-witness verifier $V'(x;w_2;w_1) = \overline{V}(x;w_1;w_2)$ that interchanges $w_1$ and $w_2$ and outputs the opposite bit as $V$ does:

$$\overline{A}(x)=1 \Rightarrow \big(\exists w_2\ \forall w_1 :\ V(x;w_1;w_2) \text{ rejects }\big) \Rightarrow \big(\exists w_2\ \forall w_1 :\ V'(x;w_2;w_1) \text{ accepts}\big)$$
$$\overline{A}(x)=0 \Rightarrow \big(\exists w_1\ \forall w_2 :\ V(x;w_1;w_2) \text{ accepts}\big) \Rightarrow \big(\exists w_1\ \forall w_2 :\ V'(x;w_2;w_1) \text{ rejects }\big)$$

**Exercise 7.7.b:** Suppose $A \in \mathsf{BPP}$, say $A \in \mathsf{BPTIME}[N^d]$ for an integer $d$. By amplification, $A$ has a $1/N^{d+1}$-error randomized program $\Pi$ with time efficiency $\leq T = cN^d \log N$ (for an integer $c$) and log-bounded word size $W$. Define a 2-witness verifier $V(x;s;r)$:

- $s = (s_1, \ldots, s_k)$ where each $s_i \in (\{0,1\}^W)^T$ for $k = TW$.
- $r = (r_1, \ldots, r_\ell)$ where each $r_j \in (\{0,1\}^W)^T$ for $\ell = kTW$.
- $V(x;s;r)$ runs $\Pi(x; r_j \oplus s_i)$ for each $i$ and $j$, and accepts iff $\forall j\ \exists i:\ \Pi(x; r_j \oplus s_i)$ accepts.

Assuming $A(x) = 0$, we show that $\exists r\ \forall s:\ V(x;s;r)$ rejects. In the proof of Theorem 7.10, we showed that $\forall s\ \mathbf{Pr}_q\big[\exists i:\ \Pi(x; q \oplus s_i) \text{ accepts}\big] \leq 1/3$ for uniformly random $q \in (\{0,1\}^W)^T$. For uniformly random $r$:

$$\begin{aligned}
& \mathbf{Pr}_r\big[\exists s\ \forall j\ \exists i:\ \Pi(x; r_j \oplus s_i) \text{ accepts}\big] \\
\leq\ & \textstyle\sum_s \mathbf{Pr}_r\big[\forall j\ \exists i:\ \Pi(x; r_j \oplus s_i) \text{ accepts}\big] && \text{(union bound)} \\
=\ & \textstyle\sum_s \prod_j \mathbf{Pr}_{r_j}\big[\exists i:\ \Pi(x; r_j \oplus s_i) \text{ accepts}\big] && (r_1, \ldots, r_\ell \text{ are independent}) \\
\leq\ & \textstyle\sum_s \prod_j 1/3 && (r_j \text{ is uniformly distributed}) \\
=\ & 2^{kTW}/3^\ell\ <\ 1
\end{aligned}$$

Thus $\exists r\ \forall s\ \big(\exists j\ \forall i:\ \Pi(x; r_j \oplus s_i) \text{ rejects}\big)$.

Assuming $A(x) = 1$, we show that $\exists s\ \forall r:\ V(x;s;r)$ accepts. In the proof of Theorem 7.10, we showed that $\exists s\ \forall q\ \big(\exists i:\ \Pi(x; q \oplus s_i) \text{ accepts}\big)$. Thus $\exists s\ \forall r\ \big(\forall j\ \exists i:\ \Pi(x; r_j \oplus s_i) \text{ accepts}\big)$.

Since $V$ has time efficiency $O(k\ell T)$, we have $A \in \mathsf{S_2TIME}[N^{4d} \log^7 N] \subseteq \mathsf{S_2P}$.

**Exercise 7.8:** The output $\vee$ gate has $N^k$ incoming wires corresponding to tuples of "shift amounts" $s = (s_1, \ldots, s_k) \in \{0, 1, \ldots, N-1\}^k$. The $s$ wire comes from an $\wedge$ gate with $N$ incoming wires corresponding to indices $r \in \{0, 1, \ldots, N-1\}$. The $r$ wire comes from an $\vee$ gate with $k$ incoming wires corresponding to values of $i \in [k]$. The $i$ wire comes from $x_{r+s_i}$, where the subscript is actually $(r + s_i) \bmod N$ but we elide the "mod $N$". This monotone formula has depth 3 and size $N^{\log N + O(1)}$.

Suppose $x$ has $< N/k$ many 1s. For every $s$:

$$\begin{aligned} \mathbf{Pr}_r\left[\exists i : \; x_{r+s_i} = 1\right] &\leq \textstyle\sum_i \mathbf{Pr}_r\left[x_{r+s_i} = 1\right] && \text{(union bound)} \\ &< \textstyle\sum_i 1/k && (r + s_i \text{ is uniformly distributed}) \\ &= k \cdot 1/k \;=\; 1 \end{aligned}$$

Thus $\forall s \; \exists r \; \forall i : \; x_{r+s_i} = 0$, so the formula outputs 0.

Suppose $x$ has $< N/k \leq N/2$ many 0s. For uniformly random $s$:

$$\begin{aligned} \mathbf{Pr}_s\left[\exists r \; \forall i : \; x_{r+s_i} = 0\right] &\leq \textstyle\sum_r \mathbf{Pr}_s\left[\forall i : \; x_{r+s_i} = 0\right] && \text{(union bound)} \\ &= \textstyle\sum_r \prod_i \mathbf{Pr}_{s_i}\left[x_{r+s_i} = 0\right] && (s_1, \ldots, s_k \text{ are independent}) \\ &< \textstyle\sum_r \prod_i 1/2 && (r + s_i \text{ is uniformly distributed}) \\ &= N/2^k \;\leq\; 1 && (k = \lceil \log N \rceil) \end{aligned}$$

Thus $\exists s \; \forall r \; \exists i : \; x_{r+s_i} = 1$, so the formula outputs 1.

**Exercise 7.9:** $\Leftarrow$: If $\mathsf{PH} = \mathsf{ZPP}$ then $\mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{ZPP} \subseteq \mathsf{coRP}$ (Lemma 7.3).

$\Rightarrow$: Assume $\mathsf{NP} \subseteq \mathsf{coRP}$. We have $\mathsf{NP} \subseteq \mathsf{BPP}$ (since $\mathsf{coRP} \subseteq \mathsf{BPP}$), which implies $\mathsf{NP} \subseteq \mathsf{RP}$ (Theorem 7.11) and thus $\mathsf{NP} \subseteq \mathsf{RP} \cap \mathsf{coRP} = \mathsf{ZPP}$ (Lemma 7.3). We also have $\mathsf{NP} \subseteq \mathsf{coNP}$ (since $\mathsf{coRP} \subseteq \mathsf{coNP}$), which implies $\mathsf{PH} \subseteq \mathsf{NP}$ (Theorem 4.23 and Exercise 4.11). In summary, $\mathsf{PH} \subseteq \mathsf{NP} \subseteq \mathsf{ZPP}$, and since $\mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{NP} \subseteq \mathsf{PH}$ holds no matter what, we conclude that $\mathsf{PH} = \mathsf{ZPP}$.

**Exercise 7.10:** Suppose $A \in \mathsf{CL}$ by a catalytic program $\Pi$ with $L$ lines, $R$ registers, and log-bounded word size $W$. Let $M = 2^W \leq \text{poly}N$, so only the first $M$ work segment words are addressable. A configuration is $1+R+M$ words representing the program counter, all $R$ registers, and the work segment's first $M$ words.

Consider the randomized program $\Pi'$ with word size $W$ that copies the randomness segment's first $M$ words to the work segment, and then runs $\Pi$. That is, $\Pi'(x;r) = \Pi(x;r)$ where $\Pi(x;r)$ denotes $\Pi$ with the work segment initialized to $r$. Then $\Pi'$ always outputs the correct bit since $\Pi$ does. We claim that for every valid input $x$, $\mathbf{E}[\text{running time of } \Pi'] \leq \text{poly}N$, which implies $A \in \mathsf{ZPP}$ (Theorem 7.15).

As usual, the computation of $\Pi(x;r)$ doesn't repeat any configuration, since otherwise it wouldn't terminate. The key observation is that if $r \neq s$, then the computations of $\Pi(x;r)$ and $\Pi(x;s)$ have no configuration in common. This is because otherwise, the computations would proceed in the same way after that common configuration and therefore terminate with the same work segment contents as each other, contradicting the fact that $\Pi(x;r)$ terminates with $r$ in the work segment and $\Pi(x;s)$ terminates with $s$ in the work segment. Thus:

$$\begin{aligned}
\textstyle\sum_r\big(\text{running time of } \Pi(x;r)\big) &= \textstyle\sum_r\big(\text{number of configurations of } \Pi(x;r) \text{ minus } 1\big) \\
&\leq \textstyle\sum_r\big(\text{number of configurations of } \Pi(x;r)\big) \\
&\leq \text{total number of possible configurations} \\
&= L2^{(R+M)W}
\end{aligned}$$

Since each of the $2^{MW}$ outcomes $r$ has probability $2^{-MW}$:

$$\begin{aligned}
\mathbf{E}_r\big[\text{running time of } \Pi(x;r)\big] &= 2^{-MW}\textstyle\sum_r\big(\text{running time of } \Pi(x;r)\big) \\
&\leq 2^{-MW} \cdot L2^{(R+M)W} \\
&= L2^{RW} \\
&\leq 2^{O(\log N)} \\
&= \text{poly}N
\end{aligned}$$

**Exercise 7.11:** Let $X_i$ be the indicator random variable for the event that the $i^{\text{th}}$ coin is heads. Let $p_i = \mathbf{E}[X_i]$ be the $i^{\text{th}}$ coin's heads probability. Let the random variable $X = \sum_{i=1}^{n} X_i$ be the number of heads. By linearity of expectation, $\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} p_i$. Since the tosses are independent,

$$\mathbf{Pr}[\text{get all tails}] = \prod_{i=1}^{n}(1-p_i) \leq \prod_{i=1}^{n} \mathrm{e}^{-p_i} = \mathrm{e}^{-\sum_{i=1}^{n} p_i} = \mathrm{e}^{-\mathbf{E}[X]}$$

by Fact 7.6.

**Exercise 7.12:** Define $p_i = \mathbf{E}[X_i]$ for each $i$, so $\mathbf{E}[X] = p_1 + \cdots + p_n$ by linearity of expectation. As in the proof of Lemma 7.20,

$$\mathbf{Pr}\left[X \geq \mathbf{E}[X] + \gamma n\right] = \mathbf{Pr}\left[X - \mathbf{E}[X] \geq \gamma n\right] = \mathbf{Pr}\left[e^{a(X-\mathbf{E}[X])} \geq e^{a\gamma n}\right] \leq \mathbf{E}\left[e^{a(X-\mathbf{E}[X])}\right] / e^{a\gamma n}$$

and

$$\mathbf{E}\left[e^{a(X-\mathbf{E}[X])}\right] = \mathbf{E}\left[e^{a((X_1-p_1)+\cdots+(X_n-p_n))}\right] = \mathbf{E}\left[e^{a(X_1-p_1)} \cdots e^{a(X_n-p_n)}\right] = \prod_{i=1}^{n} \mathbf{E}\left[e^{a(X_i-p_i)}\right]$$

and $\mathbf{E}\left[e^{a(X_i-p_i)}\right] \leq e^{a^2/4}$ for each $i$ because $0 \leq X_i \leq 1$ implies that $X_i^2 \leq X_i$ and thus:

$$\mathbf{Var}[X_i] = \mathbf{E}[X_i^2] - \mathbf{E}[X_i]^2 \leq \mathbf{E}[X_i] - \mathbf{E}[X_i]^2 = p_i - p_i^2 = p_i(1-p_i) \leq 1/4$$

Combining everything and choosing $a = 2\gamma \leq 1$:

$$\mathbf{Pr}\left[X \geq \mathbf{E}[X] + \gamma n\right] \leq e^{(a^2/4)n} / e^{a\gamma n} = e^{-(a\gamma - a^2/4)n} = e^{-\gamma^2 n}$$

$\mathbf{Pr}\left[X \leq \mathbf{E}[X] - \gamma n\right] \leq e^{-\gamma^2 n}$ follows by applying the above to $n - X = (1-X_1) + \cdots + (1-X_n)$.

**Exercise 7.13.a:** Consider an infinite random walk from $v$ (not from $u$). Let the random variable $X$ be the number of steps to return to $v$. By the theorem stated in the exercise, $\mathbf{E}[X] = 2|E|/\deg(v)$. For each neighbor $w$ of $v$, let $A_w$ be the event that the walk's first step goes to $w$, so $\mathbf{Pr}[A_w] = 1/\deg(v)$. By the law of total expectation:

$$\mathbf{E}[X] \;=\; \sum_{\text{edges } \{v,w\}} \mathbf{Pr}[A_w] \cdot \mathbf{E}[X \mid A_w] \;=\; \tfrac{1}{\deg(v)} \sum_{\text{edges } \{v,w\}} \mathbf{E}[X \mid A_w]$$

Combining these, we have $\sum_{\text{edges } \{v,w\}} \mathbf{E}[X \mid A_w] = 2|E|$ and thus $\mathbf{E}[X \mid A_u] \leq 2|E|$. The expectation of the number of steps for an infinite random walk from $u$ to visit $v$ equals $\mathbf{E}[X \mid A_u] - 1 \leq 2|E| - 1$ where the $-1$ is because $X$ counts the first step (from $v$ to $u$) but we just want to count the remaining steps (from $u$ back to $v$).

**Exercise 7.13.b:** Consider a path $s = v_0 - v_1 - v_2 - \cdots - v_\ell = t$ of length $\ell < n$. Partition an infinite random walk from $s$ into $\ell + 1$ phases: For $i \in \{1, \ldots, \ell\}$, the $i^{\text{th}}$ phase is the steps to reach $v_i$ after the $(i-1)^{\text{st}}$ phase (or from the beginning if $i = 1$), and the $(\ell+1)^{\text{st}}$ phase is the rest of the walk. For $i \in \{1, \ldots, \ell\}$, let the random variable $X_i$ be the number of steps in the $i^{\text{th}}$ phase. Let $X = X_1 + \cdots + X_\ell$. By linearity of expectation and Exercise 7.13.a:

$$\mathbf{E}[\text{steps to visit } t] \leq \mathbf{E}[X] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_\ell] \leq \ell(2|E| - 1) < 2n|E|$$

By Lemma 7.13:

$$\mathbf{Pr}\big[\text{don't visit } t \text{ within first } 6n|E| \text{ steps}\big] \leq \mathbf{E}[\text{steps to visit } t]/6n|E| < 1/3$$

**Exercise 7.14.a:** Let $a = \sum_{i \in U} \text{indeg}(i)$ and $b = \sum_{i \in U} \text{outdeg}(i)$ and $c = a - b$. For every edge $(x, y) \in E$:

- If $x \in U$ and $y \in U$ then $(x, y)$ contributes 1 to $a$ and 1 to $b$ and thus 0 to $c$.
- If $x \in W$ and $y \in W$ then $(x, y)$ contributes 0 to $a$ and 0 to $b$ and thus 0 to $c$.
- If $x \in W$ and $y \in U$ then $(x, y)$ contributes 1 to $a$ and 0 to $b$ and thus 1 to $c$.
- If $x \in U$ and $y \in W$ then $(x, y)$ contributes 0 to $a$ and 1 to $b$ and thus $-1$ to $c$.

**Exercise 7.14.b:** We give a contrapositive proof. Consider any balanced directed graph $G = (V, E)$. Suppose $G$ is not strongly connected, so some node $k$ is unreachable from some node $h$. Let $U$ be the set of all nodes reachable from $h$, and let $W = V \setminus U$. Then $U$ and $W$ partition $V$ and are nonempty since $h \in U$ and $k \in W$. We must have $E \cap (U \times W) = \emptyset$, so $E \cap (W \times U) = \emptyset$ because

$$|E \cap (W \times U)| - |E \cap (U \times W)| \; = \; \sum_{i \in U} \text{indeg}(i) - \sum_{i \in U} \text{outdeg}(i) \; = \; 0$$

by Exercise 7.14.a and the balanced property of $G$. In the undirected version of $G$, there's no edge between $U$ and $W$, so $k$ is unreachable from $h$. Therefore $G$ is not weakly connected.

**Exercise 7.14.c:** Map $(G, s, t)$ to $(H, s, t)$ where $H$ is $G$ but ignoring edge directions and retaining only one copy of any resulting multi-edges. To show this log-space mapping reduction is correct, we argue that $G$ has a path from $s$ to $t$ iff $H$ has a path from $s$ to $t$:

$\Rightarrow$: If $G$ has a path from $s$ to $t$, then the corresponding undirected edges in $H$ are a path from $s$ to $t$.

$\Leftarrow$: Suppose $s$ and $t$ are in the same connected component of $H$. This component is weakly connected in $G$ and therefore strongly connected since $G$ is balanced (Exercise 7.14.b). Thus $G$ has a directed path from $s$ to $t$.

Balanced Directed Reachability $\leq_\ell^m$ Undirected Reachability $\in$ RL (Theorem 7.24), so Balanced Directed Reachability $\in$ RL by running a log-space randomized program for Undirected Reachability on the query $(H, s, t)$ and using the reduction to (re)compute each word of the query whenever it's needed, as in Lemma 1.15.

**Exercise 7.14.d:** Like in Lemma 7.31, it suffices to show $\sum_{\text{edges } (i,j)}(x_i - y_j)^2 \geq \|x\|^2/2n^2$ where $y = xT$, and we know there exist nodes $h$ and $k$ such that $x_h - x_k \geq \|x\|/\sqrt{n}$. Since $G'$ is strongly connected, there exists a path from $h$ to $k$. Renumbering the nodes, we assume $h = 1$ and the path is:

$$1 \to 2 \to 3 \to \cdots \to k$$

Since each node has at least one self-loop, we can also consider self-loops on nodes $2, \ldots, k$:

$$1 \to \underset{\circlearrowleft}{2} \to \underset{\circlearrowleft}{3} \to \cdots \to \underset{\circlearrowleft}{k}$$

Let $K = 2(k-1) \leq 2n$ be the number of edges on this walk. The contribution of these edges to $\sum_{\text{edges } (i,j)}(x_i - y_j)^2$ is $\sum_{i=2}^{k}\big((x_{i-1} - y_i)^2 + (x_i - y_i)^2\big)$.

$$\begin{aligned}
&\textstyle\sum_{\text{edges } (i,j)}(x_i - y_j)^2 \\
&\geq (x_1 - y_2)^2 + (y_2 - x_2)^2 + \cdots + (x_{k-1} - y_k)^2 + (y_k - x_k)^2 \\
&\geq \tfrac{1}{K}\big(|x_1 - y_2| + |y_2 - x_2| + \cdots + |x_{k-1} - y_k| + |y_k - x_k|\big)^2 && \text{(Corollary 0.27)} \\
&\geq \tfrac{1}{K}\big|(x_1 - y_2) + (y_2 - x_2) + \cdots + (x_{k-1} - y_k) + (y_k - x_k)\big|^2 && \text{(Lemma 0.28)} \\
&= \tfrac{1}{K}(x_1 - x_k)^2 \\
&\geq \tfrac{1}{K}(\|x\|/\sqrt{n})^2 \\
&\geq \|x\|^2/2n^2
\end{aligned}$$

Corollary 0.27 and Lemma 0.28 used the vector $(x_1 - y_2,\ y_2 - x_2,\ \ldots,\ x_{k-1} - y_k,\ y_k - x_k)$.

**Exercise 7.14.e:** Consider any weakly connected balanced directed $n$-node graph $G$ and any nodes $s \neq t$. By Exercise 7.14.b, $G$ is strongly connected. Define $G'$ from $G$ by adding enough self-loops so each node has outdegree $n$ and indegree $n$ (which is possible since $G$ is balanced and has no multi-edges or self-loops). In particular, each node has at least one self-loop in $G'$.

We prove that in $G'$, a $(12n^4 \log n)$-step random walk from $s$ visits $t$ with probability $\geq 2/3$. This implies the same for the non-lazy random walk in $G$. As in the proof of Lemma 7.27:

- Exercise 7.14.d implies that $\Delta(vT) \leq e^{-1/2n^3}\Delta(v)$ for every distribution vector $v$.
- Letting $\ell = 4n^3 \log n$, this implies that for every node $q$ in $G'$, an $\ell$-step random walk from $q$ ends at $t$ with probability $\geq 1/2n$.
- This implies that a $3n\ell$-step random walk from $s$ visits $t$ with probability $\geq 2/3$.

**Exercise 7.15:** BPL $\subseteq$ BP*L because we can create a new register *raddr* and substitute:

read-rand Reg[$i$] $\rightsquigarrow$ read-rand Reg[$i$] $\leftarrow$ Rand[*raddr*]
let *raddr* $\leftarrow$ *raddr* + 1

We may also need to increase the word size so *raddr* doesn't wrap around. Since the original log-space randomized program is poly-time, we only need $O(\log N)$ word size for *raddr*.

BP*L $\subseteq$ L/poly by essentially the same proof as BPP $\subseteq$ P/poly (Theorem 7.9): Suppose $A \in$ BP*L and $A$ has word size $W = O(\log N)$, so inputs of size $N$ are in $(\{0,1\}^W)^N$. We amplify such a program to have error probability $< 2^{-NW}$ by running it $O(NW)$ times, with fresh randomness each time, and accepting iff a majority of the runs accept. If the original program has time efficiency $\leq T = cN^d$ (for integers $c, d$), then the new program changes each "read-rand Reg[$i$] $\leftarrow$ Rand[Reg[$j$]]" instruction to read from address $(k-1) \cdot T +$ Reg[$j$] during the $k^{\text{th}}$ run of the original program. This ensures the runs are independent of each other. The new program $\Pi$ is still log-space. By a union bound, there exists an outcome $r$ such that $\Pi(x; r) = A(x)$ for every valid $x$ of size $N$. Treating this $r$ as advice for input size $N$ (and adjusting $\Pi$ to read $r$ from the advice segment), we obtain a log-space nonuniform program for $A$, so $A \in$ L/poly.

**Exercise 7.16:** This is similar to the proof that NP $\subseteq$ BPP iff NP = RP (Theorem 7.11).

$\Leftarrow$: If NL = RL then NL $\subseteq$ RL $\subseteq$ BPL.

$\Rightarrow$: Assume NL $\subseteq$ BPL. To show NL = RL, it suffices to show Directed Reachability $\in$ RL since Directed Reachability is NL-complete (Theorem 3.13). We have Directed Reachability Search $\leq_\ell^o$ Directed Reachability (Theorem 1.22) and Directed Reachability $\in$ BPL, so Directed Reachability Search has a log-space randomized program that outputs a path from $s$ to $t$ with probability $\geq 2/3$ if one exists (by combining the proofs of Lemma 1.18 and Lemma 7.8). Consider this log-space randomized program for Directed Reachability on input $(G, s, t)$:

run the randomized program for Directed Reachability Search on input $(G, s, t)$
if it output a walk of length $\leq n-1$ from $s$ to $t$: accept
else: reject

If $t$ is reachable from $s$, this program accepts with probability $\geq 2/3$. If $t$ is unreachable from $s$, this program accepts with probability 0.

**Exercise 7.17:** Assume the classes only contain total problems. Assume $\mathsf{P} = \mathsf{NP}$. Then $\mathsf{P} = \Sigma_2\mathsf{P}$, as explained in § 7.9.2. Since $\mathsf{BPP} \subseteq \Sigma_2\mathsf{P}$ (Theorem 7.10), we have $\mathsf{BPP} \subseteq \mathsf{P}$ and of course $\mathsf{P} \subseteq \mathsf{BPP}$.

**Exercise 7.18.a:** $A \in \mathsf{BPP}$ by a randomized program $\Pi$ where $\Pi(x; w_1, w_2)$ runs $V(x; w_1; w_2)$:

$$\begin{aligned} A(x) = 1 \;&\Rightarrow\; \mathbf{Pr}_{w_1}\Big[\mathbf{Pr}_{w_2}\big[V(x; w_1; w_2) \text{ accepts}\big] = 1\Big] \geq 2/3 \\ &\Rightarrow\; \mathbf{Pr}_{w_1, w_2}\big[\Pi(x; w_1, w_2) \text{ accepts}\big] \geq 2/3 \end{aligned}$$

$$\begin{aligned} A(x) = 0 \;&\Rightarrow\; \mathbf{Pr}_{w_1}\Big[\mathbf{Pr}_{w_2}\big[V(x; w_1; w_2) \text{ rejects}\big] \geq 2/3\Big] = 1 \\ &\Rightarrow\; \mathbf{Pr}_{w_1, w_2}\big[\Pi(x; w_1, w_2) \text{ rejects}\big] \geq 2/3 \end{aligned}$$

**Exercise 7.18.b:** We prove that the third level collapses to BPP. The same argument can be repeated to prove that each higher level also collapses. Assume $A$ has a poly-time 3-witness verifier $V$ such that:

$$A(x)=1 \Rightarrow \left(\widehat{\exists} w_1\ \forall w_2\ \widehat{\exists} w_3 :\ V(x;w_1;w_2;w_3) \text{ accepts}\right)$$
$$A(x)=0 \Rightarrow \left(\forall w_1\ \widehat{\exists} w_2\ \forall w_3 :\ V(x;w_1;w_2;w_3) \text{ rejects }\right)$$

This promise problem captures the final two "quantifiers":

(call this problem $B$)

Input: Valid input $x$ to $A$, and $w_1$ such that either:
$\left(\forall w_2\ \widehat{\exists} w_3 :\ V(x;w_1;w_2;w_3) \text{ accepts}\right)$ or $\left(\widehat{\exists} w_2\ \forall w_3 :\ V(x;w_1;w_2;w_3) \text{ rejects}\right)$

Output: Is it true that $\forall w_2\ \widehat{\exists} w_3 :\ V(x;w_1;w_2;w_3)$ accepts?

$B$ is in the complement of the second level, by a 2-witness verifier $V'$ where $V'(x,w_1;w_2;w_3)$ runs $V(x;w_1;w_2;w_3)$. By Exercise 7.18.a and amplification, $B \in$ BPP by a $1/6$-error randomized program $\Pi$. Then $A \in$ BPP by a randomized program $\Pi'$ where $\Pi'(x;w_1,r)$ runs $\Pi(x,w_1;r)$:

$$\begin{aligned} A(x)=1 &\Rightarrow \widehat{\exists} w_1 :\ B(x,w_1)=1 \\ &\Rightarrow \mathbf{Pr}_{w_1}\left[\mathbf{Pr}_r\left[\Pi(x,w_1;r) \text{ accepts}\right] \geq 5/6\right] \geq 2/3 \\ &\Rightarrow \mathbf{Pr}_{w_1,r}\left[\Pi'(x;w_1,r) \text{ accepts}\right] \geq 5/9 \end{aligned}$$

$$\begin{aligned} A(x)=0 &\Rightarrow \forall w_1 :\ B(x,w_1)=0 \\ &\Rightarrow \mathbf{Pr}_{w_1}\left[\mathbf{Pr}_r\left[\Pi(x,w_1;r) \text{ rejects}\right] \geq 5/6\right] = 1 \\ &\Rightarrow \mathbf{Pr}_{w_1,r}\left[\Pi'(x;w_1,r) \text{ rejects}\right] \geq 5/6 \end{aligned}$$

By amplification, the error probability can be decreased from $4/9$ to $1/3$.

**Exercise 7.19:** $\Leftarrow$: If BPP = ZPP then RP $\subseteq$ BPP $\subseteq$ ZPP $\subseteq$ coRP (Lemma 7.3) and similarly coRP $\subseteq$ RP.

$\Rightarrow$: Assume RP = coRP. Since ZPP $\subseteq$ BPP, it suffices to prove BPP $\subseteq$ ZPP. Since ZPP = RP $\cap$ coRP (Lemma 7.3) and BPP is closed under complement, it suffices to prove BPP $\subseteq$ RP.

Suppose $A \in$ BPP. As noted in §7.9.2, the proof of Theorem 7.10 yields a poly-time 2-witness verifier $V(x;s;r)$ such that:

$$A(x)=1 \Rightarrow \left(\widehat{\exists} s\ \forall r:\ V(x;s;r) \text{ accepts}\right)$$
$$A(x)=0 \Rightarrow \left(\forall s\ \widehat{\exists} r:\ V(x;s;r) \text{ rejects}\ \right)$$

(call this problem $B$)

Input: Valid input $x$ to $A$, and $s$ such that either:
$\left(\forall r:\ V(x;s;r) \text{ accepts}\right)$ or $\left(\widehat{\exists} r:\ V(x;s;r) \text{ rejects}\right)$

Output: Is it true that $\forall r:\ V(x;s;r)$ accepts?

$B \in$ coRP by a randomized program $\Pi$ where $\Pi(x,s;r)$ runs $V(x;s;r)$. Thus $B \in$ RP by a randomized program $\Pi'$. Assume $\Pi'$ clocks itself so it terminates in poly time even on invalid inputs. Then $A \in$ RP by a randomized program $\Pi''$ where $\Pi''(x;s,q)$ runs $\Pi'(x,s;q)$:

$$\begin{aligned} A(x)=1 &\Rightarrow \widehat{\exists} s:\ B(x,s)=1 \\ &\Rightarrow \mathbf{Pr}_s\left[\mathbf{Pr}_q\left[\Pi'(x,s;q) \text{ accepts}\right] \geq 2/3\right] \geq 2/3 \\ &\Rightarrow \mathbf{Pr}_{s,q}\left[\Pi''(x;s,q) \text{ accepts}\right] \geq 4/9 \end{aligned}$$

$$\begin{aligned} A(x)=0 &\Rightarrow \forall s:\ B(x,s)=0 \\ &\Rightarrow \mathbf{Pr}_s\left[\mathbf{Pr}_q\left[\Pi'(x,s;q) \text{ rejects}\right]=1\right]=1 \\ &\Rightarrow \mathbf{Pr}_{s,q}\left[\Pi''(x;s,q) \text{ rejects}\right]=1 \end{aligned}$$

By amplification, the error probability can be decreased from 5/9 to 1/3.

**Exercise 7.20.a:** Assume $\mathsf{P} = \mathsf{BPP}$. Since Circuit Gap Majority $\in \mathsf{BPP} \subseteq \mathsf{P}$, Circuit Gap Majority $\in \mathsf{TIME}[N^c]$ for some $c$. For every $A \in \mathsf{BPTIME}[N]$, we have $A \leq_p^m$ Circuit Gap Majority via a quasi-quadratic-time reduction, because the proof that Circuit Gap Majority is BPP-complete (Lemma 7.39) uses the quasi-quadratic-size circuits from Theorem 5.11. Combining the reduction with the program for Circuit Gap Majority, we have $\mathsf{BPTIME}[N] \subseteq \mathsf{TIME}[(N^2 \operatorname{polylog} N)^c] \subseteq \mathsf{TIME}[N^{2c+1}]$. So we can take $d = 2c + 1$.

**Exercise 7.20.b:** Assume $\mathsf{P} = \mathsf{BPP}$ for promise problems. Let $d$ be such that $\mathsf{BPTIME}[N] \subseteq \mathsf{TIME}[N^d]$ (Exercise 7.20.a). By the deterministic time hierarchy (Theorem 4.7, which holds for total problems), $\mathsf{TIME}[N^{d+1}] \not\subseteq \mathsf{TIME}[N^d]$ and thus $\mathsf{BPTIME}[N^{d+1}] \not\subseteq \mathsf{BPTIME}[N]$.

The rest of the proof mirrors Theorem 7.42. Suppose for contradiction $\mathsf{BPTIME}[N^a] \subseteq \mathsf{BPTIME}[N]$ for some $a > 1$. We may assume $a$ is rational. By padding, $\mathsf{BPTIME}[N^{a^{b+1}}] \subseteq \mathsf{BPTIME}[n^{a^b}]$ for every integer $b \geq 0$:

$$\begin{array}{ccc} A \in \mathsf{BPTIME}[N^{a^{b+1}}] & & A \in \mathsf{BPTIME}[N^{a^b}] \\ \Downarrow & & \Uparrow \\ A_{\text{pad}} \in \mathsf{BPTIME}[M^a] & \Rightarrow & A_{\text{pad}} \in \mathsf{BPTIME}[M] \end{array}$$

where $M = \text{pad}(N) = \lceil N^{a^b} \rceil$. Let $k$ be an integer such that $a^k \geq d+1$. Then

$$\mathsf{BPTIME}[N^{d+1}] \subseteq \mathsf{BPTIME}[N^{a^k}] \subseteq \mathsf{BPTIME}[N^{a^{k-1}}] \subseteq \cdots \subseteq \mathsf{BPTIME}[N^a] \subseteq \mathsf{BPTIME}[N]$$

which is a contradiction.

**Exercise 7.21:** Like Verifier Interpretation with Witness $\in$ TIME$[N]$ (Lemma 4.14), we have Interpretation with Randomness $\in$ TIME$[N]$: An interpreter for randomized programs (where the code $I$ has "read-rand" instructions) just needs an extra register to remember which address of $r$ will be read next.

Interpretation with Randomness
Input: Tight code $I$, $W \leq \log T$, $x \in (\{0,1\}^W)^M$, $r \in (\{0,1\}^W)^T$
Output: Does $I_W(x;r)$ accept within $T$ steps?

Let $c$ and $d$ be rational numbers with $a \geq c > d \geq b$. We show that the following promise problem $B$ is in BPTIME$[N^c] \subseteq$ BPTIME$[N^a]$ but not in BPTIME$[N^d] \supseteq$ BPTIME$[N^b]$.

(call this problem $B$, with input size $N$)
Input: Tight code $I$ of size $L$, $e$ such that $10 \leq e \leq (c-d)\log L$, padding of size $J$
Output: Let $K = 2^{L^{c+1}}$. Let $J' = J+1$ if $J < K$, and $J' = 0$ otherwise.
Let $x = (I, e, \text{padding of size } J')$ of size $M$. Let $T = M^c$ and $W = d\log M + e$.
Let $p = \mathbf{Pr}\big[I_W(x;r) \text{ accepts within } T \text{ steps}\big]$ over uniformly random $r \in (\{0,1\}^W)^T$.

If $J < K$: output is $\begin{cases} 1 & \text{if } p \geq 2/3 \\ 0 & \text{if } p \leq 1/3 \\ \text{undefined} & \text{if } 1/3 < p < 2/3 \end{cases}$

Else: output is $\begin{cases} 1 & \text{if } p \leq 1/2 \\ 0 & \text{if } p > 1/2 \end{cases}$

```
compute K, x, T, W as in B's definition
if J < K:
    sample a uniformly random r ∈ ({0,1}^W)^T
    run the program for Interpretation with Randomness on input (I, W, x, r)
    output the same thing
else:
    for each r ∈ ({0,1}^W)^T:
        run the program for Interpretation with Randomness on input (I, W, x, r)
    if it accepted for at most half of all r: accept
    else: reject
```

$B \in$ BPTIME$[N^c]$: The above randomized algorithm is correct by definition. Computing $T$ and $W$ takes time $O(\log N)$ since $c$ and $d$ are rational. The $J < K$ case takes time $O(N^c)$ since the input to Interpretation with Randomness has size $4 + L + M + T = O(M^c) = O(N^c)$. For the $J \geq K$ case, there are $2^{TW}$ iterations, each of which takes time $O(L^c)$ since the input to Interpretation with Randomness has size $4+L+M+T = O(M^c) = O(L^c)$, so this case takes time $2^{TW} \cdot O(L^c) = 2^{O(L^c \log L)} \cdot O(L^c) = O(K) = O(N) = O(N^c)$. In both cases, $W$ is valid as part of the input since $e \leq (c-d)\log L \leq (c-d)\log M$ implies $W = d\log M + e \leq c\log M = \log T$.

Suppose for contradiction $B \in \mathsf{BPTIME}[N^d]$. Like in Claim 4.8, some 1/3-error randomized program $\Pi$ with tight code $I$ solves $B$ in time $O(N^d \log^4 N)$ with word size $d \log N + e$ for some constant $e \geq 10$. Consider the sequence of inputs $x_0, x_1, \ldots, x_K$ where $x_J = (I, e, \text{padding of size } J)$. Since $d < c$, we can ensure $L$ is large enough (by appending dummy instructions to $I$) that $e \leq (c-d)\log L$ and $\Pi$ runs in time $N^c$ on input $x_J$ if $x_J$ is a valid input to $B$. Write "$B(x_J) = ?$" if $x_J$ is an invalid input to $B$. Also, abusing notation, define

$$\Pi(x_J) = \begin{cases} 1 & \text{if } p \geq 2/3 \\ 0 & \text{if } p \leq 1/3 \\ ? & \text{if } 1/3 < p < 2/3 \end{cases}$$

where $p = \mathbf{Pr}\big[\Pi(x_J; r) \text{ accepts within } N^c \text{ steps}\big]$ over uniformly random $r$.

- If $B(x_J) \in \{0,1\}$ then $\Pi(x_J) = B(x_J)$ since $\Pi$ is correct and runs in time $N^c$ on input $x_J$.
- $B(x_J) = \Pi(x_{J+1})$ for each $J < K$ and $B(x_K) \neq \Pi(x_0)$ and $B(x_K) \in \{0,1\}$ by $B$'s definition.

Contradiction: $\Pi(x_0) \neq B(x_K) = \Pi(x_K) = B(x_{K-1}) = \Pi(x_{K-1}) = B(x_{K-2}) = \cdots = B(x_0) = \Pi(x_0)$.

**Exercise 8.1.a:** Let the random variable $X$ be the number of unordered pairs of elements of $S$ that collide: $s \neq s'$ such that $h_r(s) = h_r(s')$. As in the proof of Lemma 8.3, $\mathbf{E}[X] \leq |S|^2/2|T|$. By Lemma 7.13, $\mathbf{Pr}\left[X \geq 2|S|^2/|T|\right] \leq 1/4$. Define $\ell = 3|S|/\sqrt{|T|}$ and note that $\ell \leq \ell^2/2$ by assumption. If $X < 2|S|^2/|T|$ (which happens with probability $\geq 3/4$) then every chain has length $< \ell$, because a chain of length $\geq \ell$ would contribute $\geq \binom{\lceil \ell \rceil}{2} \geq (\ell^2 - \ell)/2 \geq \ell^2/4 = 2.25|S|^2/|T|$ to $X$.

**Exercise 8.1.b:** For each pair $i > j$, let $X_{i,j}$ be the indicator random variable for the event that $x_i$ and $x_j$ get compared. If $x_i \neq x_j$ then:

$$\mathbf{E}[X_{i,j}] \;=\; \mathbf{Pr}[x_i \text{ and } x_j \text{ get compared}] \;\leq\; \mathbf{Pr}\big[h_r(x_i) = h_r(x_j)\big] \;\leq\; 1/|T| \;\leq\; 1/N$$

With probability 1, we have $\sum_{i>j\,:\,x_i=x_j} X_{i,j} \leq 1$ because the algorithm halts as soon as it compares two equal elements. Let $X = \sum_{i>j} X_{i,j}$ be the number of comparisons. By linearity of expectation:

$$\mathbf{E}[X] \;=\; \textstyle\sum_{i>j\,:\,x_i\neq x_j} \mathbf{E}[X_{i,j}] + \mathbf{E}\big[\sum_{i>j\,:\,x_i=x_j} X_{i,j}\big] \;\leq\; \binom{N}{2}/N + 1 \;=\; (N+1)/2$$

**Exercise 8.2:** $h_r$ is a polynomial of degree $\leq k-1$. We know that if $r \neq r'$ then $h_r(u) = h_{r'}(u)$ holds for at most $k-1$ many $u \in \mathbb{Z}_p$ (Corollary 0.21).

Consider any tuple of $k$ distinct elements $(u_1, \ldots, u_k) \in U^k$. Abusing notation, denote the tuple $\big(h_r(u_1), \ldots, h_r(u_k)\big)$ by $h_r(u_1, \ldots, u_k)$. By the previous paragraph, if $r \neq r'$ then $h_r(u_1, \ldots, u_k) \neq h_{r'}(u_1, \ldots, u_k)$. There are $p^k$ possibilities of $r \in \mathbb{Z}_p^k$ and $p^k$ possibilities of $(t_1, \ldots, t_k) \in \mathbb{Z}_p^k$. We use the pigeonhole principle (third bullet of Fact 0.2): Pigeon $r$ flies into hole $h_r(u_1, \ldots, u_k)$, and no hole gets more than one pigeon, and the number of pigeons equals the number of holes, so every hole gets exactly one pigeon: $\mathbf{Pr}_r\big[h_r(u_1, \ldots, u_k) = (t_1, \ldots, t_k)\big] = p^{-k}$ for every $(t_1, \ldots, t_k)$.

**Exercise 8.3.a:** This is true. If $u \neq u'$ then $\mathbf{Pr}_A[Au = Au'] = \mathbf{Pr}_A\left[A(u-u') = 0^m\right] = 2^{-m}$ since $u-u' \neq 0^n$ and thus $A(u-u')$ is uniformly distributed (Lemma 8.6).

**Exercise 8.3.b:** This is true. If $u \neq u'$ then $\mathbf{Pr}_A[Au = Au'] = \mathbf{Pr}_A\left[A(u-u') = 0^m\right] = 2^{-m}$ since $u-u' \neq 0^n$ and thus $A(u-u')$ is uniformly distributed (Lemma 8.8).

**Exercise 8.3.c:** This is false. Let $u = 1000\cdots0$ and $u' = 0100\cdots0$ (using $n \geq 2$). Then $(Au, Au')$ is the first two columns of $A$, which are not independent since $(Au)_1 = (Au')_2$ (using $m \geq 2$).

**Exercise 8.4.a:** $\Leftarrow$: If $w$ is plain, then $w$ has size $cN^d$ since $V$ never reads beyond the first $cN^d$ words of $w$. If $w$ has size $cN^d$ and $V(x;w)$ accepts, then $w$ is plain since $V(x;w)$ must read all the first $cN^d$ words of $w$ (otherwise we could modify one of the first $cN^d$ words to obtain $w' \neq w$ of size $cN^d$ such that $V(x;w')$ also accepts).

Such a $V$ is already unambiguous, and thus $A \in \mathsf{UP}$:

- Assume $A(x) = 1$, so $V(x;w)$ accepts for exactly one $w$ of size $cN^d$. Then this $w$ is plain, and $V(x;w')$ doesn't accept for any other plain $w'$, since such a $w'$ would have size $cN^d$. Thus $V(x;w)$ accepts for exactly one plain $w$.
- Assume $A(x) = 0$, so $V(x;w)$ accepts for no $w$ of size $cN^d$. Then $V(x;w)$ accepts for no $w$ since $V$ never reads beyond the first $cN^d$ words of $w$.

$\Rightarrow$: Suppose $A \in \mathsf{UP}$ by an unambiguous poly-time verifier $V$. There exist integers $c, d$ such that $V$ has time efficiency $\leq cN^d$ and thus never reads beyond the first $cN^d$ words of $w$. Define a poly-time verifier $V'$:

```
V'(x;w) :
    run V(x;w) and remember which words of w it reads
    if it rejected: reject
    read all of the first cN^d words of w
    if at least one of these words is nonzero and wasn't read by V(x;w): reject
    accept
```

By definition, $V'$ never reads beyond the first $cN^d$ words of $w$.

- Assume $A(x) = 1$, so $V(x;w)$ accepts for exactly one plain $w$. This $w$ has size $cN^d$ and $V'(x;w)$ accepts, and $V'(x;w')$ doesn't accept for any other $w'$ of size $cN^d$ since otherwise $w'$ would be plain (with respect to $V$ and $x$) and $V(x;w')$ would accept. Thus $V'(x;w)$ accepts for exactly one $w$ of size $cN^d$.
- Assume $A(x) = 0$, so $V(x;w)$ accepts for no $w$. Then $V'(x;w)$ accepts for no $w$, in particular for no $w$ of size $cN^d$.

**Exercise 8.4.b:** We already observed Unambiguous Sat $\in$ UP (§ 8.3.1). For any $A \in$ UP, we show $A \leq_p^m$ Unambiguous Sat. Let $V$ be a poly-time unambiguous verifier for $A$ as in Exercise 8.4.a. In the proof that Sat is NP-complete (Theorem 2.10), we designed a poly-time mapping reduction from $A$ to Sat that maps $x$ to a CNF $\psi(w, y)$ (where $w$ represents a purported witness of size $cN^d$) such that for every assignment to $w, y$:

$$\psi(w, y) = 1 \iff \big(V(x; w) \text{ accepts and } y \text{ is the transcript of } V(x; w)\big)$$

This is already a mapping reduction from $A$ to Unambiguous Sat: If $A(x) = 0$ then $V(x; w)$ rejects for all $w$, so $\psi(w, y) = 0$ for all $w, y$. If $A(x) = 1$ then $V(x; w)$ accepts for exactly one $w$, so $\psi(w, y) = 1$ for exactly one $w, y$, namely the $w$ such that $V(x; w)$ accepts and the transcript $y$ of $V(x; w)$.

**Exercise 8.5:** Assume UP $\subseteq$ RP. Suppose $A \in$ NP by a verifier $V$ with time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$. On a valid input $x$ of size $N$, the potential witnesses are elements of $(\{0,1\}^W)^T = \{0,1\}^n$ where $n = TW$. For each $m$, let $h^m : \{0,1\}^{2m+n-1} \times \{0,1\}^n \to \{0,1\}^m$ be the pairwise avoiding hash function from Lemma 8.9. (Or, we could use Lemma 8.7.)

(call this problem $B$)

Input: $x, m, r, t$ where $x$ is a valid input of size $N$ to $A$, $m \in \{4, 5, \ldots, n+4\}$, $r \in \{0,1\}^{2m+n-1}$, and $t \in \{0,1\}^m$

Output: 1 if there's exactly one $w \in \{0,1\}^n$ such that $\big(V(x;w)$ accepts and $h_r^m(w) = t\big)$
0 if there's no such $w$
undefined otherwise

$B \in$ UP by an unambiguous verifier $V'$ where $V'(x, m, r, t; w)$ reads $w$'s first $n$ bits and accepts iff $V(x;w)$ accepts and $h_r^m(w) = t$. Thus $B \in$ RP by a randomized program $\Pi$. Assume $\Pi$ clocks itself so it terminates in poly time even on invalid inputs.

Consider a poly-time randomized program $\Pi'$ where $\Pi'(x)$ samples uniformly random $m \in \{4, \ldots, n+4\}$, $r \in \{0,1\}^{2m+n-1}$, and $t \in \{0,1\}^m$, and runs $\Pi(x, m, r, t)$. This $\Pi'$ runs in poly time even when it runs $\Pi$ on an invalid input to $B$.

Suppose $A(x) = 0$, so $V(x;w)$ accepts for no $w$. For all $m, r, t$, we have $B(x, m, r, t) = 0$ and so $\mathbf{Pr}\big[\Pi(x, m, r, t) \text{ accepts}\big] = 0$. Thus $\mathbf{Pr}\big[\Pi'(x) \text{ accepts}\big] = 0$.

Suppose $A(x) = 1$, so $V(x;w)$ accepts for some $w$. Let $S = \big\{w \in \{0,1\}^n : V(x;w) \text{ accepts}\big\}$. Since $1 \leq |S| \leq 2^n$, we have $16|S| \leq 2^\ell < 32|S|$ for some $\ell \in \{4, \ldots, n+4\}$. Say $(m, r)$ is *good* iff at least $3/4$ of the elements of $S$ do not collide with any element of $S$ under $h_r^m$. By Lemma 8.3, $\mathbf{Pr}_r\big[(\ell, r) \text{ is good}\big] \geq 3/4$ since $16|S| \leq 2^\ell$. Since $2^\ell < 32|S|$, we have:

$$\mathbf{Pr}_{r,t}\big[B(x, \ell, r, t) = 1 \,\big|\, (\ell, r) \text{ is good}\big] \;\geq\; (3/4)|S|/32|S| \;=\; 3/128$$

Combining everything using the chain rule:

$$\begin{aligned}
& \mathbf{Pr}\big[\Pi'(x) \text{ accepts}\big] \\
= \; & \mathbf{Pr}_{m,r,t,q}\big[\Pi(x, m, r, t; q) \text{ accepts}\big] \\
\geq \; & \mathbf{Pr}_{m,r,t,q}\big[m = \ell \text{ and } (m, r) \text{ is good and } B(x, m, r, t) = 1 \text{ and } \Pi(x, m, r, t; q) \text{ accepts}\big] \\
= \; & \mathbf{Pr}_m[m = \ell] \cdot \\
& \mathbf{Pr}_r\big[(\ell, r) \text{ is good}\big] \cdot \\
& \mathbf{Pr}_{r,t}\big[B(x, \ell, r, t) = 1 \,\big|\, (\ell, r) \text{ is good}\big] \cdot \\
& \mathbf{Pr}_{r,t,q}\big[\Pi(x, \ell, r, t; q) \text{ accepts} \,\big|\, (\ell, r) \text{ is good and } B(x, \ell, r, t) = 1\big] \\
\geq \; & \tfrac{1}{n+1} \cdot \tfrac{3}{4} \cdot \tfrac{3}{128} \cdot \tfrac{2}{3} \\
= \; & \Omega(1/n)
\end{aligned}$$

By amplification, $A \in$ RP.

**Exercise 8.6:** First proof: $\mathsf{UP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{UP} \subseteq \mathsf{RP} \Rightarrow \mathsf{NP} \subseteq \mathsf{RP}$ (Theorem 8.10). We can prove $\mathsf{UP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{UP} \subseteq \mathsf{RP}$ like $\mathsf{NP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{NP} \subseteq \mathsf{RP}$ (Theorem 7.11), using either UP-completeness of Unambiguous Sat (Exercise 8.4.b) and a search-to-decision reduction for Unambiguous Sat similar to Theorem 2.14, or search-to-decision reductions for arbitrary UP problems like in Exercise 2.15.

Second proof: $\mathsf{UP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{NP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{NP} \subseteq \mathsf{RP}$ (Theorem 7.11). We can prove $\mathsf{UP} \subseteq \mathsf{BPP} \Rightarrow \mathsf{NP} \subseteq \mathsf{BPP}$ like $\mathsf{UP} \subseteq \mathsf{RP} \Rightarrow \mathsf{NP} \subseteq \mathsf{RP}$ (Theorem 8.10), using a two-sided $0.01/n$-error randomized program $\Pi$ for $B \in \mathsf{BPP}$:

$$A(x) = 0 \Rightarrow \mathbf{Pr}\big[\Pi'(x) \text{ accepts}\big] \leq 0.01/n$$
$$A(x) = 1 \Rightarrow \mathbf{Pr}\big[\Pi'(x) \text{ accepts}\big] \geq 0.2/(n+1)$$

This implies $A \in \mathsf{BPP}$, like Corollary 7.22 (which was only stated for constant probabilities).

**Exercise 8.7.a:** Define $p_x = \mathbf{Pr}[X = x]$. Consider vectors $v$ and $w$ indexed by all $x > 0$ with $p_x > 0$, and defined by $v_x = x\sqrt{p_x}$ and $w_x = \sqrt{p_x}$. By Theorem 0.26, and with summations over all $x > 0$ such that $p_x > 0$:

$$\mathbf{E}[X]^2 = \left(\sum_x x p_x\right)^2 = (v \cdot w)^2 \leq (v \cdot v)(w \cdot w) = \left(\sum_x x^2 p_x\right)\left(\sum_x p_x\right) = \mathbf{E}[X^2]\mathbf{Pr}[X > 0]$$

Dividing by $\mathbf{E}[X^2] > 0$ (since $X$ is not constant 0) yields $\mathbf{Pr}[X > 0] \geq \mathbf{E}[X]^2/\mathbf{E}[X^2]$.

**Exercise 8.7.b:** For each $s \in S$, let the random variable $X_s$ indicate whether $h_r(s) = t$. Let $X = \sum_{s \in S} X_s$ be the number of $s \in S$ such that $h_r(s) = t$. Thus $X > 0$ iff $h_r(s) = t$ for some $s \in S$. By linearity of expectation (and with summations over elements of $S$):

$$\mathbf{E}[X] = \sum_s \mathbf{E}[X_s] = \sum_s \mathbf{Pr}\big[h_r(s) = t\big] = \sum_s 1/|T| = |S|/|T|$$

$$\begin{aligned}
\mathbf{E}[X^2] &= \mathbf{E}\Big[\sum_{s,s'} X_s X_{s'}\Big] \\
&= \sum_{s,s'} \mathbf{E}[X_s X_{s'}] \\
&= \sum_{s,s'} \mathbf{Pr}\big[h_r(s) = t = h_r(s')\big] \\
&= \sum_s 1/|T| + \sum_{s \neq s'} 1/|T|^2 \\
&= |S|/|T| + |S|\big(|S| - 1\big)/|T|^2 \\
&\leq |S|/|T| + |S|^2/|T|^2
\end{aligned}$$

By Exercise 8.7.a:

$$\mathbf{Pr}[X > 0] \geq \mathbf{E}[X]^2/\mathbf{E}[X^2] \geq \frac{|S|^2/|T|^2}{|S|/|T| + |S|^2/|T|^2} = \frac{|S|/|T|}{1 + |S|/|T|} = \frac{|S|}{|T| + |S|}$$

This is assuming $S \neq \emptyset$ so $X$ is not constant 0. If $S = \emptyset$ then the result is trivial.

**Exercise 8.8:** Each run is like a coin toss where heads means "the run's output is within a factor $1+\gamma$ of $|S|$". Lemma 7.7 says that if the big-$O$'s constant factor is large enough, then with probability $\geq 1-\varepsilon$, more than half the tosses are heads, in which case the median corresponds to a heads.

We can't use mean instead of median, because with high probability there may be at least one tails, which could be an arbitrarily bad estimate of $|S|$ and single-handedly drag the mean far from $|S|$.

**Exercise 8.9:** Assume NP $\subseteq$ BPP. Circuit Sat Counting has a fully poly-time randomized approximation scheme $\Pi$ (Theorem 8.12). Since NP $\subseteq$ RP (Theorem 7.11), Circuit Sat has a poly-time one-sided-error randomized program $\Pi'$. Define a fully poly-time one-sided-error randomized approximation scheme $\Pi''$ for Circuit Sat Counting:

> $\Pi''(C, \gamma, \varepsilon)$:
> run $\Pi'(C)$, amplified to have one-sided error probability $\varepsilon/2$
> if it rejected: output 0
> run $\Pi(C, \gamma, \varepsilon/2)$ and output the same thing

If $S = \emptyset$ then $\Pi'(C)$ always rejects, so $\Pi''(C, \gamma, \varepsilon)$ always outputs 0. If $S \neq \emptyset$ then:

$$\begin{aligned}
\mathbf{Pr}\big[\Pi''(C,\gamma,\varepsilon) \text{ errs}\big] &= \mathbf{Pr}\big[\text{the amplified } \Pi'(C) \text{ errs or } \Pi(C,\gamma,\varepsilon/2) \text{ errs}\big] \\
&\leq \mathbf{Pr}\big[\text{the amplified } \Pi'(C) \text{ errs}\big] + \mathbf{Pr}\big[\Pi(C,\gamma,\varepsilon/2) \text{ errs}\big] \\
&\leq \varepsilon/2 + \varepsilon/2 \\
&= \varepsilon
\end{aligned}$$

**Exercise 8.10:** In steps 1 and 2 of the proof of Theorem 8.12, we designed a fully poly-time deterministic oracle reduction to $A$, so it suffices to show $A \in \mathsf{P}$. In step 3, we designed a poly-time randomized reduction from $A$ to $B \in \mathsf{NP}$. Assuming $\mathsf{P} = \mathsf{NP}$, we have $B \in \mathsf{P}$ and thus $A \in \mathsf{BPP}$. Assuming $\mathsf{P} = \mathsf{NP}$, we have $\mathsf{P} = \mathsf{BPP}$ (Theorem 7.41 or Exercise 7.17), so $A \in \mathsf{P}$.

**Exercise 8.11.a:** If each $q_i$ were exactly $|S'_i|/|S_i|$, the output would be $\sum_i \big(|S'_i|/|S_i|\big)|S_i| = \sum_i |S'_i| = |S|$. By a union bound, with probability $\geq 1-\varepsilon$, every $q_i$ is within $\pm\gamma/2m$ of $|S'_i|/|S_i|$. In that case, we claim that the output $\sum_{i=1}^{m} q_i\,|S_i|$ is within a factor $1+\gamma$ of $|S|$. Note that $\sum_i |S_i| \leq m|S|$ since $\sum_i |S_i|$ counts each element of $S$ at most $m$ times (once per satisfied term).

$$\begin{aligned}
\textstyle\sum_i q_i\,|S_i| &\leq \textstyle\sum_i \big(|S'_i|/|S_i| + \gamma/2m\big)|S_i| \\
&= \textstyle\sum_i |S'_i| + (\gamma/2m)\sum_i |S_i| \\
&\leq |S| + (\gamma/2m)m|S| \\
&= (1+\gamma/2)|S| \\
&\leq (1+\gamma)|S|
\end{aligned}$$

$$\begin{aligned}
\textstyle\sum_i q_i\,|S_i| &\geq \textstyle\sum_i \big(|S'_i|/|S_i| - \gamma/2m\big)|S_i| \\
&= \textstyle\sum_i |S'_i| - (\gamma/2m)\sum_i |S_i| \\
&\geq |S| - (\gamma/2m)m|S| \\
&= (1-\gamma/2)|S| \\
&\geq \tfrac{1}{1+\gamma}|S|
\end{aligned}$$

For the last line, we used $(1+\gamma)(1-\gamma/2) = 1+\gamma-\gamma/2-\gamma^2/2 \geq 1$.

The time efficiency is $m \cdot O\big(\frac{1}{(\gamma/2m)^2}\log\frac{m}{\varepsilon}\big)\cdot(\mathrm{poly}\,N) \leq \mathrm{poly}\big(\frac{N}{\gamma}\log\frac{1}{\varepsilon}\big)$.

**Exercise 8.11.b:** If the algorithm doesn't abort, then its output is in $S$ by definition. Consider any $b \in S$, say $b \in S'_j$. The algorithm outputs $b$ iff it samples $i = j$, doesn't abort on the $2^{\text{nd}}$ line, and then samples $a = b$. By the chain rule:

$$\begin{aligned}
\mathbf{Pr}[\text{output is } b] \;&=\; \mathbf{Pr}[i = j] \cdot \\
&\quad\; \mathbf{Pr}\big[\text{not abort on } 2^{\text{nd}} \text{ line} \,\big|\, i = j\big] \cdot \\
&\quad\; \mathbf{Pr}\big[a = b \,\big|\, i = j \text{ and not abort on } 2^{\text{nd}} \text{ line}\big] \\
&= \frac{1}{m} \cdot \frac{|S_j|}{\max\big(|S_1|, \ldots, |S_m|\big)} \cdot \frac{1}{|S_j|} \\
&= 1/m \max\big(|S_1|, \ldots, |S_m|\big)
\end{aligned}$$

Since this probability doesn't depend on $b$, the output distribution is uniform over $S$, conditioned on not aborting. Since $|S| \geq \max\big(|S_1|, \ldots, |S_m|\big)$:

$$\mathbf{Pr}[\text{not abort}] \;=\; \textstyle\sum_{b \in S} \mathbf{Pr}[\text{output is } b] \;=\; |S|/m \max\big(|S_1|, \ldots, |S_m|\big) \;\geq\; 1/m$$

To amplify this to $\mathbf{Pr}[\text{abort}] \leq \varepsilon$, we run the algorithm $O\big(m \log \frac{1}{\varepsilon}\big)$ times, abort if all runs abort, and otherwise output the assignment produced by the first run that doesn't abort (Lemma 7.5).

**Exercise 8.12:** For each $s \in S$, let the random variable $X_s$ indicate whether $h_r(s) = f(s)$. Since $h_r(s)$ is uniformly distributed over $T$, we have $\mathbf{E}[X_s] = \mathbf{Pr}_r\big[h_r(s) = f(s)\big] = 1/|T|$ and $\mathbf{Var}[X_s] = \mathbf{E}[X_s]\big(1 - \mathbf{E}[X_s]\big) \leq \mathbf{E}[X_s]$ (Lemma 7.18). Let $X = \sum_{s \in S} X_s$ be the number of $s \in S$ such that $h_r(s) = f(s)$. By linearity of expectation, $\mathbf{E}[X] = \sum_{s \in S} \mathbf{E}[X_s] = |S|/|T|$. Since $h_r(s)$ (over all $s$) are pairwise independent, the $X_s$ random variables are pairwise independent, so $\mathbf{Var}[X] = \sum_{s \in S} \mathbf{Var}[X_s] \leq \sum_{s \in S} \mathbf{E}[X_s] = \mathbf{E}[X]$ (Lemma 8.16). By Lemma 7.16:

$$\begin{aligned}
\mathbf{Pr}_r\big[h_r(s) \neq f(s) \text{ for all } s \in S\big] &= \mathbf{Pr}[X = 0] \\
&\leq \mathbf{Pr}\big[\big|X - \mathbf{E}[X]\big| \geq \mathbf{E}[X]\big] \\
&\leq \mathbf{Var}[X]/\mathbf{E}[X]^2 \\
&\leq 1/\mathbf{E}[X] \\
&= |T|/|S|
\end{aligned}$$

**Exercise 9.1:** For a received word $r \in \{0,1,\perp\}^n$ and a bit $b$, define $r^b \in \{0,1\}^n$ as $r$ with $b$s in place of the $\perp$s. If $r$ agrees with $E(m)$ and has $< d$ erasures, then either $\text{dist}\big(r^0, E(m)\big) < d/2$, in which case $D(r^0) = m$, or $\text{dist}\big(r^1, E(m)\big) < d/2$, in which case $D(r^1) = m$, depending on whether $E(m)$ has more 0s or 1s among $r$'s $\perp$s. Thus the following $D'$ decodes if $< d$ erasures occur.

$D'(r)$:
    if $E(D(r^0))$ agrees with $r$: output $D(r^0)$
    else: output $D(r^1)$

**Exercise 9.2:** Consider any distinct $m_1, m_2 \in \{0,1\}^k$ with $\text{dist}\big(E(m_1), E(m_2)\big) = d$. Let $S = \big\{i \in [n] : E(m_1)_i \neq E(m_2)_i\big\}$, so $|S| = d$. Define $r$ from $E(m_1)$ by flipping $\lceil d/2 \rceil$ bits with indices in $S$ (it doesn't matter which $\lceil d/2 \rceil$ indices in $S$). Then $\text{dist}\big(r, E(m_1)\big) = \lceil d/2 \rceil$ and $\text{dist}\big(r, E(m_2)\big) = d - \lceil d/2 \rceil \leq \lceil d/2 \rceil$. But since $m_1 \neq m_2$, either $D(r) \neq m_1$ or $D(r) \neq m_2$.

**Exercise 9.3.a:** Since $E$ is linear (each codeword bit is the parity of some message bits), $\text{dist}(E)$ is the minimum weight of any nonzero message's codeword (Lemma 9.1). We have $\text{dist}(E) \geq 3$:

- If $\text{weight}(m) \geq 3$ then $\text{weight}\big(E(m)\big) \geq 3$ since $E(m)$ contains $m$.
- If $\text{weight}(m) = 2$ then $\text{weight}\big(E(m)\big) \geq 3$ since $m$ contributes 2, and some adjacent pair of unequal bits of $m$ contributes 1 (and such a pair exists, since $k \geq 3$).
- If $\text{weight}(m) = 1$ with $m_i = 1$, then $\text{weight}\big(E(m)\big) = 3$ since $m$ contributes 1 and $m_i \oplus m_{i+1} = 1$ and $m_{i-1} \oplus m_i = 1$ (wrapping around) so these two adjacent pairs each contribute 1 (and they're not the same pair, since $k \geq 3$), and all other pairs of adjacent bits of $m$ are both 0s and thus contribute 0.

The last bullet implies $\text{dist}(E) \leq 3$, so $\text{dist}(E) = 3$. The generator matrix when $k = 4$ is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

**Exercise 9.3.b:** Since $E$ is linear (each codeword bit is the parity of some message bits), $\text{dist}(E)$ is the minimum weight of any nonzero message's codeword (Lemma 9.1). We have $\text{dist}(E) \geq k$ because if $m \neq 0^k$ then $\text{weight}\big(E(m)\big) \geq k$, since $m$ contributes $\text{weight}(m)$, and the parities of pairs of bits of $m$ contribute at least $k - \text{weight}(m)$: Assuming $m_i = 1$, we have $m_i \oplus m_j = 1$ for each index $j$ such that $m_j = 0$, of which there are $k - \text{weight}(m)$. Also, $\text{dist}(E) \leq k$ because $\text{weight}\big(E(1^k)\big) = k$, since $E(1^k)$ contains the message $1^k$ and all pairs of bits of $1^k$ have parity 0. Thus $\text{dist}(E) = k$. The generator matrix when $k = 4$ is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

**Exercise 9.4:** Assume $d = \text{dist}(E)$ is odd. For any $m_1 \neq m_2$ such that $\text{dist}\big(E(m_1), E(m_2)\big) = d$, we have $\oplus(E(m_1)) \neq \oplus(E(m_2))$ because going from $E(m_1)$ to $E(m_2)$ means flipping an odd number of bits, which also flips the parity, and thus $\text{dist}\big(E_\oplus(m_1), E_\oplus(m_2)\big) = d + 1$. This shows that $\text{dist}(E_\oplus) \leq d + 1$. To see that $\text{dist}(E_\oplus) \geq d + 1$, consider any $m_1 \neq m_2$ and consider two cases:

- If $\text{dist}\big(E(m_1), E(m_2)\big) \geq d + 1$ then $\text{dist}\big(E_\oplus(m_1), E_\oplus(m_2)\big) \geq \text{dist}\big(E(m_1), E(m_2)\big) \geq d + 1$ since $E_\oplus(m)$ contains $E(m)$ for every $m$.
- If $\text{dist}\big(E(m_1), E(m_2)\big) = d$ then $\text{dist}\big(E_\oplus(m_1), E_\oplus(m_2)\big) = d + 1$, as we argued above.

**Exercise 9.5:** If $d = 0$ then the ball has size $1 = (q^n)^0$, so assume $d > 0$. The ball's exact size is $\sum_{i=0}^{d} \binom{n}{i}(q-1)^i$ since choosing a string at distance exactly $i$ from the center means choosing a set of $i$ positions to change ($\binom{n}{i}$ possibilities) and choosing which symbols to change them to ($(q-1)^i$ possibilities). Letting $p = d/n$:

$$\begin{aligned}(q^n)^{H_q(p)} &= q^{np\log_q(1/p)} \cdot q^{n(1-p)\log_q(1/(1-p))} \cdot q^{np\log_q(q-1)} \\ &= (1/p)^{np} \cdot (1/(1-p))^{n-np} \cdot (q-1)^{np} \\ &= 1/(p/(q-1))^d (1-p)^{n-d}\end{aligned}$$

Relating these two quantities using the binomial formula,

$$\begin{aligned}\text{size of ball} \big/ (q^n)^{H_q(p)} &= \textstyle\sum_{i=0}^{d} \binom{n}{i}(q-1)^i \cdot (p/(q-1))^d (1-p)^{n-d} \\ &\leq \textstyle\sum_{i=0}^{n} \binom{n}{i}(q-1)^i \cdot (p/(q-1))^i (1-p)^{n-i} \\ &= \textstyle\sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} \\ &= \big(p + (1-p)\big)^n \\ &= 1\end{aligned}$$

because

$$p/(q-1) = (d/n)/(q-1) \leq (1-1/q)/(q-1) = 1/q = 1-(1-1/q) \leq 1-d/n = 1-p$$

and thus $(p/(q-1))^d(1-p)^{n-d} \leq (p/(q-1))^i(1-p)^{n-i}$ for $i \leq d \leq n(1-1/q)$.

**Exercise 9.6:** Let $K = 2^k$ be the number of messages. Let $D$ be the number of pairs $(m, i)$ such that $E(m)_i = 1$. On one hand, $\text{weight}\big(E(m)\big) \geq d$ for every $m \neq 0^k$ (Lemma 9.1), so:

$$D \;=\; \textstyle\sum_m \text{weight}\big(E(m)\big) \;\geq\; (K-1)d$$

On the other hand, letting $K_i$ be the number of $m$ such that $E(m)_i = 1$,

$$D \;=\; \textstyle\sum_i K_i \;\leq\; nK/2$$

because $K_i = K/2$ if the $i^{\text{th}}$ column of $E$'s generator matrix is not $0^k$, by the random subsum principle (Lemma 7.2.*(i)*) since $E(m)_i = m \odot$ (this column), and otherwise $K_i = 0$. Combining these two inequalities:

$$\begin{aligned}
(K-1)d &\leq nK/2 && \\
K-1 &\leq K(n/2d) && \text{(dividing both sides by } d\text{)} \\
K(1-n/2d) &\leq 1 && \text{(rearranging)} \\
K &\leq 1/(1-n/2d) && \text{(dividing both sides by } 1-n/2d\text{)}
\end{aligned}$$

The inequality's direction didn't flip, because $1-n/2d > 0$ by the assumption $d > n/2$.

**Exercise 9.7:** Let $K = q^k$ be the number of messages. Let $D$ be the number of pairs $((m_1, m_2), i)$ where $(m_1, m_2)$ is an ordered pair of (possibly equal) messages whose codewords differ at coordinate $i$, that is, $E(m_1)_i \neq E(m_2)_i$. On one hand:

$$D \;=\; \sum_{(m_1,m_2)} \mathrm{dist}\big(E(m_1), E(m_2)\big) \;\geq\; \sum_{m_1 \neq m_2} d \;=\; K(K-1)d$$

On the other hand, if $K_{i,a}$ is the number of messages whose codewords have $a$ at coordinate $i$, then the number of ordered pairs of messages whose codewords agree at coordinate $i$ is $\sum_a K_{i,a}^2 \geq \left(\sum_a K_{i,a}\right)^2 / q = K^2/q$ (Corollary 0.27), and thus:

$$D \;=\; \sum_i \big(K^2 - \sum_a K_{i,a}^2\big) \;\leq\; \sum_i \big(K^2 - K^2/q\big) \;=\; nK^2(1-1/q)$$

Combining these two inequalities and letting $t = 1 - 1/q$:

$$\begin{aligned}
K(K-1)d &\leq nK^2 t \\
K-1 &\leq Knt/d && \text{(dividing both sides by } Kd\text{)} \\
K(1-nt/d) &\leq 1 && \text{(rearranging)} \\
K &\leq 1/(1-nt/d) && \text{(dividing both sides by } 1-nt/d > 0\text{)}
\end{aligned}$$

**Exercise 9.8.a:** This is trivial when $d = 0$, so assume $d > 0$. We showed in the proof of Lemma 9.4 that (size of ball) $= \sum_{i=0}^{d} \binom{n}{i} \geq \binom{n}{d}$ and $(2^n)^{H(p)} = 1/p^d(1-p)^{n-d}$, which imply (size of ball) $\Big/ (2^n)^{H(p)} \geq \binom{n}{d} p^d (1-p)^{n-d}$. We also showed (size of ball) $\Big/ (2^n)^{H(p)} \leq \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} = 1$ in the proof of Lemma 9.4. If we can show that the $i = d$ summand is the largest—and is thus $\geq 1/(n+1)$ since there are $n+1$ many summands—then it follows that (size of ball) $\Big/ (2^n)^{H(p)} \geq 1/(n+1)$. For $i \geq 1$, the ratio of the $i^{\text{th}}$ and $(i-1)^{\text{st}}$ summands is:

$$\frac{\binom{n}{i} p^i (1-p)^{n-i}}{\binom{n}{i-1} p^{i-1} (1-p)^{n-i+1}} = \frac{(n-i+1)p}{i(1-p)} = \frac{(n-i+1)d}{i(n-d)} \begin{cases} > 1 & \text{if } i \leq d \\ < 1 & \text{if } i > d \end{cases}$$

The inequalities are because:

$$\text{numerator} - \text{denominator} = (n-i+1)d - i(n-d) = n(d-i) + d \begin{cases} \geq d > 0 & \text{if } i \leq d \\ \leq -n + d < 0 & \text{if } i > d \end{cases}$$

Thus as $i$ goes from 0 to $n$, the summands increase until the $d^{\text{th}}$ and then decrease.

**Exercise 9.8.b:** Consider a code $E\colon \{0,1\}^k \to \{0,1\}^n$ with distance $d \geq pn$. The $2^k$ many balls in $\{0,1\}^n$ of radius $\lfloor (d-1)/2 \rfloor$ centered at the codewords must be disjoint, by the triangle inequality.

$$\begin{aligned}
(\text{number of balls}) \cdot (\text{size of each ball}) &\leq \text{size of } \{0,1\}^n \\
\text{size of each ball} &\geq (2^n)^{H(\lfloor (d-1)/2 \rfloor / n)}/(n+1) && \text{(Exercise 9.8.a)} \\
&\geq 2^{nH((d/2-1/2-1)/n)}/(n+1) \\
&\geq 2^{nH(p/2-3/2n)-\log(n+1)} \\
&\geq 2^{n(H(p/2-o(1))-\log(n+1)/n)} \\
&\geq 2^{n(H(p/2)-o(1)-o(1))} \\
&\geq 2^{n(H(p/2)-o(1))} \\
2^k \cdot 2^{n(H(p/2)-o(1))} &\leq 2^n \\
2^k &\leq 2^{n(1-H(p/2)+o(1))} \\
k/n &\leq 1-H(p/2)+o(1)
\end{aligned}$$

**Exercise 9.8.c:** Consider a code $E\colon \{0,1\}^k \to \{0,1\}^n$ with distance $d \geq (1/2-\varepsilon)n$. Define $e = \left\lfloor (1/2-\sqrt{\varepsilon})n \right\rfloor$. Let $D$ be the number of pairs $(m,r) \in \{0,1\}^k \times \{0,1\}^n$ such that $\mathrm{dist}\big(r, E(m)\big) \leq e$. On one hand, by Exercise 9.8.a:

$$D \;=\; \textstyle\sum_m (\text{size of ball of radius } e) \;\geq\; 2^k \cdot (2^n)^{H(e/n)}/(n+1) \;\geq\; 2^k \cdot 2^{n(H(1/2-\sqrt{\varepsilon})-o(1))}$$

(In the last step, we reasoned like in Exercise 9.8.b.) On the other hand, by Theorem 9.8:

$$D \;=\; \textstyle\sum_r (\text{number of codewords in ball of radius } e \text{ centered at } r) \;\leq\; 2^n/\varepsilon \;\leq\; 2^{n(1+o(1))}$$

Combining these yields:

$$\begin{aligned} 2^k \cdot 2^{n(H(1/2-\sqrt{\varepsilon})-o(1))} &\leq 2^{n(1+o(1))} \\ 2^k &\leq 2^{n(1+o(1)-H(1/2-\sqrt{\varepsilon})+o(1))} \\ k/n &\leq 1-H(1/2-\sqrt{\varepsilon})+o(1) \end{aligned}$$

**Exercise 9.9:** *(i)*: If $E\colon \mathbb{Z}_q^k \to \mathbb{Z}_q^n$ is linear then

$$E(m_1 - m_2) \;=\; E(m_1 + (q-1)m_2) \;=\; E(m_1) + (q-1)E(m_2) \;=\; E(m_1) - E(m_2)$$

and thus:

$$\begin{aligned}
\text{dist}(E) \;&=\; \min_{m_1 \neq m_2} \text{dist}\big(E(m_1), E(m_2)\big) \\
&=\; \min_{m_1 \neq m_2} \text{weight}\big(E(m_1) - E(m_2)\big) \\
&=\; \min_{m_1 \neq m_2} \text{weight}\big(E(m_1 - m_2)\big) \\
&=\; \min_{m \neq 0^k} \text{weight}\big(E(m)\big) \qquad\qquad (m_1 \neq m_2 \text{ iff } m_1 - m_2 \neq 0^k)
\end{aligned}$$

*(ii)*: We prove that $E\colon \mathbb{Z}_q^k \to \mathbb{Z}_q^n$ is linear iff there exists a $k \times n$ matrix $G$ such that $E(m) = mG$ over $\mathbb{Z}_q$:

$\Leftarrow$: Suppose there exists a generator matrix $G$ for $E$. For all $m_1$ and $m_2$,

$$E(m_1 + m_2) \;=\; (m_1 + m_2)G \;=\; m_1G + m_2G \;=\; E(m_1) + E(m_2)$$

by the distributive property of matrix multiplication, and for every scalar $c$ and every vector $m$,

$$E(cm) \;=\; (cm)G \;=\; c(mG) \;=\; cE(m)$$

by the associative property of matrix multiplication. Thus $E$ is linear.

$\Rightarrow$: Suppose $E$ is linear. For each $i \in [k]$, consider the message $e^i \in \mathbb{Z}_q^k$ with a 1 at index $i$ and 0s everywhere else. Define $G$'s $i^{\text{th}}$ row $G_i$ to be $E(e^i)$. For any message $m$:

$$E(m) \;=\; E\Big(\textstyle\sum_i m_i e^i\Big) \;=\; \sum_i m_i E(e^i) \;=\; \sum_i m_i G_i \;=\; mG$$

Thus $G$ is a generator matrix for $E$.

**Exercise 9.10:**

$$\begin{aligned}
M(0) &= 0^3 + 4 \cdot 0 + 1 = 1 \\
M(1) &= 1^3 + 4 \cdot 1 + 1 = 6 \\
M(2) &= 2^3 + 4 \cdot 2 + 1 = 3 \\
M(3) &= 3^3 + 4 \cdot 3 + 1 = 5 \\
M(4) &= 4^3 + 4 \cdot 4 + 1 = 4 \\
M(5) &= 5^3 + 4 \cdot 5 + 1 = 6 \\
M(6) &= 6^3 + 4 \cdot 6 + 1 = 3
\end{aligned}$$

$$\begin{aligned}
E_{7,4}(M) &= \big(M(0), M(1), M(2), M(3), M(4), M(5), M(6)\big) \\
&= (1, 6, 3, 5, 4, 6, 3)
\end{aligned}$$

**Exercise 9.11:** The goal is to find the unique polynomial $M$ of degree $\leq 3$ over $\mathbb{Z}_{11}$ such that $M(0) = 0$ and $M(2) = 3$ and $M(4) = 0$ and $M(7) = 1$. We have $(a_1, a_2, a_3, a_4) = (0, 2, 4, 7)$ and $(b_1, b_2, b_3, b_4) = (0, 3, 0, 1)$.

$$I_2 = \frac{(x-0)(x-4)(x-7)}{(2-0)(2-4)(2-7)} = \frac{x^3+6x}{9} = 5x^3+0x^2+8x+0$$

$$I_4 = \frac{(x-0)(x-2)(x-4)}{(7-0)(7-2)(7-4)} = \frac{x^3+5x^2+8x}{6} = 2x^3+10x^2+5x+0$$

$$\begin{aligned} M &= 0I_1 + 3I_2 + 0I_3 + 1I_4 \\ &= 3I_2 + I_4 \\ &= (4x^3+0x^2+2x+0) + (2x^3+10x^2+5x+0) \\ &= 6x^3+10x^2+7x+0 \end{aligned}$$

Let's double-check that this polynomial indeed interpolates through the given points:

$$\begin{aligned} M(0) &= 6\cdot 0^3 + 10\cdot 0^2 + 7\cdot 0 = 0 \\ M(2) &= 6\cdot 2^3 + 10\cdot 2^2 + 7\cdot 2 = 3 \\ M(4) &= 6\cdot 4^3 + 10\cdot 4^2 + 7\cdot 4 = 0 \\ M(7) &= 6\cdot 7^3 + 10\cdot 7^2 + 7\cdot 7 = 1 \end{aligned}$$

The message was $M = (6, 10, 7, 0)$. The uncorrupted codeword was:

$$E_{11,4}(M) = (0, 1, 3, 9, 0, 1, 4, 1, 6, 0, 8)$$

**Exercise 9.12:** $110 - 011 = 110 \oplus 011 = 101$

$110 \cdot 011 = \big((x^2 + x) \cdot (x + 1)\big) \bmod P = (x^3 + x) \bmod P = x^2 + x + 1 = 111$ because $x^3 + x = 1 \cdot P + (x^2 + x + 1)$

$101 \cdot 111 = \big((x^2 + 1) \cdot (x^2 + x + 1)\big) \bmod P = (x^4 + x^3 + x + 1) \bmod P = 1 = 001$ because $x^4 + x^3 + x + 1 = x \cdot P + 1$

$111^{-1} = 101$

$100/111 = 100 \cdot 101 = \big(x^2 \cdot (x^2 + 1)\big) \bmod P = (x^4 + x^2) \bmod P = x + 1 = 011$ because $x^4 + x^2 = (x + 1) \cdot P + (x + 1)$

**Exercise 9.13:**

- If $m \geq n$, define $h\colon (\{0,1\}^m)^2 \times \{0,1\}^n \to \{0,1\}^m$ by $h_{a,b}(u) = a \cdot (0^{m-n}u) + b$ using arithmetic in the field $\mathbb{F}_{2^m} = \{0,1\}^m$. Making $u$ have $m$ bits by padding it with 0s (or with anything) doesn't affect the pairwise uniformity.
- If $m < n$, define $h\colon (\{0,1\}^n)^2 \times \{0,1\}^n \to \{0,1\}^m$ by $h_{a,b}(u) = a \cdot u + b$ truncated to $m$ bits, using arithmetic in the field $\mathbb{F}_{2^n} = \{0,1\}^n$. The truncation doesn't affect the pairwise uniformity.

Either way, $h$ uses $2\max(m,n)$ random bits. The construction from Lemma 8.8 uses $2m+n-1$ random bits. We claim that $2\max(m,n) < 2m+n-1$ iff $m > (n+1)/2$ and $n > 1$. Of course, if $n = 1$ then $2\max(m,n) = 2m = 2m+n-1$, so assume $n > 1$.

- If $m \geq n$, we have $2m < 2m+n-1$ and $m > (n+1)/2$.
- If $m < n$, we have $2n < 2m+n-1$ iff $m > (n+1)/2$.

**Exercise 9.14.a:** Consider the $2^\ell$-ary code $E : \mathbb{F}_{2^\ell}^1 \to \mathbb{F}_{2^\ell}^1$ where $E(a) = a \cdot b$. Then $E'$ is linear over $\mathbb{Z}_2$ by the distributive axiom for $\mathbb{F}_{2^\ell}$, that is, $(a_1 + a_2) \cdot b = (a_1 \cdot b) + (a_2 \cdot b)$, so $E'$ has an $\ell \times \ell$ generator matrix $G_b$ (Lemma 9.2).

**Exercise 9.14.b:** We obtain the generator matrix of $E'$ (over $\mathbb{Z}_2$) from the generator matrix of $E$ (over $\mathbb{F}_{2^\ell}$) by replacing each entry with the $\ell \times \ell$ matrix over $\mathbb{Z}_2$ representing multiplication by that entry (Exercise 9.14.a).

**Exercise 9.15.a:**

$$\begin{aligned} M &= 11x^2 + 01x + 00 = 11x^2 + x \\ E_{4,3}(M) &= \big(M(00), M(01), M(10), M(11)\big) \\ &= (00, 10, 00, 10) \end{aligned}$$

because:

$$\begin{aligned} M(00) &= 11 \cdot 00^2 + 00 = 00 + 00 = 00 \\ M(01) &= 11 \cdot 01^2 + 01 = 11 + 01 = 10 \\ M(10) &= 11 \cdot 10^2 + 10 = 10 + 10 = 00 \\ M(11) &= 11 \cdot 11^2 + 11 = 01 + 11 = 10 \end{aligned}$$

**Exercise 9.15.b:** The goal is to find the unique polynomial $M$ of degree $\leq 1$ over $\mathbb{F}_4 = \{0,1\}^2$ such that $M(00) = 10$ and $M(11) = 01$. We have $(a_1, a_2) = (00, 11)$ and $(b_1, b_2) = (10, 01)$.

$$\begin{aligned} I_1 &= \frac{x-11}{00-11} = \frac{x+11}{11} = 10x+01 \\ I_2 &= \frac{x-00}{11-00} = \frac{x+00}{11} = 10x+00 \end{aligned}$$

$$\begin{aligned} M &= 10I_1 + 01I_2 \\ &= (11x+10)+(10x+00) \\ &= 01x+10 \end{aligned}$$

Let's double-check that this polynomial indeed interpolates through the given points:

$$\begin{aligned} M(00) &= 01 \cdot 00 + 10 = 10 \\ M(11) &= 01 \cdot 11 + 10 = 01 \end{aligned}$$

The message was $M = (01, 10)$. The uncorrupted codeword was $E_{4,2}(M) = (10, 11, 00, 01)$.

**Exercise 9.16:** We claim that for large enough $q = 2^\ell$ and $k = \lceil 2cq \rceil$, the flattened augmented polynomial code $E^{*\prime}_{q,k}$ has rate $\geq c$ and relative distance $> (1/2 - c)/10 > 0$.

Let's declutter by omitting the $q, k$ subscript from the codes. $E^{*\prime}$ has message length $\ell k$ and codeword length $2\ell q$ and thus rate $\ell k / 2\ell q \geq c$. As in the proof of Theorem 9.23:

$$\text{dist}(E^{*\prime}) \;=\; \min_{M \neq (0^\ell)^k} \text{weight}\big(E^*(M)'\big)$$

Consider any $M \neq (0^\ell)^k$, corresponding to a nonzero polynomial of degree $< k$ over $\mathbb{F}_q$. As in the proof of Theorem 9.23, the number of $a \in \mathbb{F}_q$ with $M(a) \neq 0^\ell$ and $\text{weight}\big((M(a), M(a) \cdot a)'\big) \leq \ell/5$ is $< q^{0.94} < (1/2 - c)q$ for large enough $q$. At least $q - k + 1 > (1 - 2c)q$ many $a \in \mathbb{F}_q$ have $M(a) \neq 0^\ell$ (since $M$ has $\leq k - 1$ roots), and more than $(1 - 2c)q - (1/2 - c)q = (1/2 - c)q$ of them have $\text{weight}\big((M(a), M(a) \cdot a)'\big) > \ell/5$, so:

$$\text{weight}\big(E^*(M)'\big) \;=\; \sum_{a \in \mathbb{F}_q} \text{weight}\big((M(a), M(a) \cdot a)'\big) \;>\; (1/2 - c)q \cdot (\ell/5) \;=\; ((1/2 - c)/10) \cdot (2\ell q)$$

Thus $E^{*\prime}$ has relative distance $> (1/2 - c)/10$.

**Exercise 9.17:** Let $E = E_{\text{in}} \circ E_{\text{out}}$. Suppose the message is $m' \in \{0,1\}^{\ell k}$ and the received word is $r' \in \{0,1\}^{hn}$, which are the flattened versions of $m \in (\{0,1\}^{\ell})^k$ and $r \in (\{0,1\}^h)^n$.

$D(r')$:
  for each $i \in [n]$: let $s_i \leftarrow D_{\text{in}}(r_i)$
  output $D_{\text{out}}(s)'$ where $s = (s_1, \ldots, s_n)$

If $\text{dist}\big(r', E(m')\big) < d_{\text{out}} d_{\text{in}}/4$, then $\text{dist}\big(r_i, E_{\text{in}}(E_{\text{out}}(m)_i)\big) \geq d_{\text{in}}/2$ for fewer than $d_{\text{out}}/2$ many $i \in [n]$, since otherwise:

$$\text{dist}\big(r', E(m')\big) \;=\; \textstyle\sum_{i=1}^{n} \text{dist}\big(r_i, E_{\text{in}}(E_{\text{out}}(m)_i)\big) \;\geq\; (d_{\text{out}}/2) \cdot (d_{\text{in}}/2)$$

By the correctness of $D_{\text{in}}$, if $\text{dist}\big(r_i, E_{\text{in}}(E_{\text{out}}(m)_i)\big) < d_{\text{in}}/2$ then $s_i = E_{\text{out}}(m)_i$. It follows that $\text{dist}\big(s, E_{\text{out}}(m)\big) < d_{\text{out}}/2$. By the correctness of $D_{\text{out}}$, we have $D_{\text{out}}(s) = m$. Thus $D(r') = m'$.

**Exercise 9.18:** For any code $E$, define $\mathrm{dist}^{\max}(E) = \max_{m_1 \neq m_2} \mathrm{dist}\big(E(m_1), E(m_2)\big)$. For all $E_{\mathrm{out}}$ and $E_{\mathrm{in}}$,

$$\mathrm{dist}(E_{\mathrm{in}} \circ E_{\mathrm{out}}) \;\leq\; \mathrm{dist}(E_{\mathrm{out}}) \cdot \mathrm{dist}^{\max}(E_{\mathrm{in}})$$

because for some $m_1 \neq m_2$ we have $\mathrm{dist}\big(E_{\mathrm{out}}(m_1), E_{\mathrm{out}}(m_2)\big) = \mathrm{dist}(E_{\mathrm{out}})$ and for each $i$ with $E_{\mathrm{out}}(m_1)_i \neq E_{\mathrm{out}}(m_2)_i$ we have $\mathrm{dist}\big(E_{\mathrm{in}}(E_{\mathrm{out}}(m_1)_i), E_{\mathrm{in}}(E_{\mathrm{out}}(m_2)_i)\big) \leq \mathrm{dist}^{\max}(E_{\mathrm{in}})$, and so:

$$\begin{aligned} \mathrm{dist}\big((E_{\mathrm{in}} \circ E_{\mathrm{out}})(m_1'), (E_{\mathrm{in}} \circ E_{\mathrm{out}})(m_2')\big) \;&=\; \textstyle\sum_i \mathrm{dist}\big(E_{\mathrm{in}}(E_{\mathrm{out}}(m_1)_i), E_{\mathrm{in}}(E_{\mathrm{out}}(m_2)_i)\big) \\ &\leq\; \mathrm{dist}(E_{\mathrm{out}}) \cdot \mathrm{dist}^{\max}(E_{\mathrm{in}}) \end{aligned}$$

The polynomial code $E_{\mathrm{out}} = E_{q,k} \colon (\{0,1\}^{\ell})^k \to (\{0,1\}^{\ell})^q$ has $\mathrm{dist}(E_{\mathrm{out}}) = q - k + 1$. The dot product code $E_{\mathrm{in}} \colon \{0,1\}^{\ell} \to \{0,1\}^q$ has $\mathrm{dist}^{\max}(E_{\mathrm{in}}) = q/2$ by the random subsum principle (Lemma 7.2.*(ii)*). Thus $\mathrm{dist}(E_{\mathrm{in}} \circ E_{\mathrm{out}}) \leq (q - k + 1) \cdot (q/2)$.

**Exercise 9.19.a:** For each $a \in \mathbb{Z}_2^\ell$, $E(M)_a = M(a)$ is the dot product of $M$ (as a vector of coefficients, indexed by sets $S \subseteq [\ell]$ with $|S| \leq e$ corresponding to monomials $\prod_{j \in S} x_j$) and the vector that has $\prod_{j \in S} a_j$ at index $S$. The latter vector is the $a^{\text{th}}$ column of the generator matrix.

**Exercise 9.19.b:** Associate each $S \subseteq [\ell]$ with its characteristic bit string $s \in \{0,1\}^\ell$, where $s_j = 1$ iff $j \in S$. The sets $S$ with $|S| \leq e$ correspond to strings $s$ with weight$(s) \leq e$, that is, the ball in $\{0,1\}^\ell$ of radius $e$ centered at $0^\ell$. This ball has size $k = \sum_{i=0}^{e} \binom{\ell}{i} \leq (2^\ell)^{H(e/\ell)}$ if $e \leq \ell/2$ (Lemma 9.4). Thus $E$ has rate $k/n \leq n^{H(e/\ell)}/n = 1/n^{1-H(e/\ell)}$.

**Exercise 9.19.c:** Since $E$ is linear (Exercise 9.19.a), $\text{dist}(E)$ is the minimum weight of any nonzero message's codeword (Lemma 9.1).

$\text{dist}(E) \leq 2^{\ell-e}$ because if $M_{\{1,\ldots,e\}} = 1$ and $M_S = 0$ for all other $S$, that is, $M(x) = \prod_{j \leq e} x_j$, then $M$ is nonzero and $\text{weight}\big(E(M)\big) = 2^{\ell-e}$ since $M(a) = 1$ for exactly $2^{\ell-e}$ many $a$, namely when $a_j = 1$ for all $j \leq e$ and $a_j$ can be anything for $j > e$.

Now, we prove $\text{dist}(E) \geq 2^{\ell-e}$. For any nonzero $M$, we show:

$$\text{weight}\big(E(M)\big) \;\geq\; 2^{\ell-(\text{total degree of } M)} \;\geq\; 2^{\ell-e}$$

Assume $M$ has total degree exactly $e$, since if $M$ has total degree $< e$, then $\text{weight}\big(E(M)\big)$ would be even higher. Consider any particular $S \subseteq [\ell]$ with $|S| = e$ and $M_S = 1$. Consider any assignment $a_{-S}$ to all the variables not indexed by $S$. For each such $a_{-S}$, of which there are $2^{\ell-e}$, we claim there's at least one assignment $a_S$ to the variables indexed by $S$ such that $M(a) = 1$ for the combined assignment $a$. Contracting $M$ by plugging in $a_{-S}$ yields a multilinear polynomial $M|_{a_{-S}}$ on the variables indexed by $S$, and $M|_{a_{-S}}$ is nonzero since the monomial $\prod_{j \in S} x_j$ remains with coefficient 1. The only multilinear polynomial expressing the all-0 function is the zero polynomial. Thus $M|_{a_{-S}}$ doesn't express the all-0 function, so $M(a) = M|_{a_{-S}}(a_S) = 1$ holds for at least one assignment $a_S$. In conclusion, $\text{weight}\big(E(M)\big) \geq 2^{\ell-e}$ since $M(a) = 1$ holds for at least $2^{\ell-e}$ many $a$.

**Exercise 9.19.d:** Consider any $a_{-S}$.

$$\sum_{a_S} M(a) \;=\; \sum_{a_S} \sum_{T \subseteq [\ell] : |T| \leq e} M_T \cdot \prod_{j \in T} a_j \;=\; \sum_{T \subseteq [\ell] : |T| \leq e} M_T \cdot \sum_{a_S} \prod_{j \in T} a_j$$

When $T = S$, we have $M_T \cdot \sum_{a_S} \prod_{j \in T} a_j = M_S$ because $\prod_{j \in T} a_j = 1$ for exactly one $a_S$, namely when $a_j = 1$ for all $j \in S$. When $T \neq S$, we have $M_T \cdot \sum_{a_S} \prod_{j \in T} a_j = 0$ because there exists $h \in S \setminus T$ (since $|S| = e \geq |T|$) and so

$$\sum_{a_S} \prod_{j \in T} a_j \;=\; \sum_{a_{S \setminus \{h\}}} \sum_{a_h} \prod_{j \in T} a_j$$

and for every $a_{S \setminus \{h\}}$, we have

$$\sum_{a_h} \prod_{j \in T} a_j \;=\; \prod_{j \in T} a_j + \prod_{j \in T} a_j = 0$$

since $h \notin T$.

**Exercise 9.19.e:** Assume $\text{dist}\big(r, E(M)\big) < 2^{\ell-e}/2$. This implies that fewer than $2^{\ell-e}/2$ of the $2^{\ell-e}$ many $a_{-S}$ have $r_a \neq M(a)$ for at least one $a_S$. Thus more than half of all $a_{-S}$ have $r_a = M(a)$ for all $a_S$. For such $a_{-S}$, we have $M_S = \sum_{a_S} M(a) = \sum_{a_S} r_a$ (Exercise 9.19.d). To recover $M_S$ from $r$:

for each $a_{-S}$: let $vote_{a_{-S}} \leftarrow \sum_{a_S} r_a$
let $M_S \leftarrow$ majority bit among $vote_{a_{-S}}$ over all $a_{-S}$

This takes poly$n$ time because it reads every bit of $r$ once, computes $n/2^e$ many parities of $2^e$ bits each, and then computes the majority of these $n/2^e$ bits.

**Exercise 9.19.f:** Given a received word $r$ with $\text{dist}\big(r, E(M)\big) < 2^{\ell-e}/2$, first recover $M_S$ for every $S$ with $|S| = e$ in poly $n$ time using Exercise 9.19.e. Then, consider the polynomial $M^*(x) = M(x) - \sum_{S\,:\,|S|=e} M_S \cdot \prod_{j\in S} x_j$ (which is $M$ with all monomials of total degree $e$ removed, so $M^*$ has total degree $\leq e-1$) and the updated received word $r^*$ where $r^*_x = r_x - \sum_{S\,:\,|S|=e} M_S \cdot \prod_{j\in S} x_j$. We have $\text{dist}\big(r^*, E(M^*)\big) = \text{dist}\big(r, E(M)\big) < 2^{\ell-e}/2 < 2^{\ell-(e-1)}/2$. Then, recurse (with $e-1$ in place of $e$) to recover the coefficients of $M^*$ (that is, the remaining coefficients of $M$) from $r^*$. For the base case when $e = 0$, the majority bit of the updated received word is $M$'s constant term $M_\emptyset$.

**Exercise 10.1:** Let $k = \lceil \log n \rceil + 3$. Define the random variables $X_0, X_1, \ldots, X_{k-1}$ as in the proof of Theorem 10.1. Recall that $X_j$ is a geometric random variable with heads probability $2^{-j}$.

$$\begin{aligned}
\mathbf{Pr}\big[|S| < k \text{ at the end}\big] &= \mathbf{Pr}\big[X_0 + X_1 + \cdots + X_{k-1} > n\big] \\
&\geq \mathbf{Pr}[X_{k-1} > n] \\
&= (1 - 2^{-(k-1)})^n \\
&\geq (1 - 2^{-(\log n + 2)})^n \\
&= (1 - 1/4n)^n \\
&\geq (1 - (1/4n)n) \\
&= 3/4
\end{aligned}$$

**Exercise 10.2:** On input $(G, s, t)$ where $G = (V, E)$ and $V = \{1, 2, \ldots, n\}$ and $n \geq 3$ and $s = 1$ and $t = n$:

```
for v ← 2, 3, ..., n − 1:
    if (s, v) ∈ E and (v, t) ∈ E: output the path s → v → t
reject
```

If this heuristic doesn't reject, then the output is correct.

$$\begin{aligned}
\mathbf{Pr}[\text{heuristic errs}] &\leq \mathbf{Pr}[\text{heuristic rejects}] \\
&= \mathbf{Pr}\Big[\forall v \in \{2, \ldots, n-1\} : (s, v) \notin E \text{ or } (v, t) \notin E\Big] \\
&= \textstyle\prod_{v=2}^{n-1}\Big(1 - \mathbf{Pr}\Big[(s, v) \in E \text{ and } (v, t) \in E\Big]\Big) \\
&= \big(1 - (1/2)^2\big)^{n-2} \\
&= (3/4)^{n-2} \\
&= 2^{-\Omega(n)}
\end{aligned}$$

This heuristic only needs log space to remember $v$ and pointers into the input.

**Exercise 10.3:** Denote the random 3-CNF by $\varphi = C_1 \vee \cdots \vee C_m$. For each assignment $a \in \{0,1\}^n$ and each clause index $j \in [m]$, we have $\mathbf{Pr}\big[C_j(a) = 1\big] = 7/8$ because no matter which 3 variables are involved and no matter what $a$ assigns those variables, 1 of the 8 patterns of negations doesn't satisfy $C_j$.

$$\begin{aligned}
\mathbf{Pr}[\varphi \text{ is satisfiable}] &\leq \textstyle\sum_{a\in\{0,1\}^n} \mathbf{Pr}\big[\varphi(a) = 1\big] && \text{(union bound)}\\
&= \textstyle\sum_{a\in\{0,1\}^n} \prod_{j=1}^{m} \mathbf{Pr}\big[C_j(a) = 1\big] && \text{(independence of clauses)}\\
&= \textstyle\sum_{a\in\{0,1\}^n} \prod_{j=1}^{m} 7/8\\
&= 2^n(7/8)^m\\
&\leq 2^n(7/8)^{5.2n}\\
&\leq 0.999^n
\end{aligned}$$

The clauses' sets of variables didn't need to be uniform and didn't need to be independent. The patterns of negations needed to be uniform and independent of each other and independent of the sets of variables.

**Exercise 10.4.a:** Say a valid input $x$ is *good* iff $\mathbf{Pr}_r\left[\Pi(x;r) \text{ errs}\right] \leq \delta$.

$$
\begin{aligned}
& \mathbf{Pr}_{x\sim D_N,\ r}\left[\Pi(x;r) \text{ succeeds}\right] \\
\geq\ & \mathbf{Pr}_{x\sim D_N,\ r}\left[\Pi(x;r) \text{ succeeds and } x \text{ is good}\right] \\
\geq\ & \mathbf{Pr}_{x\sim D_N}\left[x \text{ is good}\right] \cdot \mathbf{Pr}_{x\sim D_N,\ r}\left[\Pi(x;r) \text{ succeeds} \,\middle|\, x \text{ is good}\right] && \text{(chain rule)} \\
\geq\ & \mathbf{Pr}_{x\sim D_N}\left[x \text{ is good}\right] \cdot \min_{\text{good } x} \mathbf{Pr}_r\left[\Pi(x;r) \text{ succeeds}\right] \\
\geq\ & (1-\varepsilon)\cdot(1-\delta) \\
\geq\ & 1-\varepsilon-\delta
\end{aligned}
$$

**Exercise 10.4.b:** We prove the contrapositive. Say a valid input $x$ is *bad* iff $\mathbf{Pr}_r\left[\Pi(x;r) \text{ errs}\right] > \sqrt{\varepsilon}$, and assume $\mathbf{Pr}_{x\sim D_N}\left[x \text{ is bad}\right] > \sqrt{\varepsilon}$.

$$\begin{aligned}
& \mathbf{Pr}_{x\sim D_N,\, r}\left[\Pi(x;r) \text{ errs}\right] \\
\geq\ & \mathbf{Pr}_{x\sim D_N,\, r}\left[\Pi(x;r) \text{ errs and } x \text{ is bad}\right] \\
\geq\ & \mathbf{Pr}_{x\sim D_N}\left[x \text{ is bad}\right] \cdot \mathbf{Pr}_{x\sim D_N,\, r}\left[\Pi(x;r) \text{ errs} \,\middle|\, x \text{ is bad}\right] && \text{(chain rule)} \\
\geq\ & \mathbf{Pr}_{x\sim D_N}\left[x \text{ is bad}\right] \cdot \min_{\text{bad } x} \mathbf{Pr}_r\left[\Pi(x;r) \text{ errs}\right] \\
>\ & \sqrt{\varepsilon} \cdot \sqrt{\varepsilon} \\
=\ & \varepsilon
\end{aligned}$$

**Exercise 10.5:** Assume NP $\subseteq$ NC$^1$/poly. As in the proof of Theorem 10.4: Suppose $A \in$ NP by a verifier $V$ with time efficiency $\leq T = eN^d$ (for integers $e, d$) and word size $W$. On a valid input $x$ of size $N$, the potential witnesses are elements of $(\{0,1\}^W)^T = \{0,1\}^n$ where $n = TW$.

| FIND WITNESS FOR $V$ (call this problem $B$, with input size $N$) |
|---|
| Input: Valid input $x$ to $A$ |
| Output: $w \in \{0,1\}^n$ such that $V(x;w)$ accepts, or report that no such $w$ exists |

For each $m \in [n+1]$, let $h^m \colon \{0,1\}^{3n+1} \times \{0,1\}^n \to \{0,1\}^m$ be the pairwise uniform hash function from Lemma 8.9.

| (call this problem $C$, with input size $M$) |
|---|
| Input: $x, m, r, i$ where $x$ is a valid input of size $N$ to $A$, $m \in [n+1]$, $r \in \{0,1\}^{3n+1}$, and $i \in [n]$ |
| Output: Does there exist $w \in \{0,1\}^n$ such that $V(x;w)$ accepts and $h^m_r(w) = 0^m$ and $w_i = 1$? |

Since $C \in$ NP, we have $C \in$ NC$^1$/poly by a circuit family $D^C_1, D^C_2, \ldots$. We design an NC$^1$/poly-type circuit family $D^B_1, D^B_2, \ldots$ for $B$.

Consider any input size $N$ for $A$ and $B$, and the associated input size $M$ for $C$. For any $x$ such that $A(x) = 1$, say $(x, m, r)$ is *isolating* iff $h^m_r(w) = 0^m$ for exactly one $w$ such that $V(x;w)$ accepts. If $(x, m, r)$ is isolating, then $w = C(x, m, r, 1) \cdots C(x, m, r, n) = D^C_M(x, m, r, 1) \cdots D^C_M(x, m, r, n)$ is a witness for $x$. The proof of Theorem 8.10 showed that for uniformly random $m, r$:

$$\mathbf{Pr}_{m,r}\big[(x, m, r) \text{ is isolating}\big] \geq \tfrac{1}{n+1} \cdot \tfrac{2}{9} \geq 1/5n$$

Let $J = 5nNW$. If we sample $m^1, r^1, m^2, r^2, \ldots, m^J, r^J$ independently, then:

$$\begin{aligned}
&\mathbf{Pr}\big[\exists x : \big(A_N(x) = 1 \text{ and } \forall j : (x, m^j, r^j) \text{ is not isolating}\big)\big] \\
&\leq \textstyle\sum_{x : A_N(x)=1} \mathbf{Pr}\big[\forall j : (x, m^j, r^j) \text{ is not isolating}\big] && \text{(union bound)} \\
&= \textstyle\sum_{x : A_N(x)=1} \prod_{j=1}^{J} \mathbf{Pr}\big[(x, m^j, r^j) \text{ is not isolating}\big] && \text{(independence)} \\
&\leq \textstyle\sum_{x : A_N(x)=1} \prod_{j=1}^{J} (1 - 1/5n) && \text{(proof of Theorem 8.10)} \\
&\leq \textstyle\sum_{x : A_N(x)=1} \mathrm{e}^{-J/5n} && \text{(Fact 7.6)} \\
&\leq 2^{NW} \mathrm{e}^{-J/5n} \\
&< \mathrm{e}^{NW - J/5n} \\
&= 1
\end{aligned}$$

Thus there exist $m^1, r^1, \ldots, m^J, r^J$ (which we hardwire as nonuniform advice) such that for every $x$ with $A_N(x) = 1$, there exists $j$ such that $(x, m^j, r^j)$ is isolating.

For each $j$, define the circuit $E_j$ consisting of $D^C_M(x, m^j, r^j, 1), \ldots, D^C_M(x, m^j, r^j, n)$ in parallel, with $m^j, r^j$ and each $i \in [n]$ hardwired, with $x \in (\{0,1\}^W)^N$ as the input, and with $n$ output wires. We argued that if $A_N(x) = 1$ then there exists $j$ such that $E_j(x)$ is a witness for $x$.

Since $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{NC}^1/\mathrm{poly}$, we can turn $V$ itself into an $\mathsf{NC}^1/\mathrm{poly}$-type circuit $F$ such that $F(x,w) = V(x;w)$. Now, here is $D_N^B$: Take the circuits $E_1, \ldots, E_J$ in parallel. On top of that, have $J$ parallel copies of $F$, where the $j^{\text{th}}$ copy $F_j$ gets input $(x, E_j(x))$. On top of that, have an $\mathsf{NC}^1/\mathrm{poly}$-type circuit such that: If $F_j(x, E_j(x)) = 0$ for all $j$, then it outputs "reject." If $F_j(x, E_j(x)) = 1$ for at least one $j$, then it outputs $E_j(x)$ for the smallest $j$ such that $F_j(x, E_j(x)) = 1$.

If $D_N^B(x)$ doesn't reject then it outputs a witness for $x$. Thus if $A_N(x) = 0$ then $D_N^B(x)$ rejects, and if $A_N(x) = 1$ then for some $j$, $E_j(x)$ is a witness for $x$, so $F_j(x, E_j(x)) = 1$ and $D_N^B(x)$ outputs one such $E_j(x)$.

**Exercise 10.6:**

$$\begin{aligned} P(x_1, x_2) &= 11 \cdot (01 - x_1)x_2 \; + \; 01 \cdot x_1(01 - x_2) \\ &= 11 \cdot (x_1x_2 + x_2) \; + \; 01 \cdot (x_1x_2 + x_1) \\ &= 10x_1x_2 \; + \; x_1 \; + \; 11x_2 \end{aligned}$$

| $x_1$ | 00 | 00 | 00 | 00 | 01 | 01 | 01 | 01 | 10 | 10 | 10 | 10 | 11 | 11 | 11 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_2$ | 00 | 01 | 10 | 11 | 00 | 01 | 10 | 11 | 00 | 01 | 10 | 11 | 00 | 01 | 10 | 11 |
| codeword $P$ | 00 | 11 | 01 | 10 | 01 | 00 | 11 | 10 | 10 | 10 | 10 | 10 | 11 | 01 | 00 | 10 |

Thus $E(00, 11, 01, 00) = (00, 11, 01, 10, 01, 00, 11, 10, 10, 10, 10, 10, 11, 01, 00, 10)$. The gray entries are where the message appears directly in the codeword.

**Exercise 10.7.a:** $P(x_1, x_2, x_3) = 3 \cdot (1 - x_1)x_2(1 - x_3) = 3x_1x_2x_3 + 2x_1x_2 + 2x_2x_3 + 3x_2$

**Exercise 10.7.b:** $L(z) = (4,2,3) + z \cdot (1,2,0) = (4+z, 2+2z, 3)$
$L(0) = (4,2,3)$
$L(1) = (0,4,3)$
$L(2) = (1,1,3)$
$L(3) = (2,3,3)$
$L(4) = (3,0,3)$

**Exercise 10.7.c:**

$$\begin{aligned}(P \circ L)(z) &= P(L(z)) \\ &= 3(4+z)(2+2z)3 \ + \ 2(4+z)(2+2z) \ + \ 2(2+2z)3 \ + \ 3(2+2z) \\ &= (3z^2+2) \ + \ (4z^2+1) \ + \ (2z+2) \ + \ (z+1) \\ &= 2z^2+3z+1\end{aligned}$$

As a sanity check:

$$\begin{aligned}(P \circ L)(0) &= 1 = P(4,2,3) \\ (P \circ L)(1) &= 1 = P(0,4,3) \\ (P \circ L)(2) &= 0 = P(1,1,3) \\ (P \circ L)(3) &= 3 = P(2,3,3) \\ (P \circ L)(4) &= 0 = P(3,0,3)\end{aligned}$$

**Exercise 10.8.a:** Assume $(1-d/q)/6-1/q>0$ since 0 corruption can be handled with one query. Here's the local decoder:

given message index $a\in\{0,1\}^d$ and received word $r:\mathbb{F}^d\to\mathbb{F}$:
  sample $b\in\mathbb{F}^d$ uniformly at random
  let $L:\mathbb{F}\to\mathbb{F}^d$ be the parameterized line with anchor $a$ and direction $b$
  for each $z\in\mathbb{F}$: query $r(L(z))$
  use the polynomial code decoder to find a polynomial $Q$ of degree $\leq d$ such that:
    $Q(z)\neq r(L(z))$ for $<(q-d)/2$ many $z\in\mathbb{F}$ (if possible)
  output $Q(0)$

For

- every message $m:\{0,1\}^d\to\mathbb{F}$ with codeword $P:\mathbb{F}^d\to\mathbb{F}$, and
- every received word $r:\mathbb{F}^d\to\mathbb{F}$ with $\text{dist}(r,P)\leq q^d\big((1-d/q)/6-1/q\big)$, and
- every message index $a\in\{0,1\}^d$,

we have:

- If $Q=P\circ L$ then the decoder correctly outputs $Q(0)=P(L(0))=P(a)=m(a)$.
- If $r(L(z))\neq P(L(z))$ for $<(q-d)/2$ many $z\in\mathbb{F}$, then $Q=P\circ L$ since $\deg(Q)\leq d$ and $\deg(P\circ L)\leq d$ and they disagree on $<(q-d)/2+(q-d)/2=q-d$ points.
- $\mathbf{Pr}\big[r(L(z))\neq P(L(z))\big]=\text{dist}(r,P)/q^d\leq(1-d/q)/6-1/q$ for each $z\in\mathbb{F}\setminus\{0\}$ since $L(z)$ is uniformly random. Letting the random variable $X$ be the number of $z\in\mathbb{F}$ such that $r(L(z))\neq P(L(z))$, we have $\mathbf{E}[X]\leq(q-1)\big((1-d/q)/6-1/q\big)+1\leq(q-d)/6$ (the $+1$ is for $z=0$) and thus $\mathbf{Pr}\big[X\geq(q-d)/2\big]\leq\mathbf{E}[X]/\big((q-d)/2\big)\leq 1/3$ (Lemma 7.13).

Thus $\mathbf{Pr}\big[\text{decoder outputs } m(a)\big]\geq\mathbf{Pr}\big[Q=P\circ L\big]\geq\mathbf{Pr}\big[X<(q-d)/2\big]\geq 2/3$.

**Exercise 10.8.b:** If $P$ has canonical form

$$P(x_1, \ldots, x_d) = \sum_{S \subseteq [d]} P_S \prod_{i \in S} x_i$$

(where each $P_S \in \mathbb{F}$ is a coefficient) and

$$L(z) = a + z \cdot b + z^2 \cdot c = (a_1 + z b_1 + z^2 c_1, \ldots, a_d + z b_d + z^2 c_d)$$

then

$$(P \circ L)(z) = P(L(z)) = \sum_{S \subseteq [d]} P_S \prod_{i \in S} (a_i + z b_i + z^2 c_i)$$

and so:

$$\begin{aligned} \deg(P \circ L) &\leq \max_S \left( \deg\left( \prod_{i \in S} (a_i + z b_i + z^2 c_i) \right) \right) = \max_S \left( \sum_{i \in S} \deg(a_i + z b_i + z^2 c_i) \right) \\ &\leq \max_S \left( \sum_{i \in S} 2 \right) = \max_S (2|S|) \leq 2d \end{aligned}$$

**Exercise 10.8.c:** Consider any $y, z \in \mathbb{F} \setminus \{0\}$ with $y \neq z$ and any $e, f \in \mathbb{F}^d$. We claim that

$$\mathbf{Pr}_{b,c}\big[L(y) = e \text{ and } L(z) = f\big] \;=\; q^{-2d}$$

because "$L(y) = e$ and $L(z) = f$" holds for exactly one of the $q^{2d}$ many outcomes $(b, c)$. For each $i \in [d]$,

$$a_i + y\,b_i + y^2 c_i = e_i \qquad \text{and} \qquad a_i + z\,b_i + z^2 c_i = f_i$$

holds for exactly one $(b_i, c_i)$ because there's exactly one univariate polynomial $R$ of degree $\leq 2$ such that

$$R(0) = a_i \qquad \text{and} \qquad R(y) = e_i \qquad \text{and} \qquad R(z) = f_i$$

since a univariate polynomial of degree $\leq 2$ is uniquely determined by its values on any three distinct points, such as $0, y, z$.

**Exercise 10.8.d:** Assume $(1-2d/q)/2-\sqrt{3/q}>0$ since 0 corruption can be handled with one query. In particular, this implies $q \geq 16$. Here's the local decoder:

given message index $a \in \{0,1\}^d$ and received word $r : \mathbb{F}^d \to \mathbb{F}$:
  sample $b, c \in \mathbb{F}^d$ uniformly at random
  let $L : \mathbb{F} \to \mathbb{F}^d$ be the parameterized parabola associated with $a, b, c$
  for each $z \in \mathbb{F}$: query $r(L(z))$
  use the polynomial code decoder to find a polynomial $Q$ of degree $\leq 2d$ such that:
    $Q(z) \neq r(L(z))$ for $< (q-2d)/2$ many $z \in \mathbb{F}$ (if possible)
  output $Q(0)$

For

- every message $m : \{0,1\}^d \to \mathbb{F}$ with codeword $P : \mathbb{F}^d \to \mathbb{F}$, and
- every received word $r : \mathbb{F}^d \to \mathbb{F}$ with $\text{dist}(r,P) \leq q^d\big((1-2d/q)/2-\sqrt{3/q}\big)$, and
- every message index $a \in \{0,1\}^d$,

we have:

- If $Q = P \circ L$ then the decoder correctly outputs $Q(0) = P(L(0)) = P(a) = m(a)$.
- If $r(L(z)) \neq P(L(z))$ for $< (q-2d)/2$ many $z \in \mathbb{F}$, then $Q = P \circ L$ since $\deg(Q) \leq 2d$ and $\deg(P \circ L) \leq 2d$ (Exercise 10.8.b) and they disagree on $< (q-2d)/2 + (q-2d)/2 = q-2d$ points.
- Let $B = \big\{x \in \mathbb{F}^d \,:\, r(x) \neq P(x)\big\}$ be the symbol change ("bad") locations. Let the random variable $X$ be the number of $z \in \mathbb{F}$ such that $L(z) \in B$, and let the random variable $Y$ be the number of $z \in \mathbb{F} \setminus \{0\}$ such that $L(z) \in B$, so $X \leq Y + 1$. Since $h^a$ is pairwise uniform (Exercise 10.8.c), hash mixing (Lemma 8.15) says that for all $\varepsilon > 0$:

$$\mathbf{Pr}\Big[\big|Y/(q-1) - |B|/q^d\big| \geq \varepsilon\Big] \;\leq\; |B|/(q-1)q^d\varepsilon^2$$

  Since $|B| = \text{dist}(r,P) \leq q^d\big((1-2d/q)/2-\sqrt{3/q}\big)$:

$$\begin{aligned}
\mathbf{Pr}\big[X \geq (q-2d)/2\big] &\leq \mathbf{Pr}\big[Y \geq (q-2d)/2 - 1\big] \\
&= \mathbf{Pr}\big[Y/q \geq (1-2d/q)/2 - 1/q\big] \\
&\leq \mathbf{Pr}\Big[Y/(q-1) \geq \big(|B|/q^d + \sqrt{3/q}\big) - 1/q\Big] \\
&\leq \mathbf{Pr}\Big[\big|Y/(q-1) - |B|/q^d\big| \geq \sqrt{2/q}\Big] \\
&\leq \big((1-2d/q)/2 - \sqrt{3/q}\big)\big/(q-1)\sqrt{2/q}^2 \\
&\leq (1/2)/(q-1)(2/q) \\
&= q/4(q-1) \\
&\leq 1/3
\end{aligned}$$

Thus $\mathbf{Pr}\big[\text{decoder outputs } m(a)\big] \geq \mathbf{Pr}\big[Q = P \circ L\big] \geq \mathbf{Pr}\big[X < (q-2d)/2\big] \geq 2/3$.

**Exercise 10.8.e:** Assume $1-2d/(q-1)-\sqrt{4/q}>0$ since 0 corruption can be handled with one query. In particular, this implies $q \geq 8$. Here's the local decoder:

given message index $a \in \{0,1\}^d$ and received word $r\colon \mathbb{F}^d \to \mathbb{F} \cup \{\perp\}$:
  sample $b, c \in \mathbb{F}^d$ uniformly at random
  let $L\colon \mathbb{F} \to \mathbb{F}^d$ be the parameterized parabola associated with $a, b, c$
  for each $z \in \mathbb{F}$: query $r(L(z))$
  if $r(L(z)) \neq \perp$ for $\leq 2d$ many $z \in \mathbb{F}$: output $\perp$
  let $Z \subseteq \{z \in \mathbb{F} : r(L(z)) \neq \perp\}$ be an arbitrary set of size $2d+1$
  interpolate to find a polynomial $Q$ of degree $\leq 2d$ such that $Q(z) = r(L(z))$ for all $z \in Z$
  output $Q(0)$

For

- every message $m\colon \{0,1\}^d \to \mathbb{F}$ with codeword $P\colon \mathbb{F}^d \to \mathbb{F}$, and
- every received word $r\colon \mathbb{F}^d \to \mathbb{F} \cup \{\perp\}$ such that $r(x) \in \{P(x), \perp\}$ for all $x$ and $r(x) = \perp$ for $\leq q^d\big(1-2d/(q-1)-\sqrt{4/q}\big)$ many $x$, and
- every message index $a \in \{0,1\}^d$,

we have:

- If $Q = P \circ L$ then the decoder correctly outputs $Q(0) = P(L(0)) = P(a) = m(a)$.
- If $r(L(z)) \neq \perp$ for $\geq 2d+1$ many $z \in \mathbb{F}$, then $Q = P \circ L$ since $\deg(Q) \leq 2d$ and $\deg(P \circ L) \leq 2d$ (Exercise 10.8.b) and they agree with $r \circ L$ and thus with each other on the $2d+1$ points $z \in Z$.
- Let $G = \{x \in \mathbb{F}^d : r(x) \neq \perp\}$ be the "good" locations. Let the random variable $Y$ be the number of $z \in \mathbb{F} \setminus \{0\}$ such that $L(z) \in G$. Since $h^a$ is pairwise uniform (Exercise 10.8.c), hash mixing (Lemma 8.15) says that for all $\varepsilon > 0$:

$$\mathbf{Pr}\Big[\big|Y/(q-1) - |G|/q^d\big| \geq \varepsilon\Big] \;\leq\; |G|/(q-1)q^d\varepsilon^2$$

Since $q^d \geq |G| \geq q^d\big(2d/(q-1)+\sqrt{4/q}\big)$:

$$\begin{aligned}
\mathbf{Pr}\big[Y \leq 2d\big] &= \mathbf{Pr}\big[Y/(q-1) \leq 2d/(q-1)\big] \\
&\leq \mathbf{Pr}\Big[Y/(q-1) \leq |G|/q^d - \sqrt{4/q}\Big] \\
&\leq \mathbf{Pr}\Big[\big|Y/(q-1) - |G|/q^d\big| \geq \sqrt{4/q}\Big] \\
&\leq 1/(q-1)\sqrt{4/q}^2 \\
&= q/4(q-1) \\
&\leq 1/3
\end{aligned}$$

Thus $\mathbf{Pr}\big[\text{decoder outputs } m(a)\big] \geq \mathbf{Pr}\big[Q = P \circ L\big] \geq \mathbf{Pr}\big[Y \geq 2d+1\big] \geq 2/3$.

**Exercise 10.9:** For any $K$, consider the smallest $\ell \geq \log(8K)$ for defining $\mathbb{F} = \mathbb{F}_q$ where $q = 2^\ell \geq 8K$ using Theorem 9.19.*(ii)*. Consider the multilinear code $E_{\text{out}}$ where:

- The message is $m\colon \{0,1\}^K \to \mathbb{F}$.
- The codeword is $P\colon \mathbb{F}^K \to \mathbb{F}$.

Consider the dot product code $E_{\text{in}}\colon \{0,1\}^\ell \to \{0,1\}^q$. Define $N = \ell K + \ell$ and let $E$ be the concatenated code $E_{\text{in}} \circ E_{\text{out}}$ except that we only allow messages to $E_{\text{out}}$ that only contains 0s and 1s anyway (so the message doesn't need to be flattened):

- The message is $m\colon \{0,1\}^K \to \{0,1\}$.
- The codeword is $c\colon \{0,1\}^N \to \{0,1\}$.

Viewing the codeword as $c\colon \mathbb{F}^K \times \mathbb{F} \to \{0,1\}$ (with "unflattened" domain):

$$c(x,j) \;=\; P(x) \odot j$$

Encoding $E$ in poly$K$ space is straightforward, like in Theorem 10.11. Let $D_{\text{out}}$ be a local decoder for $E_{\text{out}}$ that handles $q^K/7 \leq q^K\big((1-K/q)/6 - 1/q\big)$ symbol changes, using $q$ symbol queries, with error probability $1/3$ (Exercise 10.8.a). Let $D_{\text{in}}$ be a (global) decoder for $E_{\text{in}}$ that handles $< q/4$ bit flips (§9.1.2). Here's a local decoder $D$ for $E$:

given message index $a \in \{0,1\}^K$ and received word $r\colon \mathbb{F}^K \times \mathbb{F} \to \{0,1\}$:
  define $r_{\text{out}}\colon \mathbb{F}^K \to \mathbb{F}$ by $r_{\text{out}}(x) = D_{\text{in}}(r_{\text{in},x})$ where $r_{\text{in},x} \in \{0,1\}^q$ is $r(x,j)$ for all $j \in \mathbb{F}$
  run $D_{\text{out}}(a, r_{\text{out}})$:
    when it queries $r_{\text{out}}(x)$:
      do the $q$ queries $r(x,j)$ for all $j \in \mathbb{F}$ to obtain $r_{\text{in},x}$
      run $D_{\text{in}}(r_{\text{in},x})$
  output whatever $D_{\text{out}}(a, r_{\text{out}})$ outputs

$D$ runs in poly$K$ time and handles $(q^K/7) \cdot (q/4) = 2^N/28$ bit flips (by the solution to Exercise 9.17), using $q^2$ bit queries, with error probability $1/3$.

In the proof of Theorem 10.5, using this $E$ (instead of the adjusted multilinear code) shows that if $(\mathsf{PSPACE}, \mathsf{U}) \subseteq \mathsf{HEUR}_{1/28}\mathsf{P}$ then $\mathsf{PSPACE} = \mathsf{BPP}$. (Using Exercise 10.8.d instead of Exercise 10.8.a improves $1/28$ to any constant $< 1/8$.)

**Exercise 10.10:** Assume $f$ is $(s,\alpha)$-hard. Consider any $k$ and $\delta > 0$. We'll show that for every circuit $C : (\{0,1\}^m)^k \to \{0,1\}$ of size $\leq cs\delta^2(1-\alpha)^2$:

$$\mathbf{Pr}_{x_1,\ldots,x_k}\big[C(x_1,\ldots,x_k) = f^{\oplus k}(x_1,\ldots,x_k)\big] \;<\; \big(1+2((1+\alpha)/2)^k+\delta\big)/2$$

For every $i \in [k]$ and assignment $x_{-i}$ to all components except the $i^{\text{th}}$, define the circuit $D_{i,x_{-i}} : \{0,1\}^m \to \{0,1\}$ such that $D_{i,x_{-i}}(x_i)$ evaluates $C(x)$ (where $x$ is the combination of $x_i$ and $x_{-i}$) and outputs $C(x) \oplus \bigoplus_{j\neq i} f(x_j)$.

Let $H \subseteq \{0,1\}^m$ be such that $|H|/2^m \geq (1-\alpha)/2$ and $f$ is $\big(H, cs\delta^2(1-\alpha)^2, \delta\big)$-hard. Let $\overline{H} = \{0,1\}^m \setminus H$. Partition $(\{0,1\}^m)^k$ into events $E_0 \cup E_1 \cup \cdots \cup E_k$ where $E_0 = \big\{x : x_j \notin H \text{ for all } j\big\}$ and for each $i \in [k]$, $E_i = \big\{x : x_i \in H \text{ and } x_j \notin H \text{ for all } j < i\big\}$.

$$\mathbf{Pr}_x[E_0] \;=\; \textstyle\prod_{j=1}^k \mathbf{Pr}_{x_j}[x_j \notin H] \;\leq\; \prod_{j=1}^k (1+\alpha)/2 \;=\; ((1+\alpha)/2)^k$$

For each $i \in [k]$,

$$\begin{aligned}
\mathbf{Pr}_x\big[C(x) = f^{\oplus k}(x) \,\big|\, E_i\big] \;&=\; \mathbf{Pr}_{x\in\overline{H}^{i-1}\times H\times(\{0,1\}^m)^{k-i}}\big[C(x)\oplus\textstyle\bigoplus_{j\neq i} f(x_j) = f(x_i)\big] \\
&=\; \mathbf{E}_{x_{-i}\in\overline{H}^{i-1}\times(\{0,1\}^m)^{k-i}}\Big[\mathbf{Pr}_{x_i\in H}\big[D_{i,x_{-i}}(x_i) = f(x_i)\big]\Big] \\
&<\; \mathbf{E}_{x_{-i}\in\overline{H}^{i-1}\times(\{0,1\}^m)^{k-i}}\big[(1+\delta)/2\big] \\
&=\; (1+\delta)/2
\end{aligned}$$

since $D_{i,x_{-i}}$ has size $\leq$ (size of $C$) $\leq cs\delta^2(1-\alpha)^2$ for every $i$ and $x_{-i}$. By the law of total probability:

$$\begin{aligned}
\mathbf{Pr}_x\big[C(x) = f^{\oplus k}(x)\big] \;&=\; \textstyle\sum_{i=0}^k \mathbf{Pr}_x[E_i]\cdot\mathbf{Pr}_x\big[C(x) = f^{\oplus k}(x)\,\big|\,E_i\big] \\
&<\; ((1+\alpha)/2)^k\cdot 1 + \textstyle\sum_{i=1}^k \mathbf{Pr}_x[E_i]\cdot(1+\delta)/2 \\
&\leq\; ((1+\alpha)/2)^k + (1+\delta)/2 \\
&=\; \big(1+2((1+\alpha)/2)^k+\delta\big)/2
\end{aligned}$$

**Exercise 10.11:** Assume $f$ is $(s,\alpha)$-hard. For all $\ell$ and small enough $\delta > 0$, $f^{\oplus\ell}$ is $\big(s\delta^3(1-\alpha)^3, \alpha^\ell + \delta\big)$-hard (Lemma 10.15). Consider any $k$ and small enough $\delta > 0$. We'll show that for every circuit $C\colon (\{0,1\}^m)^k \to \{0,1\}^k$ of size $\leq s\delta^3(1-\alpha)^3 - 3k$, letting

$$\varepsilon \;=\; \mathbf{Pr}_{x_1,\ldots,x_k}\big[C(x_1,\ldots,x_k) = f^k(x_1,\ldots,x_k)\big]$$

we have $\varepsilon < \alpha^{k/3} + \delta + 2\cdot 0.95^k$.

For any $S \subseteq [k]$ of size $\ell$ and any assignment $x \in (\{0,1\}^m)^k$, let $y \in (\{0,1\}^m)^\ell$ denote the components of $x$ indexed by $S$, and let $z \in (\{0,1\}^m)^{k-\ell}$ denote the components of $x$ indexed by $[k] \setminus S$. For every $S$ and $z$, define the circuit $D_{S,z}\colon (\{0,1\}^m)^\ell \to \{0,1\}$ such that $D_{S,z}(y)$ evaluates $C(x)$ (where $x$ is the combination of $y$ and $z$) and outputs the parity of the bits of $C(x)$ indexed by $S$.

Consider any assignment $x$. For a uniformly random $S \subseteq [k]$, letting $\ell = |S|$ and $u \in \{0,1\}^k$ be $S$'s characteristic bit string and $\odot$ be the dot product over $\mathbb{Z}_2$,

$$\begin{aligned}
\mathbf{Pr}_S\big[D_{S,z}(y) = f^{\oplus\ell}(y)\big] \;&=\; \mathbf{Pr}_S\big[\textstyle\bigoplus_{i\in S} C(x)_i = \bigoplus_{i\in S} f(x_i)\big] \\
&=\; \mathbf{Pr}_u\big[C(x)\odot u = f^k(x)\odot u\big] \\
&=\; \begin{cases} 1 & \text{if } C(x) = f^k(x) \\ 1/2 & \text{otherwise} \end{cases}
\end{aligned}$$

by the random subsum principle (Lemma 7.2.*(ii)*). Say $x$ is *good* iff $C(x) = f^k(x)$. By the law of total probability:

$$\begin{aligned}
\mathbf{Pr}_{S,x}\big[D_{S,z}(y) = f^{\oplus\ell}(y)\big] \;&=\; \mathbf{Pr}_x\big[x \text{ is good}\big]\cdot \mathbf{Pr}_{S,x}\big[D_{S,z}(y) = f^{\oplus\ell}(y)\,\big|\,x \text{ is good}\big] + \\
&\qquad \mathbf{Pr}_x\big[x \text{ is bad}\big]\cdot \mathbf{Pr}_{S,x}\big[D_{S,z}(y) = f^{\oplus\ell}(y)\,\big|\,x \text{ is bad}\big] \\
&=\; \varepsilon\cdot 1 + (1-\varepsilon)\cdot(1/2) \\
&=\; (1+\varepsilon)/2
\end{aligned}$$

By Lemma 9.4:

$$\mathbf{Pr}_S\big[|S| < k/3\big] \;\leq\; (2^k)^{H(1/3)}/2^k \;=\; (2^{H(1/3)-1})^k \;\leq\; 0.95^k$$

Thus:

$$\begin{aligned}
\mathbf{E}_{S,z}\Big[\mathbf{Pr}_y\big[D_{S,z}(y) = f^{\oplus\ell}(y) \text{ and } |S| \geq k/3\big]\Big] \;&=\; \mathbf{Pr}_{S,x}\big[D_{S,z}(y) = f^{\oplus\ell}(y) \text{ and } |S| \geq k/3\big] \\
&\geq\; \mathbf{Pr}_{S,x}\big[D_{S,z}(y) = f^{\oplus\ell}(y)\big] - \mathbf{Pr}_S\big[|S| < k/3\big] \\
&\geq\; (1+\varepsilon)/2 - 0.95^k
\end{aligned}$$

By the probabilistic method, there exist $S$ of size $\ell \geq k/3$ and $z$ such that:

$$\mathbf{Pr}_y\big[D_{S,z}(y) = f^{\oplus\ell}(y)\big] \;\geq\; (1+\varepsilon)/2 - 0.95^k$$

Since $D_{S,z}$ has size $\leq$ (size of $C$) $+ 3k \leq s\delta^3(1-\alpha)^3$, Lemma 10.15 says that

$$(1+\varepsilon)/2 - 0.95^k \;<\; (1+\alpha^\ell+\delta)/2$$

and thus $\varepsilon < \alpha^{k/3} + \delta + 2\cdot 0.95^k$.

**Exercise 11.1:** First proof: There exists $g : T \to \{0,1\}$ such that:

$$\begin{aligned}\Delta\big(f(D_0), f(D_1)\big) &= \Big|\mathbf{Pr}\big[g(f(D_0)) = 1\big] - \mathbf{Pr}\big[g(f(D_1)) = 1\big]\Big| \\ &= \Big|\mathbf{Pr}\big[(g \circ f)(D_0) = 1\big] - \mathbf{Pr}\big[(g \circ f)(D_1) = 1\big]\Big| \\ &\leq \Delta(D_0, D_1)\end{aligned}$$

Second proof: For any $E \subseteq T$, let $f^{-1}(E) = \big\{s \in S \,:\, f(s) \in E\big\}$. We have

$$f(D_0)(E) \;=\; \sum_{t \in E} f(D_0)(t) \;=\; \sum_{t \in E} \sum_{s\,:\,f(s)=t} D_0(s) \;=\; \sum_{s \in f^{-1}(E)} D_0(s) \;=\; D_0(f^{-1}(E))$$

and $f(D_1)(E) = D_1(f^{-1}(E))$.

$$\begin{aligned}\Delta\big(f(D_0), f(D_1)\big) &= \max_{E \subseteq T} \big|f(D_0)(E) - f(D_1)(E)\big| \\ &= \max_{E \subseteq T} \big|D_0(f^{-1}(E)) - D_1(f^{-1}(E))\big| \\ &\leq \Delta(D_0, D_1)\end{aligned}$$

Third proof:

$$\begin{aligned}2\Delta\big(f(D_0), f(D_1)\big) &= \sum_{t \in T} \big|f(D_0)(t) - f(D_1)(t)\big| && \text{(Lemma 11.1)} \\ &= \sum_{t \in T} \Big|\sum_{s\,:\,f(s)=t} D_0(s) - \sum_{s\,:\,f(s)=t} D_1(s)\Big| \\ &= \sum_{t \in T} \Big|\sum_{s\,:\,f(s)=t} \big(D_0(s) - D_1(s)\big)\Big| \\ &\leq \sum_{t \in T} \sum_{s\,:\,f(s)=t} \big|D_0(s) - D_1(s)\big| && \text{(Lemma 0.28)} \\ &= \sum_{s \in S} \big|D_0(s) - D_1(s)\big| \\ &= 2\Delta(D_0, D_1) && \text{(Lemma 11.1)}\end{aligned}$$

**Exercise 11.2:** Let $E = \{s : D_1(s) < D_0(s)\}$ and $E' = \{s : D_1(s) < D_0(s)/2\} \subseteq E$. For each $s \in E'$, we have $D_1(s) < D_0(s) - D_1(s)$.

$$\begin{aligned} D_1(E') &= \textstyle\sum_{s \in E'} D_1(s) \\ &\leq \textstyle\sum_{s \in E'} \big(D_0(s) - D_1(s)\big) \\ &\leq \textstyle\sum_{s \in E} \big(D_0(s) - D_1(s)\big) \\ &= D_0(E) - D_1(E) \\ &= \Delta(D_0, D_1) \qquad \text{(Lemma 11.1)} \end{aligned}$$

**Exercise 11.3:** For every $f \in F$:

$$\begin{aligned}
& \left|\mathbf{Pr}\left[f(G(D))=1\right]-\mathbf{Pr}\left[f(U_n)=1\right]\right| \\
= & \left|\left(\mathbf{Pr}\left[f(G(D))=1\right]-\mathbf{Pr}\left[f(G(U_\ell))=1\right]\right)+\left(\mathbf{Pr}\left[f(G(U_\ell))=1\right]-\mathbf{Pr}\left[f(U_n)=1\right]\right)\right| \\
\leq & \left|\mathbf{Pr}\left[f(G(D))=1\right]-\mathbf{Pr}\left[f(G(U_\ell))=1\right]\right|+\left|\mathbf{Pr}\left[f(G(U_\ell))=1\right]-\mathbf{Pr}\left[f(U_n)=1\right]\right| \\
\leq & \left|\mathbf{Pr}\left[(f\circ G)(D)=1\right]-\mathbf{Pr}\left[(f\circ G)(U_\ell)=1\right]\right|+\varepsilon \\
\leq & \ \Delta(D,U_\ell)+\varepsilon \\
\leq & \ \delta+\varepsilon
\end{aligned}$$

**Exercise 11.4.a:** Consider any $f\colon \{0,1\}^n \to \{0,1\}$. By the proof of Lemma 2.9, $f$ can be expressed as a sum (over $\mathbb{R}$) of terms $f = \sum_{a\,:\,f(a)=1} T_a$ where $T_a$ accepts $a$ and only $a$.

$$\begin{aligned}
&\left|\mathbf{Pr}\left[f(G(U_\ell))=1\right]-\mathbf{Pr}\left[f(U_n)=1\right]\right| \\
=\ &\left|\mathbf{E}\left[\textstyle\sum_{a\,:\,f(a)=1} T_a(G(U_\ell))\right]-\mathbf{E}\left[\sum_{a\,:\,f(a)=1} T_a(U_n)\right]\right| \\
=\ &\left|\textstyle\sum_{a\,:\,f(a)=1}\left(\mathbf{E}\left[T_a(G(U_\ell))\right]-\mathbf{E}\left[T_a(U_n)\right]\right)\right| && \text{(linearity of expectation)} \\
\leq\ &\textstyle\sum_{a\,:\,f(a)=1}\left|\mathbf{E}\left[T_a(G(U_\ell))\right]-\mathbf{E}\left[T_a(U_n)\right]\right| && \text{(Lemma 0.28)} \\
\leq\ &\textstyle\sum_{a\,:\,f(a)=1} 2(\varepsilon/2^{n+1}) && \text{(Theorem 11.11)} \\
\leq\ &\varepsilon
\end{aligned}$$

**Exercise 11.4.b:** Suppose $f : \{0,1\}^n \to \{0,1\}$ is computed by a decision tree with $\leq m$ leaves. By the proof of Theorem 6.21, $f$ can be expressed as a sum (over $\mathbb{R}$) of terms $f = \sum_{\text{leaf } v \text{ labeled } 1} T_v$ where $T_v$ accepts inputs that lead to $v$.

$$\begin{aligned}
&\left|\mathbf{Pr}\left[f(G(U_\ell)) = 1\right] - \mathbf{Pr}\left[f(U_n) = 1\right]\right| \\
=\ &\left|\mathbf{E}\left[\textstyle\sum_{\text{leaf } v \text{ labeled } 1} T_v(G(U_\ell))\right] - \mathbf{E}\left[\sum_{\text{leaf } v \text{ labeled } 1} T_v(U_n)\right]\right| \\
=\ &\left|\textstyle\sum_{\text{leaf } v \text{ labeled } 1}\left(\mathbf{E}\left[T_v(G(U_\ell))\right] - \mathbf{E}\left[T_v(U_n)\right]\right)\right| && \text{(linearity of expectation)} \\
\leq\ &\textstyle\sum_{\text{leaf } v \text{ labeled } 1}\left|\mathbf{E}\left[T_v(G(U_\ell))\right] - \mathbf{E}\left[T_v(U_n)\right]\right| && \text{(Lemma 0.28)} \\
\leq\ &\textstyle\sum_{\text{leaf } v \text{ labeled } 1} 2(\varepsilon/2m) && \text{(Theorem 11.11)} \\
\leq\ &\varepsilon
\end{aligned}$$

**Exercise 11.4.c:** Let $\ell \geq \log n$ be the least integer for defining $\mathbb{F} = \mathbb{F}_{2^\ell}$ with the simple irreducible polynomial from Theorem 9.19*(ii)*. Identify $[n]$ with a subset of $\mathbb{F}$. Define $G^*: \mathbb{F}^d \to \mathbb{F}^n$ by $G^*(s_{d-1}, \ldots, s_0)_i = s_{d-1} i^{d-1} + s_{d-2} i^{d-2} + \cdots + s_1 i + s_0$ for each $i \in [n]$. If $s = (s_{d-1}, \ldots, s_0)$ is uniformly random and $I \subseteq [n]$ has size $d$, then $G^*(s)_I$ is uniformly random over $\mathbb{F}^d$, as in the solution to Exercise 8.2. Also, $G^*$ is just the polynomial code $E_{2^\ell,d}$ from §9.5.2, possibly with some codeword coordinates removed, so $G^*$ is linear over $\mathbb{F}$, and thus linear over $\mathbb{Z}_2$, for the same reason as $E_{2^\ell,d}$.

For $k = d\ell = O(d \log n)$, obtain $G: \{0,1\}^k \to \{0,1\}^n$ from $G^*$ by flattening the seed and keeping only one bit (it doesn't matter which one) of each output coordinate. This $G$ is linear over $\mathbb{Z}_2$ (since $G^*$ is) with a generator matrix $H$. If $s \sim U_k$ then $G(s)_I \sim U_d$ for every $I \subseteq [n]$ of size $d$. Thus for every $f: \{0,1\}^n \to \{0,1\}$ that depends on only $d$ variables, say $f(x) = f'(x_I)$ for some $f': \{0,1\}^d \to \{0,1\}$ and $I \subseteq [n]$ of size $d$:

$$\mathbf{Pr}\big[f(G(U_k)) = 1\big] \;=\; \mathbf{Pr}\big[f'(G(U_k)_I) = 1\big] \;=\; \mathbf{Pr}\big[f'(U_d) = 1\big] \;=\; \mathbf{Pr}\big[f(U_n) = 1\big]$$

More generally, suppose $f: \{0,1\}^n \to \{0,1\}$ is computed by a depth-$d$ decision tree. Without loss of generality, every leaf is at depth exactly $d$. By the proof of Theorem 6.21, $f$ can be expressed as a sum (over $\mathbb{R}$) of width-$d$ terms $f = \sum_{\text{leaf } v \text{ labeled } 1} T_v$ where $T_v$ accepts inputs that lead to $v$.

$$\begin{aligned}
\mathbf{Pr}\big[f(G(U_k)) = 1\big] &= \mathbf{E}\Big[\textstyle\sum_{\text{leaf } v \text{ labeled } 1} T_v(G(U_k))\Big] \\
&= \textstyle\sum_{\text{leaf } v \text{ labeled } 1} \mathbf{E}\big[T_v(G(U_k))\big] \\
&= \textstyle\sum_{\text{leaf } v \text{ labeled } 1} \mathbf{E}\big[T_v(U_n)\big] \\
&= \mathbf{E}\Big[\textstyle\sum_{\text{leaf } v \text{ labeled } 1} T_v(U_n)\Big] \\
&= \mathbf{Pr}\big[f(U_n) = 1\big]
\end{aligned}$$

**Exercise 11.4.d:** (This $\ell$ is not the same $\ell$ from the solution to Exercise 11.4.c.)

Let $H$ be the $k \times n$ matrix for $G_2$, so $G_2(s) = sH$. Let $H_i$ be $H$'s $i^{\text{th}}$ column. Suppose $I \subseteq [n]$ and $|I| = d$. (The same argument works if $|I| < d$.) First, we claim that $\oplus_I \circ G_2 = \oplus_J : \{0,1\}^k \to \{0,1\}$ for some $J \subseteq [k]$. This is because

$$(\oplus_I \circ G_2)(s) \;=\; \textstyle\sum_{i \in I} G_2(s)_i \;=\; \sum_{i \in I} (sH)_i \;=\; \sum_{i \in I} s \odot H_i \;=\; s \odot \left(\sum_{i \in I} H_i\right) \;=\; \oplus_J(s)$$

for some $J \subseteq [k]$, with arithmetic over $\mathbb{Z}_2$. Next, we claim that $\oplus_I(U_n)$ is the same distribution as $\oplus_J(U_k)$. This is because $G_2(U_k)_I \sim U_d$ since $G_2$ is a 0-PRG for $d$-juntas, so $\oplus_I(U_n)$ is the same as $\oplus_I(G_2(U_k)) = (\oplus_I \circ G_2)(U_k) = \oplus_J(U_k)$. Since $G_1$ $\varepsilon$-fools $\oplus_J$:

$$\left|\mathbf{Pr}\big[\oplus_I(G_2(G_1(U_\ell))) = 1\big] - \mathbf{Pr}\big[\oplus_I(U_n) = 1\big]\right| \;=\; \left|\mathbf{Pr}\big[\oplus_J(G_1(U_\ell)) = 1\big] - \mathbf{Pr}\big[\oplus_J(U_k) = 1\big]\right| \;\leq\; \varepsilon$$

**Exercise 11.4.e:** By Exercise 11.4.c, there exists a poly($n$)-time computable linear 0-PRG $G_2 : \{0,1\}^k \to \{0,1\}^n$ for $d$-juntas with seed length $k = O(d \log n)$. By Theorem 11.7, there exists a $\mathrm{poly}\big(k \log(1/\varepsilon)\big)$-time computable $\varepsilon$-bias generator $G_1 : \{0,1\}^\ell \to \{0,1\}^k$ with seed length $\ell = O\big(\log(k/\varepsilon)\big) = O\big(\log(d/\varepsilon) + \log\log n\big)$. By Exercise 11.4.d, $G_2 \circ G_1 : \{0,1\}^\ell \to \{0,1\}^n$ is a $\mathrm{poly}\big(n \log(1/\varepsilon)\big)$-time computable $\varepsilon$-PRG for parities of at most $d$ variables.

**Exercise 11.4.f:** By Exercise 11.4.e, there exists an $(\varepsilon/2^{d+1})$-PRG $G\colon \{0,1\}^{\ell} \to \{0,1\}^n$ for parities of at most $d$ variables with seed length

$$\ell \;=\; O\big(\log(d/(\varepsilon/2^{d+1})) + \log\log n\big) \;=\; O\big(d + \log(1/\varepsilon) + \log\log n\big)$$

and computable in time $\mathrm{poly}\big(n\log(1/(\varepsilon/2^{d+1}))\big) = \mathrm{poly}\big(n\log(1/\varepsilon)\big)$. Suppose $f\colon \{0,1\}^n \to \{0,1\}$ is computed by a depth-$d$ decision tree. Without loss of generality, every leaf is at depth exactly $d$. By the proof of Theorem 6.21, $f$ can be expressed as a sum (over $\mathbb{R}$) of width-$d$ terms $f = \sum_{\text{leaf } v \text{ labeled } 1} T_v$ where $T_v$ accepts inputs that lead to $v$. By the proof of Theorem 11.11, each width-$d$ term is a linear combination of parities of at most $d$ variables and is $2(\varepsilon/2^{d+1}) = (\varepsilon/2^d)$-fooled by $G$.

$$\begin{aligned}
&\Big|\mathbf{Pr}\big[f(G(U_\ell)) = 1\big] - \mathbf{Pr}\big[f(U_n) = 1\big]\Big| \\
=\;& \Big|\mathbf{E}\big[\textstyle\sum_{\text{leaf } v \text{ labeled } 1} T_v(G(U_\ell))\big] - \mathbf{E}\big[\sum_{\text{leaf } v \text{ labeled } 1} T_v(U_n)\big]\Big| \\
=\;& \Big|\textstyle\sum_{\text{leaf } v \text{ labeled } 1}\Big(\mathbf{E}\big[T_v(G(U_\ell))\big] - \mathbf{E}\big[T_v(U_n)\big]\Big)\Big| && \text{(linearity of expectation)} \\
\le\;& \textstyle\sum_{\text{leaf } v \text{ labeled } 1}\Big|\mathbf{E}\big[T_v(G(U_\ell))\big] - \mathbf{E}\big[T_v(U_n)\big]\Big| && \text{(Lemma 0.28)} \\
\le\;& \textstyle\sum_{\text{leaf } v \text{ labeled } 1} \varepsilon/2^d \\
\le\;& \varepsilon
\end{aligned}$$

**Exercise 11.5.a:** Consider any valid input $x$ of size $N$. We have $\mathbf{Pr}_{q\sim U_b}\big[\Pi(x;q)\neq A(x)\big]\leq 1/3$. If we run $\Pi$ $n$ times with independent randomness for each run and output the majority vote of the runs, the error probability would be $\leq \varepsilon/2$ (Lemma 7.7).

Consider the length-$n$ width-$n$ block-size-$b$ streaming branching program $B$ that simulates the above amplification: The input is $(q_1,\ldots,q_n)\in(\{0,1\}^b)^n$. The output is the majority of $\Pi(x;q_1),\ldots,\Pi(x;q_n)$. The non-output layers are numbered $1,\ldots,n$. The nodes in layer $i$ are numbered $0,\ldots,i-1$. Reaching node $j$ in layer $i$ means exactly $j$ of $\Pi(x;q_1),\ldots,\Pi(x;q_{i-1})$ accepted. The start node is node 0 in layer 1. The outgoing edge labeled $q\in\{0,1\}^b$ from node $j$ in layer $i$ goes to node $j$ in layer $i+1$ if $\Pi(x;q)$ rejects, or to node $j+1$ in layer $i+1$ if $\Pi(x;q)$ accepts. "Layer $n+1$" would have nodes numbered $0,\ldots,n$, but those numbered $0,\ldots,\lfloor n/2\rfloor$ are combined into one reject node, and those numbered $\lceil n/2\rceil,\ldots,n$ are combined into one accept node.

$$\mathbf{Pr}\big[\Pi'(x)\neq A(x)\big] \;=\; \mathbf{Pr}\big[B(G(U_\ell))\neq A(x)\big] \;\leq\; \mathbf{Pr}\big[B(U_{bn})\neq A(x)\big]+\varepsilon/2 \;\leq\; \varepsilon/2+\varepsilon/2 \;=\; \varepsilon$$

**Exercise 11.5.b:** If $A(x) = 0$ then $\Pi(x;q)$ rejects for all $q \in \{0,1\}^b$ and therefore $\Pi'(x)$ rejects with probability 1. Now, assume $A(x) = 1$. Recall the recursive definition of $G_d$:

> $G_d(r_1, \ldots, r_d, s)$:
> if $d = 0$: output $s$
> else: call $G_{d-1}(r_1, \ldots, r_{d-1}, s)$ and then call $G_{d-1}(r_1, \ldots, r_{d-1}, h_{r_d}(s))$

We show that this maintains the invariant that $\mathbf{Pr}_{r_1,\ldots,r_d}\big[(r_1,\ldots,r_d) \text{ is bad}\big] \leq d2^{-b/3}$. This holds for the base case $G_0$ because the empty tuple () is good since:

$$\mathbf{Pr}_s\big[s \text{ is bad}\big] \;=\; \mathbf{Pr}_s\big[\Pi(x;s) \text{ rejects}\big] \;\leq\; 1/3 \;\leq\; 3^{-2^0} + 4 \cdot 2^{-b/3}$$

To see that $G_d$ maintains the invariant when $d > 0$, assume the invariant holds for $G_{d-1}$:

$$\mathbf{Pr}_{r_1,\ldots,r_{d-1}}\big[(r_1,\ldots,r_{d-1}) \text{ is bad}\big] \;\leq\; (d-1)2^{-b/3}$$

Consider any particular good $(r_1, \ldots, r_{d-1})$, and let $Y = \big\{s \in \{0,1\}^b \;:\; (r_1,\ldots,r_{d-1},s) \text{ is bad}\big\}$ so:

$$\mathbf{Pr}_{s,t}\big[s \in Y \text{ and } t \in Y\big] \;=\; \mathbf{Pr}_s\big[s \in Y\big]^2 \;\leq\; \big(3^{-2^{d-1}} + 4 \cdot 2^{-b/3}\big)^2 \;\leq\; 3^{-2^d} + 3 \cdot 2^{-b/3}$$

For any $r_d$ and $s$: $(r_1,\ldots,r_d,s)$ is bad iff $(r_1,\ldots,r_{d-1},s)$ and $(r_1,\ldots,r_{d-1},h_{r_d}(s))$ are both bad iff $s \in Y$ and $h_{r_d}(s) \in Y$. Thus for any $r_d$, if $(r_1,\ldots,r_d)$ is bad then

$$\mathbf{Pr}_s\big[s \in Y \text{ and } h_{r_d}(s) \in Y\big] \;>\; 3^{-2^d} + 4 \cdot 2^{-b/3} \;\geq\; \mathbf{Pr}_{s,t}\big[s \in Y \text{ and } t \in Y\big] + 2^{-b/3}$$

which implies that $h_{r_d}$ is $(Y, Y, 2^{-b/3})$-bad. By Lemma 11.13:

$$\mathbf{Pr}_{r_d}\big[(r_1,\ldots,r_d) \text{ is bad}\big] \;\leq\; \mathbf{Pr}_{r_d}\big[h_{r_d} \text{ is } (Y,Y,2^{-b/3})\text{-bad}\big] \;\leq\; 1/2^b(2^{-b/3})^2 \;=\; 2^{-b/3}$$

It follows that:

$$\begin{aligned}
&\mathbf{Pr}_{r_1,\ldots,r_d}\big[(r_1,\ldots,r_d) \text{ is bad}\big] \\
\leq\; &\mathbf{Pr}\big[(r_1,\ldots,r_{d-1}) \text{ is bad or } (r_1,\ldots,r_d) \text{ is bad}\big] \\
\leq\; &\mathbf{Pr}\big[(r_1,\ldots,r_{d-1}) \text{ is bad}\big] + \mathbf{Pr}\big[(r_1,\ldots,r_d) \text{ is bad} \,\big|\, (r_1,\ldots,r_{d-1}) \text{ is good}\big] \\
\leq\; &(d-1)2^{-b/3} + 2^{-b/3} \\
=\; &d2^{-b/3}
\end{aligned}$$

This proves that $G_d$ maintains the invariant. In conclusion:

$$\begin{aligned}
\mathbf{Pr}\big[\Pi'(x) \text{ rejects}\big] \;&=\; \mathbf{Pr}\big[(r_1,\ldots,r_d,s) \text{ is bad}\big] \\
&\leq\; \mathbf{Pr}\big[(r_1,\ldots,r_d) \text{ is bad or } (r_1,\ldots,r_d,s) \text{ is bad}\big] \\
&\leq\; \mathbf{Pr}\big[(r_1,\ldots,r_d) \text{ is bad}\big] + \mathbf{Pr}\big[(r_1,\ldots,r_d,s) \text{ is bad} \,\big|\, (r_1,\ldots,r_d) \text{ is good}\big] \\
&\leq\; d2^{-b/3} + \big(3^{-2^d} + 4 \cdot 2^{-b/3}\big) \\
&=\; 3^{-n} + (d+4)2^{-b/3}
\end{aligned}$$

**Exercise 11.6:** $G_a$ is an $\varepsilon$-PRG for size-$t$ circuits because for every size-$t$ circuit $C\colon \{0,1\}^n \to \{0,1\}$, defining the size-$t$ circuit $C_a\colon \{0,1\}^n \to \{0,1\}$ by $C_a(x) = C(x \oplus a)$ (which is $C$ with a $\neg$ gate added to input node $x_i$ if $a_i = 1$):

$$\begin{aligned}
&\left|\mathbf{Pr}\left[C(G_a(U_\ell)) = 1\right] - \mathbf{Pr}\left[C(U_n) = 1\right]\right| \\
= &\left|\mathbf{Pr}\left[C(G(U_\ell) \oplus a) = 1\right] - \mathbf{Pr}\left[C(U_n \oplus a) = 1\right]\right| && (U_n \oplus a = U_n) \\
= &\left|\mathbf{Pr}\left[C_a(G(U_\ell)) = 1\right] - \mathbf{Pr}\left[C_a(U_n) = 1\right]\right| \\
\leq &\ \varepsilon && (G \text{ is an } \varepsilon\text{-PRG for size-}t \text{ circuits})
\end{aligned}$$

Now, $G(U_\ell)$ and $G_a(U_\ell)$ are $2\varepsilon$-indistinguishable by size-$t$ circuits because for every size-$t$ circuit $C\colon \{0,1\}^n \to \{0,1\}$:

$$\begin{aligned}
&\left|\mathbf{Pr}\left[C(G(U_\ell)) = 1\right] - \mathbf{Pr}\left[C(G_a(U_\ell)) = 1\right]\right| \\
\leq &\left|\mathbf{Pr}\left[C(G(U_\ell)) = 1\right] - \mathbf{Pr}\left[C(U_n) = 1\right]\right| + \left|\mathbf{Pr}\left[C(U_n) = 1\right] - \mathbf{Pr}\left[C(G_a(U_\ell)) = 1\right]\right| \\
\leq &\ \varepsilon + \varepsilon
\end{aligned}$$

**Exercise 11.7:**

$$\mathbf{Pr}\left[f'(D)=1\right] \;=\; \mathbf{Pr}_{x\sim D}\left[f(x_{<i}) \neq x_i\right] \;<\; 1/2-\varepsilon$$

and

$$\mathbf{Pr}\left[f'(U_n)=1\right] \;=\; \mathbf{Pr}_{x\sim U_n}\left[f(x_{<i}) \neq x_i\right] \;=\; 1/2$$

since $f(x_{<i})$ and $x_i \sim U_1$ are independent.

**Exercise 11.8:** First, we prove an analogue of Lemma 11.27:

**Lemma.** *$G\colon \{0,1\}^\ell \to \{0,1\}^n$ defined by $G(s)_i = g(s_{H_i})$ is a $(t,\varepsilon)$-PRG if $g\colon \{0,1\}^m \to \{0,1\}$ is $(t2^k, 2\varepsilon/n)$-hard and $(H_1,\ldots,H_n)$ is a $(k,\ell,m,n)$-system.*

**Proof.** Suppose $G$ isn't a $(t,\varepsilon)$-PRG. For some $t$-node branching program $B\colon \{0,1\}^n \to \{0,1\}$:

$$\left|\mathbf{Pr}\big[B(G(U_\ell))=1\big]-\mathbf{Pr}\big[B(U_n)=1\big]\right| > \varepsilon$$

Using $B$ as $f$ in Lemma 11.26, we obtain $f^*\colon \{0,1\}^m \to \{0,1\}$ such that:

$$\mathbf{Pr}_{z\sim U_m}\big[f^*(z)=g(z)\big] > (1+2\varepsilon/n)/2$$

To conclude that $g$ is $(t2^k, 2\varepsilon/n)$-easy, we design a $t2^k$-node branching program $B^*$ that computes:

$$f^*(z) = f\Big(g_1\big(z_{H_i\cap H_1}\big),\ldots,g_{i-1}\big(z_{H_i\cap H_{i-1}}\big),y_{\geq i}\Big)\oplus b$$

We start with $B$. The size doesn't increase when we hardwire $y_{\geq i}$ (and contract) and interchange the output nodes if $b=1$. We replace each node that reads $x_j$ for some $j<i$ (there are $\leq t$ such nodes) with a decision tree with $\leq 2^{|H_i\cap H_j|} \leq 2^k$ nodes that computes $g_j$ by reading all bits of $z$ it depends on. ■

The next part is the same as the proof of Theorem 11.23. Assume $\mathsf{E}$ contains a total $A\colon \{0,1\}^+ \to \{0,1\}$ such that $A_N$ is $(2^{cN}, 2^{-cN})$-hard for some constant $c>0$ and all large enough $N$. We design a $(t,1/7)$-PRG with seed length $O(\log t)$ and computable in time $\operatorname{poly} t$, for all large enough $t \geq n$. Define:

$$d = \lceil 1/c\rceil \qquad k = \lceil \log(4t)\rceil \qquad m = 2dk \qquad \ell = 12dm$$

Let $g = A_m\colon \{0,1\}^m \to \{0,1\}$, which is $(2^{cm}, 2^{-cm})$-hard and thus $(t2^k, 2/7n)$-hard since:

$$t2^k \leq 2^k 2^k = 2^{m/d} \leq 2^{cm} \qquad 2/7n \geq 1/4t \geq 2^{-k} \geq 2^{-cm}$$

There exists a $(k,\ell,m,n)$-system $(H_1,\ldots,H_n)$ since $n \leq t \leq 2^k \leq (k\ell/em^2)^k$ (Lemma 11.28). Thus $G$ is a $(t,1/7)$-PRG (by our analogue of Lemma 11.27) with seed length $\ell = 24d^2\lceil\log(4t)\rceil = O(\log t)$ and computable in time $\operatorname{poly}(n2^\ell) = \operatorname{poly} t$, as in the proof of Theorem 11.23.

Branching Program Gap Majority is $\mathsf{BP}\cdot\mathsf{L}$-complete by a proof analogous to Lemma 7.39 except using Theorem 5.26 to obtain a log-space uniform branching program family for Output of $\Pi$ with Randomness.

To conclude that $\mathsf{BP}\cdot\mathsf{L} \subseteq \mathsf{P}$, we show Branching Program Gap Majority $\in \mathsf{P}$ like in Lemma 11.5: Say $G_{n,t}\colon \{0,1\}^\ell \to \{0,1\}^n$ is a 1/7-PRG for $n$-variable $t$-node branching programs where $t \geq n$. Given such a branching program $B$, we evaluate $B(G_{n,t}(s))$ for every $s \in \{0,1\}^\ell$ and accept iff a majority of the evaluations accepted. This takes $\operatorname{poly} t$ time since computing $G_{n,t}(s)$ and evaluating $B$ both take $\operatorname{poly} t$ time, and there are $2^\ell = 2^{O(\log t)} = \operatorname{poly} t$ possibilities of $s$. This is correct since:

$$\mathbf{Pr}\big[B(U_n)=1\big] \geq 2/3 \Rightarrow \mathbf{Pr}\big[B(G_{n,t}(U_\ell))=1\big] \geq 2/3-1/7 > 1/2$$
$$\mathbf{Pr}\big[B(U_n)=1\big] \leq 1/3 \Rightarrow \mathbf{Pr}\big[B(G_{n,t}(U_\ell))=1\big] \leq 1/3+1/7 < 1/2$$

**Exercise 11.9:** Consider any $G\colon \{0,1\}^{\ell} \to \{0,1\}^n$ with $\ell \leq c \log n$. We design a size-$n$ circuit $C\colon \{0,1\}^n \to \{0,1\}$ (with $\leq n$ many $\wedge$ and $\vee$ gates) such that:

$$\left|\mathbf{Pr}\big[C(G(U_\ell)) = 1\big] - \mathbf{Pr}\big[C(U_n) = 1\big]\right| \;>\; 1/2$$

For each $s \in \{0,1\}^{\ell}$, let $I_s \subseteq [n]$ be a set of $\lfloor n/2^{\ell} \rfloor$ many variable indices, such that the sets $I_s$ are pairwise disjoint over all $s$. Let $C$ be a DNF where each $s \in \{0,1\}^{\ell}$ has an associated term on variables $I_s$ that accepts $G(s)$. As a bounded-fan-in circuit, each term of $C$ contributes $\lfloor n/2^{\ell} \rfloor - 1$ many $\wedge$ gates and one $\vee$ gate, so $C$ has $\leq 2^{\ell} \cdot (\lfloor n/2^{\ell} \rfloor - 1 + 1) \leq n$ many $\wedge$ and $\vee$ gates. We have $\mathbf{Pr}\big[C(G(U_\ell)) = 1\big] = 1$ and:

$$\begin{aligned}
\mathbf{Pr}\big[C(U_n) = 1\big] &\leq \textstyle\sum_s \mathbf{Pr}\big[s\text{'s term accepts } U_n\big] && \text{(union bound)}\\
&= \textstyle\sum_s 1/2^{\lfloor n/2^{\ell} \rfloor}\\
&\leq 2^{\ell}/2^{n/2^{\ell}-1}\\
&\leq n^c/2^{n/n^c-1}\\
&< 1/2 && \text{(for large enough } n\text{)}
\end{aligned}$$

**Exercise 11.10:** Assume $\mathsf{P} = \mathsf{NP}$. By Exercise 5.15, $\mathsf{E}$ contains a total $A\colon \{0,1\}^+ \to \{0,1\}$ with the maximum possible circuit size complexity for all $N$. By Theorem 5.8, this circuit size complexity is $\Omega(2^N/N) \geq 2^{\Omega(N)}$. By Theorem 10.19, $\mathsf{E}$ contains a total $B\colon \{0,1\}^+ \to \{0,1\}$ such that $B_N$ is $(2^{cN}, 2^{-cN})$-hard for some constant $c > 0$ and all large enough $N$. The proof of Theorem 11.23 turns $B$ into a $(t, 1/7)$-PRG with seed length $O(\log t)$ and computable in time $\operatorname{poly} t$, for all large enough $t \geq n$.

**Exercise 11.11.a:** To turn a sorting network of size $O(m \log^2 m)$ (Theorem 2.34) into a monotone circuit of size $O(m \log^2 m)$ for $T_k^m$, turn each comparator into an $\vee$ gate and an $\wedge$ gate, and output the $(m-k+1)^{\text{st}}$ bit in sorted order.

To obtain $C_k$ from $C$, push negations to the input variables (which at most doubles the size) and replace each $\neg x_i$ with $T_k^{n-1}(x_{-i})$ where $x_{-i} = x_1 \cdots x_{i-1} x_{i+1} \cdots x_n$. This $C_k$ is a monotone circuit of size $O(t + n^2 \log^2 n)$ and agrees with $C$ on every weight-$k$ input $x$ because $x_i = 0$ iff $\text{weight}(x_{-i}) \geq k$, and thus $\neg x_i = T_k^{n-1}(x_{-i})$.

**Exercise 11.11.b:** Assume $G\colon \{0,1\}^{\ell} \to \{0,1\}^n$ is an $\varepsilon/2(n+1)$-PRG for size-$c(t+n^2\log^2 n)$ monotone circuits. Consider any size-$t$ (nonmonotone) circuit $C\colon \{0,1\}^n \to \{0,1\}$. For each $k \in \{0,\ldots,n\}$, define the function $F_k\colon \{0,1\}^n \to \{0,1\}$ by

$$F_k(x) \;=\; \begin{cases} C(x) & \text{if weight}(x) = k \\ 0 & \text{if weight}(x) \neq k \end{cases}$$

and note that $C'_k$ is the size-$c(t+n^2\log^2 n)$ monotone circuit $T^n_{k+1}(x) \vee \big(T^n_k(x) \wedge C_k(x)\big)$ where $C_k$ is from Exercise 11.11.a. Also note that $C(x) = \sum_{k=0}^n F_k(x) = \sum_{k=0}^n \big(C'_k(x) - T^n_{k+1}(x)\big)$.

$$\begin{aligned}
&\Big|\mathbf{Pr}\big[C(G(U_\ell)) = 1\big] - \mathbf{Pr}\big[C(U_n) = 1\big]\Big| \\
=\;& \Big|\textstyle\sum_{k=0}^n \Big(\mathbf{E}\big[C'_k(G(U_\ell))\big] - \mathbf{E}\big[T^n_{k+1}(G(U_\ell))\big]\Big) - \sum_{k=0}^n \Big(\mathbf{E}\big[C'_k(U_n)\big] - \mathbf{E}\big[T^n_{k+1}(U_n)\big]\Big)\Big| \\
=\;& \Big|\textstyle\sum_{k=0}^n \Big(\mathbf{E}\big[C'_k(G(U_\ell))\big] - \mathbf{E}\big[C'_k(U_n)\big]\Big) + \sum_{k=0}^n \Big(\mathbf{E}\big[T^n_{k+1}(U_n)\big] - \mathbf{E}\big[T^n_{k+1}(G(U_\ell))\big]\Big)\Big| \\
\leq\;& \textstyle\sum_{k=0}^n \Big|\mathbf{E}\big[C'_k(G(U_\ell))\big] - \mathbf{E}\big[C'_k(U_n)\big]\Big| + \sum_{k=0}^n \Big|\mathbf{E}\big[T^n_{k+1}(U_n)\big] - \mathbf{E}\big[T^n_{k+1}(G(U_\ell))\big]\Big| \\
\leq\;& \textstyle\sum_{k=0}^n \varepsilon/2(n+1) + \sum_{k=0}^n \varepsilon/2(n+1) \\
=\;& \varepsilon
\end{aligned}$$

By the way, since $\mathbf{E}\big[T^n_{n+1}(U_n)\big] - \mathbf{E}\big[T^n_{n+1}(G(U_\ell))\big] = 0 - 0 = 0$, we could disregard it and handle $\varepsilon/(2n+1)$ instead of $\varepsilon/2(n+1)$.

**Exercise 11.12:** Identify $[\ell]$ with $\mathbb{F} \times \mathbb{F}$. For each $x \in \mathbb{F}$, the set $\{(x, y) : y \in \mathbb{F}\}$ is an "interval." A structured subset of $\mathbb{F} \times \mathbb{F}$ contains exactly one element from each interval. There are $n = m^{k+1}$ many polynomials of degree $\leq k$ over $\mathbb{F}$. For the $i^{\text{th}}$ such polynomial $P_i$, define the structured set $H_i = \{(x, P_i(x)) : x \in \mathbb{F}\}$. Then $(H_1, \ldots, H_n)$ is a $(k, \ell, m, n)$-system (Corollary 0.21).

This is quantitatively better than Lemma 11.28 because the latter only yields $(k\ell/\mathrm{e}m^2)^k = (k/\mathrm{e})^k < m^{k+1}$ subsets. However, this exercise requires $\ell = m^2$, which is not good enough for the applications to PRGs and extractors, which require $\ell = O(m)$.

**Exercise 11.13.a:** The parity function works because xoring together the uniformly random bits yields a uniformly random bit, and xoring with the constant bits either negates the output or doesn't, so the output is still a uniformly random bit.

**Exercise 11.13.b:** Output the parity of the first $d/2$ bits, and the parity of the last $d/2$ bits. Since the source has min-entropy $> d/2$:

- At least one of the first $d/2$ bits is uniform, so the first output bit is uniform.
- At least one of the last $d/2$ bits is uniform, so the second output bit is uniform.

The two output bits are independent.

**Exercise 11.13.c:** Output the parity of the first $2d/3$ bits, and the parity of the last $2d/3$ bits. Since the source has min-entropy $> d/3$:

- At least one of the first $2d/3$ bits is uniform, so the first output bit is uniform.
- At least one of the last $2d/3$ bits is uniform, so the second output bit is uniform.
- At least one of the first $d/3$ or last $d/3$ bits is uniform, so the xor of the two output bits (which is the parity of the first $d/3$ and last $d/3$ source bits, since the middle $d/3$ bits cancel out) is uniform.

By Lemma 8.5, the output distribution is uniform over $\{0,1\}^2$.

**Exercise 11.14:** We prove the contrapositive. Suppose for some large enough $d$, $E_d$ is computed by a size-$(d^c - 4d)$ circuit $C\colon \{0,1\}^d \to \{0,1\}$. Let $b \in \{0,1\}$ be such that $\mathbf{Pr}\big[C(U_d) = b\big] \geq 1/2$. Define a size-$d^c$ circuit sampler $C'\colon \{0,1\}^{2d} \to \{0,1\}^d$ that takes uniformly random $(x,y) \in (\{0,1\}^d)^2$, evaluates $C(x)$, outputs $x$ if $C(x) = b$, and outputs $y$ if $C(x) = \overline{b}$. For each $z \in \{0,1\}^d$, if $C(z) = 1$ then

$$\mathbf{Pr}_{x,y}\big[C'(x,y) = z\big] \;=\; \mathbf{Pr}[x = z] + \mathbf{Pr}\big[C(x) = \overline{b} \text{ and } y = z\big] \;\leq\; 2^{-d} + (1/2) \cdot 2^{-d} \;\leq\; 2^{-(d-1)}$$

and if $C(z) = 0$ then

$$\mathbf{Pr}_{x,y}\big[C'(x,y) = z\big] \;=\; \mathbf{Pr}\big[C(x) = \overline{b} \text{ and } y = z\big] \;\leq\; (1/2) \cdot 2^{-d} \;=\; 2^{-(d+1)}$$

and thus $C'(U_{2d})$ has min-entropy $\geq d - 1$. But

$$\mathbf{Pr}_{x,y}\big[E_d(C'(x,y)) = \overline{b}\big] \;=\; \mathbf{Pr}\big[C(x) = \overline{b}\big] \cdot \mathbf{Pr}\big[C(y) = \overline{b}\big] \;\leq\; (1/2) \cdot (1/2) \;=\; 1/4$$

so $\Delta\big(E_d(C'(U_{2d})), U_1\big) \geq 1/4 > 1/5$. Thus $E_d$ is not a $(d-1, 1/5)$-extractor for size-$d^c$ circuit samplers.

**Exercise 11.15.a:** First, adapting Lemma 11.35, we show that $\Delta(D,U) \leq \mathbf{E}_{i\sim C}\big[\Delta(D_i,U)\big]$: For every event $A \subseteq S$:

$$\begin{aligned}
\big|D(A)-U(A)\big| &= \Big|\big(\textstyle\sum_i C(i)\cdot D_i(A)\big)-U(A)\Big| && \text{(law of total probability)}\\
&= \Big|\textstyle\sum_i C(i)\cdot\big(D_i(A)-U(A)\big)\Big| \\
&\leq \textstyle\sum_i C(i)\cdot\big|D_i(A)-U(A)\big| && \text{(Lemma 0.28)}\\
&\leq \textstyle\sum_i C(i)\cdot\Delta(D_i,U) \\
&= \mathbf{E}_{i\sim C}\big[\Delta(D_i,U)\big]
\end{aligned}$$

Say $i$ is *bad* iff $\Delta(D_i,U) > \varepsilon$.

$$\begin{aligned}
\Delta(D,U) &\leq \mathbf{E}_{i\sim C}\big[\Delta(D_i,U)\big] \\
&= \mathbf{Pr}_{i\sim C}[i \text{ is bad}] \cdot \mathbf{E}_{i\sim C}\big[\Delta(D_i,U)\,\big|\,i \text{ is bad}\big] + \\
&\quad\ \mathbf{Pr}_{i\sim C}[i \text{ is good}]\cdot \mathbf{E}_{i\sim C}\big[\Delta(D_i,U)\,\big|\,i \text{ is good}\big] && \text{(law of total expectation)}\\
&\leq \delta\cdot 1 + 1\cdot\varepsilon
\end{aligned}$$

**Exercise 11.15.b:** Define $H = h + \log(m/\delta)$. For every $i \in [m]$ and $r \in S$,

$$C(i) \cdot D_i(r) \;\leq\; \sum_{j=1}^{m} C(j) \cdot D_j(r) \;=\; D(r) \;\leq\; 2^{-H}$$

and thus $D_i(r) \leq 2^{-H}/C(i) = 2^{-(H-\log(1/C(i)))}$. It follows that $D_i$ has min-entropy $\geq H - \log(1/C(i))$.

$$\begin{aligned}
\mathbf{Pr}_{i\sim C}\big[D_i \text{ has min-entropy} < h\big] &\leq \mathbf{Pr}_{i\sim C}\big[D_i \text{ has min-entropy} \leq H - \log(m/\delta)\big] \\
&\leq \mathbf{Pr}_{i\sim C}\big[H - \log(1/C(i)) \leq H - \log(m/\delta)\big] \\
&= \mathbf{Pr}_{i\sim C}\big[C(i) \leq \delta/m\big] \\
&= \textstyle\sum_{i\in[m]\,:\,C(i)\leq\delta/m} C(i) \\
&\leq \textstyle\sum_{i\in[m]} \delta/m \\
&= \delta
\end{aligned}$$

**Exercise 11.15.c:** Consider a mixture $D = \sum_{i=1}^{m} C(i) \cdot D_i$ where $D$ has min-entropy $\geq h + \log(m/\delta)$ and each $D_i \in T$. With probability $\geq 1 - \delta$ over $i \sim C$, $D_i$ has min-entropy $\geq h$ (Exercise 11.15.b) and therefore $\Delta\big(E(D_i), U_n\big) \leq \varepsilon$ by assumption. Since $E(D)$ is the mixture $\sum_{i=1}^{m} C(i) \cdot E(D_i)$ (like in the proof of Lemma 11.37), $\Delta\big(E(D), U_n\big) \leq \delta + \varepsilon$ (Exercise 11.15.a).

**Exercise 11.15.d:** By assumption, $E\colon \{0,1\}^d \to \{0,1\}^n$ is an $(h,\varepsilon)$-extractor for the set $T$ of all distributions over $\{0,1\}^d$ where the first $d/2$ bits are independent of the last $d/2$ bits. By Exercise 11.15.c, it suffices to show that the distribution $D$ over $\{0,1\}^d$ sampled by any $m$-node sampler is a mixture of at most $m$ distributions from $T$. For each node $i$ in the sampler's middle layer, let $D_i$ be $D$ conditioned on the random walk visiting node $i$. Then $D_i \in T$ because a random walk's distribution after reaching node $i$ doesn't depend on how it got to $i$. We have $D = \sum_i C(i) \cdot D_i$ where $C(i) = \mathbf{Pr}\big[\text{the random walk visits } i\big]$.

**Exercise 11.15.e:** $E\colon \{0,1\}^{d/2} \times \{0,1\}^{d/2} \to \{0,1\}$ defined by $E(r_1, r_2) = r_1 \odot r_2$ (the dot product over $\mathbb{Z}_2$) is a two-source $\big(d/2 + 2\log(1/2\varepsilon),\ \varepsilon\big)$-extractor (Theorem 11.31) and therefore a $\big(d/2 + 2\log(1/2\varepsilon) + \log(m/\delta),\ \delta + \varepsilon\big)$-extractor for $m$-node samplers (Exercise 11.15.d). Choosing $\delta = \varepsilon = \gamma/2$, $E$ is a $\big(d/2 + \log(2m/\gamma^3),\ \gamma\big)$-extractor for $m$-node samplers.

**Exercise 11.16:** Say $D = C(1) \cdot D_1 + C(2) \cdot D_2$. For each $r \in S$, if $D_1(r) \geq D_2(r)$ then

$$D_1(r) \;=\; C(1)D_1(r) + C(2)D_1(r) \;\geq\; D(r) \;\geq\; C(1)D_2(r) + C(2)D_2(r) \;=\; D_2(r)$$

and thus

$$\begin{aligned} \big|D_1(r) - D_2(r)\big| &= D_1(r) - D_2(r) \\ &= \big(D_1(r) - D(r)\big) + \big(D(r) - D_2(r)\big) \\ &= \big|D_1(r) - D(r)\big| + \big|D(r) - D_2(r)\big| \end{aligned}$$

and similarly, if $D_1(r) < D_2(r)$ then $\big|D_1(r) - D_2(r)\big| = \big|D_1(r) - D(r)\big| + \big|D(r) - D_2(r)\big|$. By Lemma 11.1:

$$\begin{aligned} 2\Delta(D_1, D_2) &= \textstyle\sum_{r \in S} \big|D_1(r) - D_2(r)\big| \\ &= \textstyle\sum_{r \in S} \big(\big|D_1(r) - D(r)\big| + \big|D(r) - D_2(r)\big|\big) \\ &= \textstyle\sum_{r \in S} \big|D_1(r) - D(r)\big| + \sum_{r \in S} \big|D(r) - D_2(r)\big| \\ &= 2\Delta(D_1, D) + 2\Delta(D, D_2) \end{aligned}$$

**Exercise 11.17.a:** First proof: Let $E_1, E_2 \subseteq S$ be events such that $D_1(E_1) - U(E_1) = \Delta(D_1, U) \geq 1-\delta$ and $D_2(E_2) - U(E_2) = \Delta(D_2, U) \geq 1-\varepsilon$. Say $D_1(E_1) = 1-\alpha$, so $U(E_1) \leq (1-\alpha)-(1-\delta) = \delta - \alpha$. Say $D_2(E_2) = 1-\beta$, so $U(E_2) \leq (1-\beta)-(1-\varepsilon) = \varepsilon - \beta$. Let $E = E_1 \cup E_2$, so $U(E) \leq U(E_1)+U(E_2) \leq (\delta-\alpha)+(\varepsilon-\beta) = \delta+\varepsilon-\alpha-\beta$. For any mixture $D = C(1)\cdot D_1 + C(2)\cdot D_2$,

$$\begin{aligned} D(E) &= C(1)D_1(E) + C(2)D_2(E) \\ &\geq C(1)D_1(E_1) + C(2)D_2(E_2) \\ &= C(1)(1-\alpha) + C(2)(1-\beta) \\ &= 1 - C(1)\alpha - C(2)\beta \\ &\geq 1-\alpha-\beta \end{aligned}$$

Thus $D(E) - U(E) \geq (1-\alpha-\beta)-(\delta+\varepsilon-\alpha-\beta) = 1-\delta-\varepsilon$, so $\Delta(D, U) \geq 1-\delta-\varepsilon$.

Second proof: Define:

$$\begin{aligned} a &= \textstyle\sum_{r\,:\,D_1(r) \leq \min(D_2(r), U(r))} \big(\min(D_2(r), U(r)) - D_1(r)\big) \\ b &= \textstyle\sum_{r\,:\,U(r) \geq \max(D_1(r), D_2(r))} \big(U(r) - \max(D_1(r), D_2(r))\big) \\ c &= \textstyle\sum_{r\,:\,D_2(r) \leq \min(D_1(r), U(r))} \big(\min(D_1(r), U(r)) - D_2(r)\big) \end{aligned}$$

Then

$$\begin{aligned} a + b &= \Big(\textstyle\sum_{r\,:\,D_1(r) \leq U(r) < D_2(r)} \big(U(r) - D_1(r)\big) + \sum_{r\,:\,D_1(r) \leq D_2(r) \leq U(r)} \big(D_2(r) - D_1(r)\big)\Big) + \\ &\quad \Big(\textstyle\sum_{r\,:\,D_1(r) \leq D_2(r) \leq U(r)} \big(U(r) - D_2(r)\big) + \sum_{r\,:\,D_2(r) < D_1(r) \leq U(r)} \big(U(r) - D_1(r)\big)\Big) \\ &= \textstyle\sum_{r\,:\,D_1(r) \leq U(r)} \big(U(r) - D_1(r)\big) \\ &= \Delta(D_1, U) \\ &\geq 1-\delta \end{aligned}$$

and similarly $b + c = \Delta(D_2, U) \geq 1-\varepsilon$ and:

$$a + b + c = \textstyle\sum_{r\,:\,U(r) \geq \min(D_1(r), D_2(r))} \big(U(r) - \min(D_1(r), D_2(r))\big) \leq \sum_r U(r) = 1$$

Next, note that:

$$\begin{aligned} a &= (a+b+c) - (b+c) \leq 1 - (1-\varepsilon) = \varepsilon \\ b &= (a+b) - a \geq (1-\delta) - \varepsilon = 1 - \delta - \varepsilon \end{aligned}$$

For any mixture $D = C(1)\cdot D_1 + C(2)\cdot D_2$,

$$\Delta(D, U) = \textstyle\sum_{r\,:\,U(r) \geq C(1)D_1(r) + C(2)D_2(r)} \big(U(r) - C(1)D_1(r) - C(2)D_2(r)\big) \geq b \geq 1-\delta-\varepsilon$$

since $C(1)D_1(r) + C(2)D_2(r) \leq \max(D_1(r), D_2(r))$.

These proofs didn't use the assumption that $U$ is uniform. The result holds with any distribution in place of $U$.

**Exercise 11.17.b:** Say $D = \sum_{i=1}^{n} C(i) \cdot D_i$ where each $C(i) > 0$. For each $k \in [n]$, define the distribution:

$$B_k = \sum_{i=1}^{k} \frac{C(i)}{\sum_{j=1}^{k} C(j)} \cdot D_i$$

We can also define these distributions using an iterative algorithm:

let $B_1 = D_1$
for $k \leftarrow 2, \ldots, n$:
    let $B_k = \frac{\sum_{j=1}^{k-1} C(j)}{\sum_{j=1}^{k} C(j)} \cdot B_{k-1} + \frac{C(k)}{\sum_{j=1}^{k} C(j)} \cdot D_k$

This maintains the invariant that $\Delta(B_k, U) \geq 1 - k\varepsilon$. This holds for $k = 1$ by assumption. Assuming $\Delta(B_{k-1}, U) \geq 1-(k-1)\varepsilon$ and $\Delta(D_k, U) \geq 1-\varepsilon$, we have $\Delta(B_k, U) \geq 1-(k-1)\varepsilon-\varepsilon = 1 - k\varepsilon$ since $B_k$ is a mixture of $B_{k-1}$ and $D_k$ (Exercise 11.17.a). At the end, $B_n = D$ and thus $\Delta(D, U) \geq 1 - n\varepsilon$.

This proof didn't use the assumption that $U$ is uniform. The result holds with any distribution in place of $U$.

**Exercise 11.18:** We partition $[0,1)$ into $[0,3/12)$ for a, and $[3/12,5/12)$ for b, and $[5/12, 9/12)$ for c, and $[9/12,1)$ for d.

Starting at $y = 0$, we have $B_y = \{0, 4/12, 8/12\}$ and $A_y = \{\mathrm{a},\mathrm{b},\mathrm{c}\}$. As we increase $y$, $A_y$ remains $\{\mathrm{a},\mathrm{b},\mathrm{c}\}$ until $y = 1/12$, at which point $y+4/12$ leaves b's interval and enters c's interval, and $y+8/12$ leaves c's interval and enters d's interval. Then $A_y$ remains $\{\mathrm{a},\mathrm{c},\mathrm{d}\}$ until $y = 3/12$, at which point $y$ leaves a's interval and enters b's interval. Then $A_y$ remains $\{\mathrm{b},\mathrm{c},\mathrm{d}\}$ for the rest of $y < 4/12$.

| $r$ | a | b | c | d |
|---|---|---|---|---|
| $U_{\{\mathrm{a},\mathrm{b},\mathrm{c}\}}(r)$ | 1/3 | 1/3 | 1/3 | 0 |
| $U_{\{\mathrm{a},\mathrm{b},\mathrm{d}\}}(r)$ | 1/3 | 1/3 | 0 | 1/3 |
| $U_{\{\mathrm{a},\mathrm{c},\mathrm{d}\}}(r)$ | 1/3 | 0 | 1/3 | 1/3 |
| $U_{\{\mathrm{b},\mathrm{c},\mathrm{d}\}}(r)$ | 0 | 1/3 | 1/3 | 1/3 |

In summary:

$$\begin{aligned} D &= \tfrac{1/12-0}{1/3} \cdot U_{\{\mathrm{a},\mathrm{b},\mathrm{c}\}} + \tfrac{3/12-1/12}{1/3} \cdot U_{\{\mathrm{a},\mathrm{c},\mathrm{d}\}} + \tfrac{4/12-3/12}{1/3} \cdot U_{\{\mathrm{b},\mathrm{c},\mathrm{d}\}} \\ &= (1/4) \cdot U_{\{\mathrm{a},\mathrm{b},\mathrm{c}\}} + (1/2) \cdot U_{\{\mathrm{a},\mathrm{c},\mathrm{d}\}} + (1/4) \cdot U_{\{\mathrm{b},\mathrm{c},\mathrm{d}\}} \end{aligned}$$

**Exercise 11.19.a:** This is like the proof of Lemma 11.37.

$\Rightarrow$: This is because $k$-flat distributions have min-entropy $\log k$.

$\Leftarrow$: Consider any $D_1$ and $D_2$ that both have min-entropy $\geq \log k$. By Lemma 11.36,

$$\begin{aligned} D_1 &= \sum_{A_1 \subseteq \{0,1\}^d : |A_1| = k} C_1(A_1) \cdot U_{A_1} \\ D_2 &= \sum_{A_2 \subseteq \{0,1\}^d : |A_2| = k} C_2(A_2) \cdot U_{A_2} \end{aligned}$$

are mixtures of $k$-flat distributions. Then $E(D_1, D_2)$ is the mixture $\sum_{A_1, A_2} \big(C_1(A_1) \cdot C_2(A_2)\big) \cdot E(U_{A_1}, U_{A_2})$ like in the proof of Lemma 11.37. Since $\Delta\big(E(U_{A_1}, U_{A_2}), U_n\big) \leq \varepsilon$ for each pair $(A_1, A_2)$ by assumption, $\Delta\big(E(D_1, D_2), U_n\big) \leq \varepsilon$ (Lemma 11.35).

**Exercise 11.19.b:** This is like the proof of Theorem 11.33.

Sample $E\colon \{0,1\}^d \times \{0,1\}^d \to \{0,1\}^n$ uniformly at random: For each $(r_1, r_2) \in \{0,1\}^d \times \{0,1\}^d$ independently, sample $E(r_1, r_2) \sim U_n$. Consider any $A_1, A_2 \subseteq \{0,1\}^d$ with $|A_1| = |A_2| = 2^h$ and any $f\colon \{0,1\}^n \to \{0,1\}$. For each $(r_1, r_2) \in A_1 \times A_2$, imagine a coin toss where heads means $f(E(r_1, r_2)) = 1$. These $2^{2h}$ coin tosses are fully independent, and each has heads probability $p = \mathbf{Pr}\big[f(U_n) = 1\big]$. For any outcome $E$, the fraction of tosses that are heads is $\mathbf{Pr}\big[f(E(U_{A_1}, U_{A_2})) = 1\big]$, and by Lemma 7.20:

$$\mathbf{Pr}_E\Big[\text{fraction of heads isn't within } \pm\varepsilon \text{ of } p\Big] \;\leq\; 2e^{-\varepsilon^2 2^{2h}} \;<\; 2^{1-\varepsilon^2 2^{2h}}$$

Thus:

$$\begin{aligned}
&\mathbf{Pr}_E\big[E \text{ isn't a two-source } (h,h,\varepsilon)\text{-extractor}\big] \\
&= \mathbf{Pr}_E\big[\Delta\big(E(U_{A_1}, U_{A_2}), U_n\big) > \varepsilon \text{ for some } 2^h\text{-flat } U_{A_1}, U_{A_2}\big] && \text{(Exercise 11.19.a)} \\
&= \mathbf{Pr}_E\Big[\big|\mathbf{Pr}\big[f(E(U_{A_1}, U_{A_2})) = 1\big] - p\big| > \varepsilon \text{ for some } 2^h\text{-flat } U_{A_1}, U_{A_2} \text{ and some } f\Big] \\
&\leq \textstyle\sum_{A_1, A_2, f} \mathbf{Pr}_E\Big[\big|\mathbf{Pr}\big[f(E(U_{A_1}, U_{A_2})) = 1\big] - p\big| > \varepsilon\Big] && \text{(union bound)} \\
&< \textstyle\sum_{A_1, A_2, f} 2^{1-\varepsilon^2 2^{2h}} \\
&= \binom{2^d}{2^h}^2 \cdot 2^{2^n} \cdot 2^{1-\varepsilon^2 2^{2h}} \\
&\leq 2^{2d2^h + 2^n + 1 - \varepsilon^2 2^{2h}} && \left(\tbinom{2^d}{2^h} \leq (2^d)^{2^h}\right) \\
&< 1
\end{aligned}$$

**Exercise 11.20.a:**

$$(D-U)\cdot(D-U) \;=\; (D\cdot D)-2(D\cdot U)+(U\cdot U) \;=\; p-2/|S|+1/|S| \;=\; p-1/|S|$$

$$C\cdot C \;=\; \sum_{r\in S} C(r)^2 \;=\; \sum_{r\in S} 1 \;=\; |S|$$

$$\begin{aligned}
2\Delta(D,U) &= \textstyle\sum_{r\in S}\big|(D-U)(r)\big| && \text{(Lemma 11.1)}\\
&= (D-U)\cdot C\\
&\le \sqrt{\big((D-U)\cdot(D-U)\big)(C\cdot C)} && \text{(Theorem 0.26)}\\
&= \sqrt{(p-1/|S|)|S|}\\
&= \sqrt{|S|p-1}
\end{aligned}$$

**Exercise 11.20.b:** Consider any distribution $D$ over $\{0,1\}^d$ with min-entropy $\geq h$. Note that $D$ has collision probability $\sum_{r \in S} D(r)^2 \leq \sum_{r \in S} 2^{-h} D(r) = 2^{-h}$. Now, we bound the collision probability of $(E(D, U_\ell), U_\ell)$. If we independently sample $(r,s) \sim (D, U_\ell)$ and $(r',s') \sim (D, U_\ell)$:

$$\begin{aligned}
& \mathbf{Pr}\Big[(E(r,s),s) = (E(r',s'),s')\Big] \\
= \; & \mathbf{Pr}\Big[s = s'\Big] \cdot \mathbf{Pr}\Big[E(r,s) = E(r',s') \,\Big|\, s = s'\Big] && \text{(chain rule)} \\
= \; & 2^{-\ell} \cdot \mathbf{Pr}\Big[E(r,s) = E(r',s)\Big] \\
= \; & 2^{-\ell} \cdot \Big(\mathbf{Pr}\Big[r = r'\Big] \cdot \mathbf{Pr}\Big[E(r,s) = E(r',s) \,\Big|\, r = r'\Big] + \\
& \qquad \mathbf{Pr}\Big[r \neq r'\Big] \cdot \mathbf{Pr}\Big[E(r,s) = E(r',s) \,\Big|\, r \neq r'\Big]\Big) && \text{(law of total probability)} \\
\leq \; & 2^{-\ell} \cdot \left(2^{-h} \cdot 1 + 1 \cdot 2^{-n}\right) && (H \text{ is pairwise avoiding}) \\
= \; & 2^{-\ell-h} + 2^{-\ell-n}
\end{aligned}$$

By Exercise 11.20.a:

$$\Delta\Big((E(D,U_\ell),U_\ell),\,(U_n,U_\ell)\Big) \;\leq\; \tfrac{1}{2}\sqrt{2^{n+\ell}(2^{-\ell-h}+2^{-\ell-n})-1} \;=\; 2^{(n-h)/2-1} \;\leq\; \varepsilon$$

**Exercise 11.21.a:** We prove the contrapositive. Say $s$ is *bad* iff $\Delta\big(E(D,s),\, U_n\big) > \sqrt{\varepsilon}$. Assume $\mathbf{Pr}_{s\sim U_\ell}\big[s \text{ is bad}\big] > \sqrt{\varepsilon}$. For each bad $s$, let $f_s\colon \{0,1\}^n \to \{0,1\}$ be such that $\mathbf{Pr}\big[f_s(E(D,s)) = 1\big] > \mathbf{Pr}\big[f_s(U_n) = 1\big] + \sqrt{\varepsilon}$. For each good $s$, let $f_s$ be the constant-0 function. Define $f\colon \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}$ by $f(q,s) = f_s(q)$.

$$\begin{aligned}
\mathbf{Pr}\big[f(E(D,U_\ell),U_\ell) = 1\big] &= \mathbf{E}_s\Big[\mathbf{Pr}\big[f(E(D,s),s) = 1\big]\Big] \\
&= \textstyle\sum_s 2^{-\ell} \cdot \mathbf{Pr}\big[f_s(E(D,s)) = 1\big] \\
&> \textstyle\sum_{\text{bad } s} 2^{-\ell} \cdot \big(\mathbf{Pr}\big[f_s(U_n) = 1\big] + \sqrt{\varepsilon}\big) \\
&= \textstyle\sum_s 2^{-\ell} \cdot \mathbf{Pr}\big[f_s(U_n) = 1\big] + \mathbf{Pr}_s\big[s \text{ is bad}\big] \cdot \sqrt{\varepsilon} \\
&> \mathbf{Pr}\big[f(U_n,U_\ell) = 1\big] + \sqrt{\varepsilon} \cdot \sqrt{\varepsilon}
\end{aligned}$$

Thus $\Delta\big((E(D,U_\ell),U_\ell),\,(U_n,U_\ell)\big) > \varepsilon$.

**Exercise 11.21.b:** Say $s$ is *bad* iff $\Delta\big(E(D,s),\, U_n\big) > \gamma$. Assume $\mathbf{Pr}_{s\sim U_\ell}\big[s \text{ is bad}\big] \leq \delta$. Consider any $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}$. For each $s$, define $f_s : \{0,1\}^n \to \{0,1\}$ by $f_s(q) = f(q,s)$.

$$
\begin{aligned}
& \mathbf{Pr}\big[f(E(D,U_\ell),U_\ell) = 1\big] \\
= \; & \mathbf{E}_s\Big[\mathbf{Pr}\big[f(E(D,s),s) = 1\big]\Big] \\
= \; & \mathbf{E}_s\Big[\mathbf{Pr}\big[f_s(E(D,s)) = 1\big]\Big] \\
\leq \; & \mathbf{E}_s\Big[\mathbf{Pr}\big[f_s(U_n) = 1\big] + \Delta\big(E(D,s),\, U_n\big)\Big] \\
= \; & \mathbf{Pr}\big[f(U_n,U_\ell) = 1\big] + \mathbf{E}_s\Big[\Delta\big(E(D,s),\, U_n\big)\Big] && \text{(linearity of expectation)} \\
= \; & \mathbf{Pr}\big[f(U_n,U_\ell) = 1\big] + \\
& \mathbf{Pr}_s\big[s \text{ is good}\big] \cdot \mathbf{E}_s\Big[\Delta\big(E(D,s),\, U_n\big) \,\Big|\, s \text{ is good}\Big] + \\
& \mathbf{Pr}_s\big[s \text{ is bad}\big] \;\cdot \mathbf{E}_s\Big[\Delta\big(E(D,s),\, U_n\big) \,\Big|\, s \text{ is bad}\Big] && \text{(law of total expectation)} \\
\leq \; & \mathbf{Pr}\big[f(U_n,U_\ell) = 1\big] + 1 \cdot \gamma + \delta \cdot 1
\end{aligned}
$$

Thus $\Delta\big((E(D,U_\ell),U_\ell),\,(U_n,U_\ell)\big) \leq \gamma + \delta$.

**Exercise 12.1:** Consider any private-key encryption scheme $(E, D)$ where the key length $\ell$ is less than the message length $n$, and with ciphertext length $c$. The proof of Theorem 12.5 showed that there exist messages $m_0 \neq m_1$ and a function $A_n \colon \{0,1\}^c \to \{0,1\}$ such that:

$$\mathbf{Pr}\left[A_n(E_{U_\ell}(m_0)) = 1\right] - \mathbf{Pr}\left[A_n(E_{U_\ell}(m_1)) = 1\right] \geq 1 - 1/2 > \varepsilon$$

This means $\Delta\left(E_{U_\ell}(m_0), E_{U_\ell}(m_1)\right) > \varepsilon$, so $(E, D)$ isn't $\varepsilon$-secure.

**Exercise 12.2:** Consider any size-$t$ circuit $C : \{0,1\}^n \to \{0,1\}$. Think of the distribution $G_2(U_m)$ as a hybrid that interpolates between $G_2(G_1(U_\ell))$ and $U_n$. We have

$$\left|\mathbf{Pr}\left[C(G_2(U_m)) = 1\right] - \mathbf{Pr}\left[C(U_n) = 1\right]\right| \leq \varepsilon/2$$

since $G_2$ is a $(t, \varepsilon/2)$-PRG. Define the size-$(t+t')$ circuit $C' : \{0,1\}^m \to \{0,1\}$ by $C'(x) = C(B(x))$ where $B : \{0,1\}^m \to \{0,1\}^n$ is a size-$t'$ circuit for $G_2$. We have

$$\begin{aligned}\left|\mathbf{Pr}\left[C(G_2(G_1(U_\ell))) = 1\right] - \mathbf{Pr}\left[C(G_2(U_m)) = 1\right]\right| &= \left|\mathbf{Pr}\left[C'(G_1(U_\ell)) = 1\right] - \mathbf{Pr}\left[C'(U_m) = 1\right]\right| \\ &\leq \varepsilon/2\end{aligned}$$

since $G_1$ is a $(t + t', \varepsilon/2)$-PRG. In conclusion:

$$\begin{aligned}&\left|\mathbf{Pr}\left[C(G_2(G_1(U_\ell))) = 1\right] - \mathbf{Pr}\left[C(U_n) = 1\right]\right| \\ \leq\ &\left|\mathbf{Pr}\left[C(G_2(G_1(U_\ell))) = 1\right] - \mathbf{Pr}\left[C(G_2(U_m)) = 1\right]\right| + \left|\mathbf{Pr}\left[C(G_2(U_m)) = 1\right] - \mathbf{Pr}\left[C(U_n) = 1\right]\right| \\ \leq\ &\varepsilon/2 + \varepsilon/2\end{aligned}$$

**Exercise 12.3:** Assume $\mathsf{P} = \mathsf{NP}$. Consider any poly-time computable function family $f_1, f_2, \ldots$ where $f_n : \{0,1\}^n \to \{0,1\}^n$. To show the family isn't $(\text{poly}\, n, \text{negl}\, n)$-one-way, we show that for some constant $d$ and infinitely many $n$ (in fact, for all large enough $n$), $f_n$ isn't $(n^d, 1/2)$-one-way.

(call this problem $A$)

Input: $y \in \{0,1\}^n$ and partial assignment $a \in \{0,1,*\}^n$

Output: Does there exist $z \in \{0,1\}^n$ consistent with $a$ such that $f_n(z) = y$?

$A \in \mathsf{NP}$ by a verifier $V$ where $V(y, a; z) = 1$ iff $z$ is consistent with $a$ and $f_n(z) = y$, using a poly-time program for $f_n$. Thus $A \in \mathsf{P}$, so for some constant $e$ and all large enough $N$, $A_N$ has a circuit of size $N^e$. Given $y = f_n(x)$, we can find some $z$ such that $f_n(z) = y$ by a search-to-decision reduction that makes $n$ queries to $A_N$ (where $N$ is $A$'s input length associated with $n$). Combining this with the circuit for $A_N$ yields a circuit $C : \{0,1\}^n \to \{0,1\}^n$ of size $n^d$ (for some constant $d$) such that $\mathbf{Pr}_{x \sim U_n}\left[f_n(C(f_n(x))) = f_n(x)\right] = 1 > 1/2$.

**Exercise 12.4:** Suppose $G$ isn't a $(t,\varepsilon)$-OWF. Let $C\colon \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$ be a size-$t$ circuit such that:

$$\mathbf{Pr}_{s\sim U_\ell}\Big[G(C(G(s))) = G(s)\Big] \;>\; \varepsilon$$

Define a circuit $C'\colon \{0,1\}^{2\ell} \to \{0,1\}$ such that $C'(x) = 1$ iff $G(C(x)) = x$. (That is, $C'$ tests whether $C$ successfully finds a preimage of $x$.) By assumption, $\mathbf{Pr}\Big[C'(G(U_\ell)) = 1\Big] > \varepsilon$. We have $\mathbf{Pr}\Big[C'(U_{2\ell}) = 1\Big] \leq 2^{\ell}/2^{2\ell} = 2^{-\ell}$ since at most $2^{\ell}$ of the $2^{2\ell}$ elements of $\{0,1\}^{2\ell}$ even have a preimage. It follows that:

$$\Big|\mathbf{Pr}\Big[C'(G(U_\ell)) = 1\Big] - \mathbf{Pr}\Big[C'(U_{2\ell}) = 1\Big]\Big| \;>\; \varepsilon - 2^{-\ell}$$

$C'$ has size $\leq t + t' + 8\ell$ since it consists of $C$ and a size-$t'$ circuit for $G$ and $8\ell - 1$ many $\wedge$ and $\vee$ gates to check whether two bit strings of length $2\ell$ are equal. Thus $G$ isn't a $\big(t + t' + 8\ell,\, \varepsilon - 2^{-\ell}\big)$-PRG.

**Exercise 12.5:** $f \circ f$ is a permutation since $f$ is a permutation. Suppose $f \circ f$ isn't a $(t, \varepsilon)$-OWP. Let $C : \{0,1\}^n \to \{0,1\}^n$ be a size-$t$ circuit such that:

$$\mathbf{Pr}_{x \sim U_n}\left[C(f(f(x))) = x\right] > \varepsilon$$

Define a circuit $C' : \{0,1\}^n \to \{0,1\}^n$ such that $C'(y) = C(f(y))$.

$$\mathbf{Pr}_{x \sim U_n}\left[C'(f(x)) = x\right] = \mathbf{Pr}_{x \sim U_n}\left[C(f(f(x))) = x\right] > \varepsilon$$

$C'$ has size $\leq t + t'$ since it consists of $C$ and a size-$t'$ circuit for $f$. Thus $f$ isn't a $(t + t', \varepsilon)$-OWP.

**Exercise 12.6:** Assume $f$ and $h$ have size-$t'$ circuits, and $f$ isn't $(t,\varepsilon)$-one-way. Let $C\colon \{0,1\}^n \to \{0,1\}^n$ be a size-$t$ circuit such that $\mathbf{Pr}_{x\sim U_n}\big[C(f(x)) = x\big] > \varepsilon$. Consider this randomized algorithm that takes input $y \in \{0,1\}^n$ and randomness $b \sim U_1$:

```
Π(y; b):
    evaluate z ← C(y)
    if f(z) = y: output h(z)
    else: output b
```

Since $f$ is a permutation, we have $f(z) = y$ iff $z = f^{-1}(y)$. By the law of total probability:

$$\begin{aligned}\mathbf{Pr}_{x\sim U_n,\, b\sim U_1}\big[\Pi(f(x);b) = h(x)\big] &= \mathbf{Pr}_x\big[C(f(x)) = x\big]\cdot 1 + \mathbf{Pr}_x\big[C(f(x)) \neq x\big]\cdot \mathbf{Pr}_{x,b}\big[b = h(x)\big]\\ &> \varepsilon + (1-\varepsilon)\cdot 1/2\\ &= (1+\varepsilon)/2\end{aligned}$$

For some $b \in \{0,1\}$, $\mathbf{Pr}_x\big[\Pi(f(x);b) = h(x)\big] > (1+\varepsilon)/2$. We implement $\Pi(y;b)$ as a circuit $C'(y)$, so $\mathbf{Pr}_x\big[C'(f(x)) = h(x)\big] > (1+\varepsilon)/2$. Thus $h$ isn't $\big(t+2t'+4n,\, \varepsilon\big)$-hardcore for $f$, because $C'$ has size $t + 2t' + 4n$ since it consists of $C$, and the circuits for $f$ and $h$, and $4n-1$ many $\wedge$ and $\vee$ gates for comparing $f(z)$ to $y$, and one gate for the final mux (contracted with $b$).

**Exercise 12.7:**

| $s$ | $G^4(s)$ |
|---|---|
| 00 | 0010 |
| 01 | 1001 |
| 10 | 0100 |
| 11 | 1001 |

| $x$ | $\mathbf{Pr}_{(U_1, G^3(U_2))}[x]$ |
|---|---|
| 0000 | 0 |
| 0001 | 1/8 |
| 0010 | 1/8 |
| 0011 | 0 |
| 0100 | 1/4 |
| 0101 | 0 |
| 0110 | 0 |
| 0111 | 0 |
| 1000 | 0 |
| 1001 | 1/8 |
| 1010 | 1/8 |
| 1011 | 0 |
| 1100 | 1/4 |
| 1101 | 0 |
| 1110 | 0 |
| 1111 | 0 |

**Exercise 12.8:** Suppose $G^n$ isn't a $(t, \varepsilon)$-PRG. Let $C\colon \{0,1\}^n \to \{0,1\}$ be a size-$t$ circuit such that:

$$\left|\mathbf{Pr}\big[C(G^n(U_\ell)) = 1\big] - \mathbf{Pr}\big[C(U_n) = 1\big]\right| > \varepsilon$$

Lemma 11.25 is stated for predicting the $i^{\text{th}}$ bit from the first $i-1$ bits, but by symmetry, it can also be stated for predicting the $i^{\text{th}}$ bit from the last $n-i$ bits. Thus there exist $i \in [n]$ and $C'\colon \{0,1\}^{n-i} \to \{0,1\}$ (a constant if $i = n$) such that

$$\mathbf{Pr}_{x \sim G^n(U_\ell)}\big[C'(x_{>i}) = x_i\big] > 1/2 + \varepsilon/n$$

and $C'(x_{>i}) = C(y_{\leq i}, x_{>i}) \oplus b$ for some hardwired $y_{\leq i} \in \{0,1\}^i$ and bit $b$. To show that $h$ isn't $\big(t + nt', 2\varepsilon/n\big)$-hardcore for $f$, define a circuit $C''\colon \{0,1\}^\ell \to \{0,1\}$ such that $C''(z) = C'(G^{n-i}(z))$.

$$\begin{aligned}
\mathbf{Pr}_{s \sim U_\ell}\big[C''(f(s)) = h(s)\big] &= \mathbf{Pr}_{s \sim U_\ell}\big[C'(G^{n-i}(f(s))) = h(s)\big] \\
&= \mathbf{Pr}_{s \sim U_\ell}\big[C'(G^{n-i+1}(s)_{>1}) = G^{n-i+1}(s)_1\big] \\
&= \mathbf{Pr}_{x_{\geq i} \sim G^{n-i+1}(U_\ell)}\big[C'(x_{>i}) = x_i\big] \\
&= \mathbf{Pr}_{x \sim G^n(U_\ell)}\big[C'(x_{>i}) = x_i\big] \qquad (f \text{ is a permutation}) \\
&> (1 + 2\varepsilon/n)/2
\end{aligned}$$

$C''$ has size $\leq t + nt'$ since it consists of the size-$t$ circuit $C'$ and at most $n$ copies of the size-$t'$ circuit for $G$.

**Exercise 12.9:** We prove that if $(E, D)$ isn't a $(t, \varepsilon)$-secure private-key encryption scheme, then $h$ isn't $(t, \varepsilon)$-hardcore for $f$. Let $C : \{0,1\}^{\ell+1} \to \{0,1\}$ be a size-$t$ circuit such that

$$\mathbf{Pr}\big[C(E_{U_\ell}(0)) = 1\big] - \mathbf{Pr}\big[C(E_{U_\ell}(1)) = 1\big] > \varepsilon$$

(assuming $C$ has already been negated if necessary to remove the absolute value bars). For each bit $b$, define a circuit $C'_b : \{0,1\}^\ell \to \{0,1\}$ such that:

$$C'_b(y) = \begin{cases} b & \text{if } C(y,b) = 1 \\ \overline{b} & \text{if } C(y,b) = 0 \end{cases}$$

$C'_1$ is $C$ with the last input bit hardwired to 1, and $C'_0$ is $C$ with the last input bit hardwired to 0 and with the output negated. Thus $C'_1$ and $C'_0$ are size-$t$ circuits. Using the law of total probability:

$$\begin{aligned}
\max_b \Big(\mathbf{Pr}_k\big[C'_b(f(k)) = h(k)\big]\Big) &\geq \mathbf{Pr}_{k,b}\big[C'_b(f(k)) = h(k)\big] \\
&= \mathbf{Pr}_{k,b}\big[b = h(k)\big] \cdot \mathbf{Pr}_{k,b}\big[C'_b(f(k)) = h(k) \,\big|\, b = h(k)\big] + \\
&\quad \mathbf{Pr}_{k,b}\big[b = \overline{h(k)}\big] \cdot \mathbf{Pr}_{k,b}\big[C'_b(f(k)) = h(k) \,\big|\, b = \overline{h(k)}\big] \\
&= \tfrac{1}{2} \cdot \mathbf{Pr}_k\big[C'_{h(k)}(f(k)) = h(k)\big] + \tfrac{1}{2} \cdot \mathbf{Pr}_k\big[C'_{\overline{h(k)}}(f(k)) = h(k)\big] \\
&= \Big(\mathbf{Pr}_k\big[C(f(k), h(k)) = 1\big] + \mathbf{Pr}_k\big[C(f(k), \overline{h(k)}) = 0\big]\Big)/2 \\
&= \Big(\mathbf{Pr}_k\big[C(E_k(0)) = 1\big] + \mathbf{Pr}_k\big[C(E_k(1)) = 0\big]\Big)/2 \\
&= \Big(1 + \mathbf{Pr}_k\big[C(E_k(0)) = 1\big] - \mathbf{Pr}_k\big[C(E_k(1)) = 1\big]\Big)/2 \\
&> (1 + \varepsilon)/2
\end{aligned}$$

Thus $\mathbf{Pr}_k\big[C'_b(f(k)) = h(k)\big] > (1+\varepsilon)/2$ for some $b$.

**Exercise 12.10.a:** Since $G$ is a $(t, \varepsilon/2)$-PRG, $E_{a,U_\ell}(0) = G(U_\ell)$ and $E_{a,U_\ell}(1) = G(U_\ell) \oplus a$ are $(t, \varepsilon)$-indistinguishable for each $a$ (Exercise 11.6).

**Exercise 12.10.b:**

$$\begin{aligned}
\mathbf{Pr}_a\left[\exists x,k,k' :\ D_{a,k}(x)=0 \text{ and } D_{a,k'}(x)=1\right] &= \mathbf{Pr}_a\left[\exists k,k' :\ G(k)=G(k')\oplus a\right] \\
&\leq \sum_{k,k'} \mathbf{Pr}_a\left[G(k)=G(k')\oplus a\right] \quad \text{(union bound)} \\
&= \sum_{k,k'} 2^{-3\ell} \\
&= 2^{-\ell}
\end{aligned}$$

**Exercise 12.11:** Assume $\ell \leq \log t - \log\log t - 1$ and $\ell < 2^n$, and consider any $F\colon \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}$. Identify $\{0,1\}^n$ with the binary numbers $\{0,1,\ldots,2^n-1\} \supseteq \{0,1,\ldots,\ell\}$. Consider an oracle circuit $C$ such that $C(f)$ queries $f(i)$ for each $i \in \{0,1,\ldots,\ell\}$ and outputs 1 iff $\big(f(0),f(1),\ldots,f(\ell)\big) \in \{0,1\}^{\ell+1}$ equals $\big(F_s(0),F_s(1),\ldots,F_s(\ell)\big)$ for at least one $s \in \{0,1\}^\ell$. We have $\mathbf{Pr}\big[C(F_{U_\ell}) = 1\big] = 1$ and $\mathbf{Pr}\big[C(U_{n\to 1}) = 1\big] \leq 2^\ell/2^{\ell+1} = 1/2$ since at most $2^\ell$ of the $2^{\ell+1}$ strings in $\{0,1\}^{\ell+1}$ equal $\big(F_s(0),F_s(1),\ldots,F_s(\ell)\big)$ for some $s \in \{0,1\}^\ell$. It follows that:

$$\Big|\mathbf{Pr}\big[C(F_{U_\ell}) = 1\big] - \mathbf{Pr}\big[C(U_{n\to 1}) = 1\big]\Big| \;\geq\; 1/2 \;>\; 1/3$$

$C$ has size $\leq (\ell+1) + \ell 2^\ell + (2^\ell - 1) \leq 2\ell 2^\ell \leq t$. Thus $F$ isn't a $(t, 1/3)$-PRF.

**Exercise 12.12:**

| $s$ | $G(s)$ | $G_0(s)$ | $G_1(s)$ |
|---|---|---|---|
| 00 | 0011 | 00 | 11 |
| 01 | 1000 | 10 | 00 |
| 10 | 1101 | 11 | 01 |
| 11 | 0110 | 01 | 10 |

| $x$ | $F_{11}(x)$ |
|---|---|
| 000 | $G_0(G_0(G_0(11))) = 11$ |
| 001 | $G_1(G_0(G_0(11))) = 01$ |
| 010 | $G_0(G_1(G_0(11))) = 00$ |
| 011 | $G_1(G_1(G_0(11))) = 11$ |
| 100 | $G_0(G_0(G_1(11))) = 01$ |
| 101 | $G_1(G_0(G_1(11))) = 10$ |
| 110 | $G_0(G_1(G_1(11))) = 10$ |
| 111 | $G_1(G_1(G_1(11))) = 00$ |

**Exercise 12.13:** Consider the size-$(4n+1)$ oracle circuit that queries $(a,b) \leftarrow f(x,y)$ and $(a',b') \leftarrow f(x',y)$ for some arbitrary $x \neq x'$ and $y$, and accepts iff $x \oplus a = x' \oplus a'$. If $f \sim F'^{2}_{U_{2\ell}}$ then this accepts with probability 1 because:

$$x \oplus a \;=\; x \oplus \big(x \oplus F_{s_1}(y)\big) \;=\; F_{s_1}(y) \;=\; x' \oplus \big(x' \oplus F_{s_1}(y)\big) \;=\; x' \oplus a'$$

If $f \sim U_{2n \to 2n}$ then this accepts with probability $2^{-n}$ because $a$ and $a'$ are independent of each other since $(x,y) \neq (x',y)$, and thus $a = x \oplus (x' \oplus a')$ holds with probability $2^{-n}$.

**Exercise 12.14:** Say an outcome $q = (r, r')$ is *good* iff $r \neq r'$. We have $\mathbf{Pr}_q\big[q \text{ is bad}\big] = 2^{-n}$. For each $q = (r, r')$, consider an oracle circuit $C_{0,q}$ such that

$$C_{0,q}(f) \;=\; C\Big(\big(r,\, m_0 \oplus f(r)\big),\; \big(r',\, B\big(r,\, m_0 \oplus f(r)\big) \oplus f(r')\big)\Big)$$

and an analogous oracle circuit $C_{1,q}$ using $m_1$ instead of $m_0$. Each $C_{0,q}$ and $C_{1,q}$ has size $\leq 2t + 3n + 2$ because it consists of $C$ and $B$, and two oracle gates, and $3n$ many $\wedge$ and $\vee$ gates for bitwise xoring $B(\ldots)$ with $f(r')$, and some $\neg$ gates and constants. If $q$ is good then

$$C_{0,q}(U_{n\to n}) \;=\; C\big((r,\, U_n),\, (r',\, U_n)\big) \;=\; C_{1,q}(U_{n\to n})$$

where the two $U_n$s in the middle are independent, since $f(r)$ and $f(r')$ are independent. Putting everything together:

$$\begin{aligned}
&\Big|\mathbf{Pr}_{k,q}\big[C\big(E_{k,r}(m_0),\, E_{k,r'}(B(E_{k,r}(m_0)))\big) = 1\big] - \mathbf{Pr}_{k,q}\big[C\big(E_{k,r}(m_1),\, E_{k,r'}(B(E_{k,r}(m_1)))\big) = 1\big]\Big| \\
=\;& \Big|\mathbf{Pr}_{k,q}\big[C_{0,q}(F_k) = 1\big] - \mathbf{Pr}_{k,q}\big[C_{1,q}(F_k) = 1\big]\Big| \\
\leq\;& \mathbf{E}_q\Big|\mathbf{Pr}\big[C_{0,q}(F_{U_\ell}) = 1\big] - \mathbf{Pr}\big[C_{1,q}(F_{U_\ell}) = 1\big]\Big| \\
\leq\;& \max_{\text{good } q}\Big|\mathbf{Pr}\big[C_{0,q}(F_{U_\ell}) = 1\big] - \mathbf{Pr}\big[C_{1,q}(F_{U_\ell}) = 1\big]\Big| + \mathbf{Pr}_q\big[q \text{ is bad}\big] \\
\leq\;& \max_{\text{good } q}\Big(\big|\mathbf{Pr}\big[C_{0,q}(F_{U_\ell}) = 1\big] - \mathbf{Pr}\big[C_{0,q}(U_{n\to n}) = 1\big]\big| + \\
&\qquad\qquad \big|\mathbf{Pr}\big[C_{1,q}(U_{n\to n}) = 1\big] - \mathbf{Pr}\big[C_{1,q}(F_{U_\ell}) = 1\big]\big| \;\Big) + \mathbf{Pr}_q\big[q \text{ is bad}\big] \\
\leq\;& \max_{\text{good } q}\big((\varepsilon - 2^{-n})/2 + (\varepsilon - 2^{-n})/2\big) + 2^{-n} \\
=\;& \varepsilon
\end{aligned}$$

**Exercise 12.15:** Assume $F$ is pairwise uniform. Consider any $m \in \{0,1\}^n$ and any $g : \{0,1\}^b \to \{0,1\}^n \times \{0,1\}^b$. For any $a \in \{0,1\}^b$, $g(a) = (m', a')$ is a forgery iff $a' = F_k(m')$ and $m' \neq m$, so since $F$ is pairwise uniform:

$$\begin{aligned}\mathbf{Pr}_{k\sim U_\ell}\left[g(a) \text{ is a forgery} \,\middle|\, F_k(m) = a\right] &= \begin{cases} 0 & \text{if } m' = m \\ \mathbf{Pr}_{k\sim U_\ell}\left[F_k(m') = a' \,\middle|\, F_k(m) = a\right] = 2^{-b} & \text{if } m' \neq m \end{cases} \\ &\leq 2^{-b}\end{aligned}$$

By the law of total probability:

$$\begin{aligned}\mathbf{Pr}_{k\sim U_\ell}\left[g(F_k(m)) \text{ is a forgery}\right] &= \sum_a \mathbf{Pr}_{k\sim U_\ell}\left[F_k(m) = a\right] \cdot \mathbf{Pr}_{k\sim U_\ell}\left[g(a) \text{ is a forgery} \,\middle|\, F_k(m) = a\right] \\ &\leq \sum_a \mathbf{Pr}_{k\sim U_\ell}\left[F_k(m) = a\right] \cdot 2^{-b} \\ &= 2^{-b}\end{aligned}$$

Thus $F$ is a one-time perfectly unforgeable MAC.

**Exercise 12.16:** Consider any $m \in \{0,1\}^n$ and $m' \in \{0,1\}^n$ with $m \neq m'$. Define a size-2 oracle circuit that queries $(a_1, a_2) \leftarrow F'_{k_1,k_2}(m,m)$ and $(a'_1, a'_2) \leftarrow F'_{k_1,k_2}(m',m')$ and outputs $\big((m,m'),(a_1,a'_2)\big)$. This is a forgery (with probability 1) since $F'_{k_1,k_2}(m,m') = \big(F_{k_1}(m), F_{k_2}(m')\big) = (a_1, a'_2)$ and $(m,m') \neq (m,m)$ and $(m,m') \neq (m',m')$ (since $m \neq m'$). Thus $F'$ isn't $(2,\varepsilon)$-unforgeable if $\varepsilon < 1$.

**Exercise 13.1.a:** Here it is:

$L\big((x^1, c(x^1)), \ldots, (x^m, c(x^m))\big)$:
    if $c(x^i) = 1$ for some $i$:
        output the hypothesis $h = c_{a,b}$ where:
            $a$ is the minimum $x^i$ such that $c(x^i) = 1$
            $b$ is the maximum $x^i$ such that $c(x^i) = 1$
    else: output the constant-0 hypothesis $h$

**Exercise 13.1.b:** There are $\binom{2^n}{2}$ concepts $c_{\ell,u}$ with $\ell < u$. There are $2^n$ concepts $c_{\ell,u}$ with $\ell = u$. There is 1 distinct concept $c_{\ell,u}$ with $\ell > u$. Thus $|C| = \binom{2^n}{2} + 2^n + 1 = (2^{2n} + 2^n + 2)/2 \leq 2^{2n}$. By Theorem 13.4, if $m \geq \frac{1}{\varepsilon} \ln \frac{|H|}{\delta}$ then every $m$-consistent learner for $C$ is a $(\delta, \varepsilon)$-PAC learner for $C$. Thus some $\frac{1}{\varepsilon}\left(O(n) + \ln \frac{1}{\delta}\right)$ training examples suffice.

**Exercise 13.1.c:** Consider any target concept $c = c_{\ell,u}$ and any distribution $D$ over $\{0,1\}^n$. If $\ell > u$ then $c$ is the constant-0 concept and $L$ outputs the constant-0 hypothesis, so assume $\ell \leq u$. We use the interval notation $[\ell,u] = \{x : \ell \leq x \leq u\}$ from §13.1.2. If $D([\ell,u]) \leq \varepsilon$ then

$$\mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\Big[\mathbf{Pr}_{y\sim D}\big[h(y) = c(y)\big] \geq 1-\varepsilon\Big] \;=\; 1$$

since either $h$ is the constant-0 hypothesis, in which case $\mathbf{Pr}_{y\sim D}\big[h(y) \neq c(y)\big] = D([\ell,u]) \leq \varepsilon$, or $\ell \leq a \leq b \leq u$, in which case $\mathbf{Pr}_{y\sim D}\big[h(y) \neq c(y)\big] = D([\ell,a-1] \cup [b+1,u]) \leq D([\ell,u]) \leq \varepsilon$. Now, assume $D([\ell,u]) > \varepsilon$ and consider the smallest $\ell' \geq \ell$ such that $D([\ell,\ell']) > \varepsilon/2$ and the largest $u' \leq u$ such that $D([u',u]) > \varepsilon/2$. If $a \in [\ell,\ell']$ and $b \in [u',u]$ then:

$$\begin{aligned}
\mathbf{Pr}_{y\sim D}\big[h(y) \neq c(y)\big] &= D([\ell,a-1] \cup [b+1,u]) \\
&\leq D([\ell,\ell'-1]) + D([u'+1,u]) \\
&\leq \varepsilon/2 + \varepsilon/2 \\
&= \varepsilon
\end{aligned}$$

Thus:

$$\begin{aligned}
&\mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\Big[\mathbf{Pr}_{y\sim D}\big[h(y) = c(y)\big] < 1-\varepsilon\Big] \\
&\leq \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[a \notin [\ell,\ell'] \text{ or } b \notin [u',u]\big] \\
&\leq \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[a \notin [\ell,\ell']\big] + \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[b \notin [u',u]\big] && \text{(union bound)} \\
&\leq \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\forall i :\; x^i \notin [\ell,\ell']\big] + \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\forall i :\; x^i \notin [u',u]\big] \\
&= \textstyle\prod_{i=1}^m \mathbf{Pr}_{x^i\sim D}\big[x^i \notin [\ell,\ell']\big] + \prod_{i=1}^m \mathbf{Pr}_{x^i\sim D}\big[x^i \notin [u',u']\big] && (x^1,\ldots,x^m \text{ are independent}) \\
&= \textstyle\prod_{i=1}^m (1 - D([\ell,\ell'])) + \prod_{i=1}^m (1 - D([u',u])) \\
&\leq 2(1-\varepsilon/2)^m \\
&\leq 2\mathrm{e}^{-m\varepsilon/2} && \text{(Fact 7.6)} \\
&\leq \delta
\end{aligned}$$

**Exercise 13.2.a:** Given $(x^1, c(x^1)), \ldots, (x^m, c(x^m))$, the learner outputs the hypothesis $h = c^T$ where $T = \{x^i : c(x^i) = 1\}$. Consider any concept $c = c^S$ where $|S| \leq k$ and any distribution $D$ over $\{0,1\}^n$. Note that $T = S \cap \{x^1, \ldots, x^m\} \subseteq S$ and thus $|T| \leq k$, so $h \in C$. Say $y \in S$ is *heavy* if $D(y) > \varepsilon/k$ and *light* otherwise. If every heavy $y \in S$ is in $\{x^1, \ldots, x^m\}$ and therefore in $T$, then:

$$\mathbf{Pr}_{y \sim D}\big[h(y) \neq c(y)\big] = \sum_{y \in S \setminus T} D(y) \leq \sum_{\text{light } y \in S} D(y) \leq |S| \cdot \varepsilon/k \leq \varepsilon$$

Thus:

$$\begin{aligned}
& \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\Big[\mathbf{Pr}_{y\sim D}\big[h(y) = c(y)\big] < 1 - \varepsilon\Big] \\
\leq\ & \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\text{some heavy } y \in S \text{ is not in } \{x^1, \ldots, x^m\}\big] \\
\leq\ & \sum_{\text{heavy } y \in S} \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\forall i :\ x^i \neq y\big] && \text{(union bound)} \\
=\ & \sum_{\text{heavy } y \in S} \prod_{i=1}^m \mathbf{Pr}_{x^i \sim D}\big[x^i \neq y\big] && (x^1, \ldots, x^m \text{ are independent}) \\
=\ & \sum_{\text{heavy } y \in S} \prod_{i=1}^m (1 - D(y)) \\
\leq\ & \sum_{\text{heavy } y \in S} (1 - \varepsilon/k)^m \\
\leq\ & k e^{-m\varepsilon/k} && (|S| \leq k \text{ and Fact 7.6}) \\
\leq\ & \delta
\end{aligned}$$

**Exercise 13.2.b:** Given $(x^1, c(x^1)), \ldots, (x^m, c(x^m))$, the learner outputs the hypothesis $h = c^T$ where $T$ contains $\{x^i : c(x^i) = 1\}$ along with arbitrary elements of $\{0,1\}^n \setminus \{x^1, \ldots, x^m\}$ to bring the size up to $|T| = k$ (so $h \in C$ assuming $c \in C$).

Now, consider any $m$-consistent learner for $C$ (not necessarily the one we just designed). Consider any concept $c = c^S$ where $|S| = k$ and any distribution $D$ over $\{0,1\}^n$. Suppose that given $(x^1, c(x^1)), \ldots, (x^m, c(x^m))$, the learner outputs some hypothesis $h = c^T$ with $|T| = k$. Then $S \cap \{x^1, \ldots, x^m\} = T \cap \{x^1, \ldots, x^m\}$ since the learner is consistent. Say $y \in \{0,1\}^n$ is *heavy* if $D(y) > \varepsilon/2k$ and *light* otherwise. If every heavy $y$ is in $\{x^1, \ldots, x^m\}$ and thus not in $(S \setminus T) \cup (T \setminus S)$, then:

$$\mathbf{Pr}_{y \sim D}\big[h(y) \neq c(y)\big] = \sum_{y \in (S \setminus T) \cup (T \setminus S)} D(y) \leq \sum_{\text{light } y \in S \cup T} D(y) \leq |S \cup T| \cdot \varepsilon/2k \leq \varepsilon$$

Thus:

$$\begin{aligned}
& \mathbf{Pr}_{(x^1,\ldots,x^m) \sim D^m}\Big[\mathbf{Pr}_{y \sim D}\big[h(y) = c(y)\big] < 1 - \varepsilon\Big] \\
\leq\ & \mathbf{Pr}_{(x^1,\ldots,x^m) \sim D^m}\big[\text{some heavy } y \text{ is not in } \{x^1, \ldots, x^m\}\big] \\
\leq\ & \sum_{\text{heavy } y} \mathbf{Pr}_{(x^1,\ldots,x^m) \sim D^m}\big[\forall i : x^i \neq y\big] && \text{(union bound)} \\
=\ & \sum_{\text{heavy } y} \prod_{i=1}^m \mathbf{Pr}_{x^i \sim D}\big[x^i \neq y\big] && (x^1, \ldots, x^m \text{ are independent}) \\
=\ & \sum_{\text{heavy } y} \prod_{i=1}^m (1 - D(y)) \\
\leq\ & \sum_{\text{heavy } y} (1 - \varepsilon/2k)^m \\
\leq\ & (2k/\varepsilon)\mathrm{e}^{-m\varepsilon/2k} && (\leq 2k/\varepsilon \text{ many } y \text{ are heavy and Fact 7.6}) \\
\leq\ & \delta
\end{aligned}$$

**Exercise 13.3:** This is like the proof of Theorem 13.4. Assume $m \geq \frac{2}{\varepsilon} \ln \frac{|H|}{\delta\varepsilon} + 1$ and $L$ is an $m$-nearly consistent learner for $(C, H)$. Consider any concept $c \in C$ and distribution $D$ over $\{0,1\}^n$. Say a hypothesis $h \in H$ is *bad* iff $\mathbf{Pr}_{y \sim D}\big[h(y) = c(y)\big] < 1 - \varepsilon$. Then:

$$
\begin{aligned}
& \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\Big[L\big((x^1, c(x^1)), \ldots, (x^m, c(x^m))\big) \text{ is bad}\Big] \\
\leq\ & \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\Big[\exists \text{ bad } h \text{ nearly consistent with } (x^1, c(x^1)), \ldots, (x^m, c(x^m))\Big] \\
\leq\ & \textstyle\sum_{\text{bad } h} \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\exists i \in [m]\ \forall j \neq i :\ h(x^j) = c(x^j)\big] && \text{(union bound)} \\
\leq\ & \textstyle\sum_{\text{bad } h} \sum_{i=1}^{m} \mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\big[\forall j \neq i :\ h(x^j) = c(x^j)\big] && \text{(union bound)} \\
=\ & \textstyle\sum_{\text{bad } h} \sum_{i=1}^{m} \prod_{j \neq i} \mathbf{Pr}_{x^j \sim D}\big[h(x^j) = c(x^j)\big] && (x^1, \ldots, x^m \text{ are independent}) \\
\leq\ & \textstyle\sum_{\text{bad } h} \sum_{i=1}^{m} (1-\varepsilon)^{m-1} \\
\leq\ & |H| m \mathrm{e}^{-\varepsilon(m-1)} && \text{(Fact 7.6)} \\
=\ & |H|\big(\tfrac{2}{\varepsilon} \ln \tfrac{|H|}{\delta\varepsilon} + 1\big)\big(\tfrac{\delta\varepsilon}{|H|}\big)^2 && \text{(assuming } m = \tfrac{2}{\varepsilon} \ln \tfrac{|H|}{\delta\varepsilon} + 1\text{; see below)} \\
=\ & \big(2 \ln \tfrac{|H|}{\delta\varepsilon} + \varepsilon\big)\big(\tfrac{\delta\varepsilon}{|H|}\big)\delta \\
\leq\ & \delta
\end{aligned}
$$

If we increase $m$, then the inequality still holds since $\mathrm{e}^{\varepsilon(m-1)}$ (with no $-$ sign in the exponent) grows faster than $m$. Thus $L$ is a $(\delta, \varepsilon)$-PAC learner for $(C, H)$.

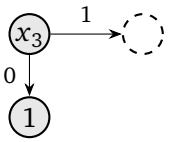
**Exercise 13.4:** Ignore $x^2 = 000101$ and $x^5 = 101010$, and form this matrix:

$$\begin{bmatrix} x^1 \\ x^3 \\ x^4 \\ x^6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

*(1)* Exclude $x_1$ and $\overline{x_1}$ since column 1 has a 0 and a 1.
*(2)* Include $\overline{x_2}$ since column 2 is all-0.
*(3)* Exclude $x_3$ and $\overline{x_3}$ since column 3 has a 0 and a 1.
*(4)* Include $x_4$ since column 4 is all-1.
*(5)* Exclude $x_5$ and $\overline{x_5}$ since column 5 has a 0 and a 1.
*(6)* Include $\overline{x_6}$ since column 6 is all-0.

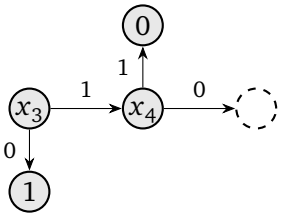
The term $\overline{x_2} \wedge x_4 \wedge \overline{x_6}$ accepts all four positive training examples and rejects both negative training examples.

**Exercise 13.5:** Among the training examples, if $x_3 = 0$ then $c(x) = 1$, so we pick $(3, 0, 1)$.



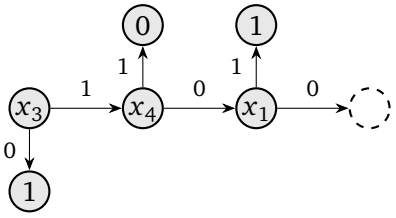

$(0010, 0)$ $(0101, 1)$
$(1111, 0)$ $(0110, 1)$
$(1010, 1)$

Among the remaining training examples, if $x_4 = 1$ then $c(x) = 0$, so we pick $(4, 1, 0)$.



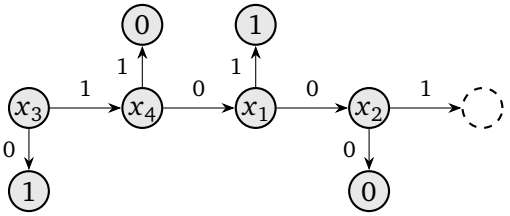

$(0010, 0)$ $(0101, 1)$
$(1111, 0)$ $(0110, 1)$
$(1010, 1)$

Among the remaining training examples, if $x_1 = 1$ then $c(x) = 1$, so we pick $(1, 1, 1)$. (Or, we could pick $(2, 1, 1)$.)



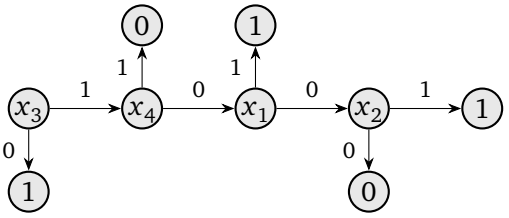

$(0010, 0)$ $(0101, 1)$
$(1111, 0)$ $(0110, 1)$
$(1010, 1)$

Among the remaining training examples, if $x_2 = 0$ then $c(x) = 0$, so we pick $(2, 0, 0)$. (Or, we could pick $(2, 1, 1)$.)



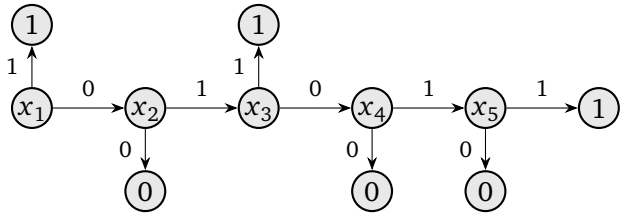

$(0010, 0)$ $(0101, 1)$
$(1111, 0)$ $(0110, 1)$
$(1010, 1)$

All remaining training examples have $c(x) = 1$, so we have a terminal node labeled 1.




$(0010, 0)$ $(0101, 1)$
$(1111, 0)$ $(0110, 1)$
$(1010, 1)$

**Exercise 13.6:**



**Exercise 13.7:** $\Leftarrow$: Assume that for every nonempty $S \subseteq \{0,1\}^n$, either $c$ is constant on $S$ or there exist $b \in \{0,1\}$ and a literal $\ell$ that accepts a string in $S \cap c^{-1}(b)$ but none in $S \cap c^{-1}(\overline{b})$. Since a literal $\ell$ has the form "$x_j = a$" for some $j \in [n]$ and $a \in \{0,1\}$, the assumption is equivalent to: For every nonempty $S \subseteq \{0,1\}^n$, either $c$ is constant on $S$ or there exists $(j,a,b)$ such that: $c(x) = b$ for all $x \in S$ with $x_j = a$, and such an $x$ exists. Thus if we run the algorithm from § 13.2.3 with training data $(x, c(x))$ for all $x \in \{0,1\}^n$, it won't get stuck: In each iteration, letting $S$ be the remaining training examples, the assumption implies that the algorithm either halts (if $c$ is constant on $S$) or picks a $(j,a,b)$ that makes progress. Thus there exists a decision list for $c$, since the algorithm finds one.

$\Rightarrow$: Assume there exists a decision list for $c$. Consider any nonempty $S \subseteq \{0,1\}^n$ such that $c$ is not constant on $S$. Consider the first $(j,a,b)$ in the decision list such that $x_j = a$ for some $x \in S$. Then $c(x) = b$ and the literal "$x_j = a$" accepts this $x \in S \cap c^{-1}(b)$ but accepts no string in $S \cap c^{-1}(\overline{b})$ (since otherwise the decision list would incorrectly output $b$ on the latter strings).

**Exercise 13.8:** We use the poly-time proper consistent learner for terms (§13.2.2) to design a learner for 2-term DNFs in which at least one variable appears positively in one term and negatively in the other term. The key observation is that if one term $T_1$ contains $x_j$ and the other term $T_0$ contains $\overline{x_j}$, then the DNF's output is the value of $T_1$ on all examples with $x_j = 1$, and is the value of $T_0$ on all examples with $x_j = 0$. So $x_j$ is the selector for a mux that picks which term to evaluate. We just try all possibilities of the selector variable index $j$.

given labeled examples $E = \big((x^i, \varphi(x^i)) : i \in [m]\big)$:
  for each $j \in [n]$:
    run the term learner on $E_1 = \big((x^i, \varphi(x^i)) : x^i_j = 1\big)$ to get a term $T_1$ containing $x_j$
    run the term learner on $E_0 = \big((x^i, \varphi(x^i)) : x^i_j = 0\big)$ to get a term $T_0$ containing $\overline{x_j}$
    if $T_1$ rejects all negative examples in $E_1$ and $T_0$ rejects all negative examples in $E_0$:
      halt and output the hypothesis $\psi = T_1 \vee T_0$

First, we claim that in iteration $j$, we may indeed assume $T_1$ contains $x_j$ and $T_0$ contains $\overline{x_j}$. If $E_1$ has at least one positive example, then $T_1$ contains $x_j$ since $T_1$ consists of all literals that evaluate to 1 on all positive examples in $E_1$. If $E_1$ has no positive example, we may assume $T_1$ is a width-$n$ term that rejects all examples in $E_1$ and still contains $x_j$. Similarly, $T_0$ contains $\overline{x_j}$.

Next, we claim that if this learner outputs a hypothesis $\psi$ in iteration $j$, then $\psi$ is consistent with the training data. For each $i \in [m]$:

- If $\varphi(x^i) = 1$ then $\psi(x^i) = 1$ because:
  - If $x^i_j = 1$ then $T_1(x^i) = 1$.
  - If $x^i_j = 0$ then $T_0(x^i) = 1$.
- If $\varphi(x^i) = 0$ then $\psi(x^i) = 0$ because if $x^i_j = 1$ (or analogously if $x^i_j = 0$):
  - $T_1(x^i) = 0$ since $T_1$ rejects all negative examples in $E_1$.
  - $T_0(x^i) = 0$ since $T_0$ contains $\overline{x_j}$.

Finally, we claim that if the training data comes from a target concept $\varphi$, then the learner outputs a hypothesis before running out of options for $j$. Assume $\varphi$ has $x_j$ in one term $U_1$ and $\overline{x_j}$ in the other term $U_0$. Then we claim that the learner would halt in iteration $j$ (if not sooner): All positive examples in $E_1$ are rejected by $U_0$ (since $U_0$ contains $\overline{x_j}$) and thus accepted by $U_1$. Thus $T_1$ contains all literals of $U_1$ (and possibly more). Since $U_1$ rejects all negative examples in $E_1$, so does $T_1$. Similarly, $T_0$ rejects all negative examples in $E_0$. Thus the learner would halt in iteration $j$.

The learner is poly-time since the term learner is poly-time and the $j \in [n]$ loop is only a poly-time overhead.

**Exercise 13.9:** Let $C'$ be the concept class of dictatorships on $|C|$ variables, and index these variables by the concepts $c \in C$.

- $F_{\text{ex}}: \{0,1\}^n \to \{0,1\}^{|C|}$: Define $x' = F_{\text{ex}}(x)$ by $x'_c = c(x)$ for all $c$.
- $F_{\text{con}}: C \to C'$: Define $c' = F_{\text{con}}(c)$ by $c'(x') = x'_c$ for all $x'$ (the dictatorship corresponding to index $c$). Note that $c(x) = c'(x')$.
- $F_{\text{hyp}}: C' \to C$: By definition of $C'$, for every $h' \in C'$, there exists $h \in C$ such that $h'(x') = x'_h$ for all $x'$. Define $F_{\text{hyp}}(h') = h$. Note that $h(x) = h'(x')$.

**Exercise 13.10.a:**

- $F_{\text{ex}}: \{0,1\}^n \to \{0,1\}^{n+1}$: Define $x' = F_{\text{ex}}(x) = x0$ (just appending a 0).
- $F_{\text{con}}: C \to C'$: For each point function $c =$ "$x = a$", define $c' = F_{\text{con}}(c) =$ "$x' = a0$". For the constant-0 function $c$, define $c' = F_{\text{con}}(c) =$ "$x' = a1$" for some $a \in \{0,1\}^n$ (it doesn't matter which $a$). Note that $c(x) = c'(x')$ in all cases.
- $F_{\text{hyp}}: C' \to C$: If $h' =$ "$x' = a0$" then define $h = F_{\text{hyp}}(h') =$ "$x = a$". If $h' =$ "$x' = a1$" then define $h = F_{\text{hyp}}(h') =$ the constant-0 function. Note that $h(x) = h'(x')$ in all cases.

**Exercise 13.10.b:** Suppose for contradiction $(F_{\text{ex}}, F_{\text{con}}, F_{\text{hyp}})$ is a learning reduction from $C$ to $C'$. Let $c^{1\prime} = F_{\text{con}}(c^1)$ and $c^{2\prime} = F_{\text{con}}(c^2)$ and $x^{1\prime} = F_{\text{ex}}(x^1)$ and $x^{2\prime} = F_{\text{ex}}(x^2)$. Since $c^{1\prime}(x^{1\prime}) = c^1(x^1) = 0$ and $c^{1\prime}(x^{2\prime}) = c^1(x^2) = 1$, we have $x^{1\prime} \neq x^{2\prime}$. Since $c^{2\prime}(x^{1\prime}) = c^2(x^1) = 1$ and $c^{2\prime}(x^{2\prime}) = c^2(x^2) = 1$ and $c^{2\prime}$ is a point function, we have $x^{1\prime} = x^{2\prime}$. This is a contradiction.

**Exercise 13.11:** Let $\varphi$ be a 4-clause CNF on $n \geq 4$ variables. Then $\overline{\varphi}$ is a 4-term DNF. By the proof of Lemma 13.8 (but with 4 instead of 3 everywhere), $\overline{\varphi}$ is equivalent to a 4-CNF $\overline{\psi}$, whose negation $\psi$ is a 4-DNF equivalent to $\varphi$.

**Exercise 13.12.a:** Let $A$ be Consistency for $C$. As shown in the proof of Theorem 13.7, it suffices to prove $A$ is NP-hard. Since Independent Set is NP-complete (Theorem 2.16), it suffices to show Independent Set $\leq_p^m A$. Map $(G,k)$ to an input to $A$ as follows. Assume $G$ has $n$ nodes, numbered $1,\ldots,n$. We use the same $n$ in the input to $A$, so an example $(x,t)$ has the bits of $x = x_1 \cdots x_n$ indexed by $G$'s nodes.

- We have a positive example $(1^n,k)$.
- For each edge $\{v,w\}$ in $G$, we have a negative example $(x,2)$ where $x$ has 1s at indices $v$ and $w$ and 0s elsewhere.

To show this poly-time mapping reduction is correct, we argue that $G$ has an independent set of size $\geq k$ iff there exists $c_I \in C$ consistent with the training data:

$\Rightarrow$: Assume $G$ has an independent set $I \subseteq [n]$ of size $\geq k$. Then $c_I$ is consistent with the training data because:

- $c_I(1^n,k) = 1$ since $\sum_{i \in I} 1 = |I| \geq k$.
- For each edge $\{v,w\}$ in $G$, the associated negative example $(x,2)$ has $c_I(x,2) = 0$ since $\sum_{i \in I} x_i \leq 1$ since $I$ is independent and therefore contains at most one of $v,w$.

$\Leftarrow$: Assume $c_I$ is consistent with the training data. Then $I$ is an independent set of size $\geq k$ because:

- Since $c_I(1^n,k) = 1$, we have $|I| = \sum_{i \in I} 1 \geq k$.
- For each edge $\{v,w\}$ in $G$, since the associated negative example $(x,2)$ has $c_I(x,2) = 0$, we have $\sum_{i \in I} x_i \leq 1$ and thus $I$ contains at most one of $v,w$.

**Exercise 13.12.b:** Let $A$ be Consistency for $C$. As shown in the proof of Theorem 13.7, it suffices to prove $A$ is NP-hard. Since Independent Set is NP-complete (Theorem 2.16), it suffices to show Independent Set $\leq_p^m A$. Map $(G,k)$ to an input to $A$ as follows. Assume $G$ has $\ell$ nodes, numbered $1,\ldots,\ell$, and that $2 \leq k \leq \ell$. Define $n = \ell + k$ in the input to $A$, so an example $x$ has bits $x_1 \cdots x_\ell$ indexed by $G$'s nodes and extra bits $x_{\ell+1} \cdots x_{\ell+k}$. The training data has two parts. The "enforce $t = k$" part is:

*(i)* We have a positive example $0^\ell 1^k$ (all 1s for the extra bits).
*(ii)* For each $i \in [k]$, we have a negative example $0^\ell 1^{i-1} 0 1^{k-i}$ (all 1s for the extra bits, except a 0 for the $i^{\text{th}}$ extra bit).

The "graph" part is:

*(iii)* We have a positive example $1^\ell 0^k$.
*(iv)* For each edge $\{v,w\}$ in $G$, we have a negative example $x$ where $x_v = x_w = 1$ and $x_{\ell+1} \cdots x_{\ell+k-2} = 1^{k-2}$ and all other bits of $x$ are 0s.

To show this poly-time mapping reduction is correct, we argue that $G$ has an independent set of size $\geq k$ iff there exists $c_{I,t} \in C$ consistent with the training data:

$\Rightarrow$: Assume $G$ has an independent set $S \subseteq [\ell]$ of size $\geq k$. Let $I = S \cup \{\ell+1,\ldots,\ell+k\} \subseteq [n]$. Then $c_{I,k}$ is consistent with the training data because:

*(i)* $c_{I,k}(0^\ell 1^k) = 1$ since $\{\ell+1,\ldots,\ell+k\} \subseteq I$.
*(ii)* For each $i \in [k]$, $c_{I,k}(0^\ell 1^{i-1} 0 1^{k-i}) = 0$ since $0^\ell 1^{i-1} 0 1^{k-i}$ has fewer than $k$ many 1s.
*(iii)* $c_{I,k}(1^\ell 0^k) = 1$ since $\sum_{i \in S} 1 = |S| \geq k$.
*(iv)* For each edge $\{v,w\}$ in $G$, the associated negative example $x$ has $c_{I,k}(x) = 0$ since $S$ is independent and therefore contains at most one of $v,w$, and $I$ also contains the $k-2$ indices $\ell+1,\ldots,\ell+k-2$, for a total of $\leq k-1$ many 1s among the bits of $x$ indexed by $I$.

$\Leftarrow$: Assume $c_{I,t}$ is consistent with the training data. First, we claim that $t = k$ and $\{\ell+1,\ldots,\ell+k\} \subseteq I$. For each $i \in \{\ell+1,\ldots,\ell+k\}$, by *(i)* and *(ii)* we have

$$\left|I \cap \{\ell+1,\ldots,\ell+k\}\right| \;\geq\; t \;>\; \left|I \cap \big(\{\ell+1,\ldots,\ell+k\} \setminus \{i\}\big)\right|$$

which implies $i \in I$. Thus $\{\ell+1,\ldots,\ell+k\} \subseteq I$ and

$$k \;=\; \left|I \cap \{\ell+1,\ldots,\ell+k\}\right| \;\geq\; t \;>\; \left|I \cap \{\ell+1,\ldots,\ell+k-1\}\right| \;=\; k-1$$

so $t = k$. This proves the claim. Then $S = I \cap [\ell]$ is an independent set of size $\geq k$ because:

*(iii)* Since $c_{I,k}(1^\ell 0^k) = 1$, we have $|S| = \sum_{i \in S} 1 \geq k$.
*(iv)* For each edge $\{v,w\}$ in $G$, since the associated negative example $x$ has $c_{I,k}(x) = 0$, we have $\left|I \cap \{v,w,\ell+1,\ldots,\ell+k-2\}\right| < k$. Since $\left|I \cap \{\ell+1,\ldots,\ell+k-2\}\right| = k-2$, we have $\left|I \cap \{v,w\}\right| < 2$ and thus $S$ contains at most one of $v,w$.

**Exercise 13.13:**

$$\left|C_{\{00\}}\right| = 2 \qquad \left|C_{\{01\}}\right| = 2 \qquad \left|C_{\{10\}}\right| = 2 \qquad \left|C_{\{11\}}\right| = 2$$

$$\left|C_{\{00,01\}}\right| = 3 \quad \left|C_{\{00,10\}}\right| = 3 \quad \left|C_{\{00,11\}}\right| = 3 \quad \left|C_{\{01,10\}}\right| = 3 \quad \left|C_{\{01,11\}}\right| = 3 \quad \left|C_{\{10,11\}}\right| = 3$$

$$\left|C_{\{00,01,10\}}\right| = 4 \qquad \left|C_{\{00,01,11\}}\right| = 4 \qquad \left|C_{\{00,10,11\}}\right| = 4 \qquad \left|C_{\{01,10,11\}}\right| = 4$$

$$\left|C_{\{00,01,10,11\}}\right| = 5$$

$$g_C(0) = 1 \qquad g_C(1) = 2 \qquad g_C(2) = 3 \qquad g_C(3) = 4 \qquad g_C(4) = 5$$

$C$'s VC dimension is 1.

**Exercise 13.14:** The class of interval functions $\left\{c_{\ell,u} : \ell, u \in \{0,1\}^n\right\}$ has VC dimension 2: The VC dimension is $\geq 2$ because some set $S$ of size 2 (in fact, every such set) is shattered: Say $S = \{x, y\}$ where $x < y$.

$$\begin{array}{llll} c_{y,x}(x) = 0 & c_{y,y}(x) = 0 & c_{x,x}(x) = 1 & c_{x,y}(x) = 1 \\ c_{y,x}(y) = 0 & c_{y,y}(y) = 1 & c_{x,x}(y) = 0 & c_{x,y}(y) = 1 \end{array}$$

The VC dimension is $\leq 2$ because no set $S$ of size 3 is shattered: Say $S = \{x, y, z\}$ where $x < y < z$. Then no interval function $c_{\ell,u}$ agrees with the function $f : S \to \{0,1\}$ defined by $f(x) = f(z) = 1$ and $f(y) = 0$, because $c_{\ell,u}(x) = 1$ implies $\ell \leq x$, and $c_{\ell,u}(z) = 1$ implies $z \leq u$, in which case $\ell \leq x < y < z \leq u$ and therefore $c_{\ell,u}(y) = 1 \neq f(y)$.

**Exercise 13.15:** The class $C$ of literals on $n$ variables has VC dimension $\lfloor \log |C| \rfloor = \lfloor \log 2n \rfloor = \lfloor \log n \rfloor + 1$: The VC dimension is automatically $\leq \lfloor \log |C| \rfloor$. Now, we show the VC dimension is $\geq \lfloor \log n \rfloor + 1$. There exists a $\lfloor \log n \rfloor \times n$ binary matrix with all $2^{\lfloor \log n \rfloor} \leq n$ possible columns. This matrix's rows together with $0^n$ are a shattered set $S$ of size $\lfloor \log n \rfloor + 1$: For every $f : S \to \{0,1\}$, if $f(0^n) = 0$ then $f$ agrees with some positive literal, and if $f(0^n) = 1$ then $f$ agrees with some negative literal.

**Exercise 13.16:** For the learning reduction $(F_{\text{ex}}, F_{\text{con}}, F_{\text{hyp}})$, denote $x' = F_{\text{ex}}(x)$ and $c' = F_{\text{con}}(c)$ for any example $x$ and concept $c$.

Suppose $C$ has VC dimension $d$. We claim that $C'$ has VC dimension $\geq d$. Let $S$ be a size-$d$ set shattered by $C$. We claim that $S' = \{x' : x \in S\}$ has size $d$ and is shattered by $C'$. To show that $S'$ has size $d$, we observe that for all $x^1 \neq x^2$ in $S$, we have $x^{1\prime} \neq x^{2\prime}$ because for any $c \in C$ with $c(x^1) \neq c(x^2)$ (which exists since $C$ shatters $S$), we have $c'(x^{1\prime}) = c(x^1) \neq c(x^2) = c'(x^{2\prime})$. To show that $C'$ shatters $S'$, consider any $f' : S' \to \{0,1\}$. Define $f : S \to \{0,1\}$ by $f(x) = f'(x')$. Since $C$ shatters $S$, there exists $c \in C$ such that $c(x) = f(x)$ for all $x \in S$. Thus $c'(x') = c(x) = f(x) = f'(x')$ for all $x' \in S'$.

**Exercise 13.17:** Suppose for contradiction there exist $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that $w \cdot x \geq \theta$ iff $\oplus(x) = 1$.

- $\oplus(000\cdots0) = 0$ implies $0 < \theta$.
- $\oplus(100\cdots0) = 1$ implies $w_1 \geq \theta$.
- $\oplus(010\cdots0) = 1$ implies $w_2 \geq \theta$.
- $\oplus(110\cdots0) = 0$ implies $w_1 + w_2 < \theta$.

This yields the contradiction $\theta < \theta + \theta \leq w_1 + w_2 < \theta$.

**Exercise 13.18.a:** Consider the size-$(n+1)$ set $S = \{x^0, x^1, \ldots, x^n\} \subseteq \{0,1\}^n$ where $x^0$ is the all-0 string and for each $i \in [n]$, $x^i$ has a 1 at index $i$ and 0s elsewhere. To see that the class of decision lists shatters $S$, consider any $f : S \to \{0,1\}$:

- If $f(x^0) = 0$ then $f$ agrees with the clause $\bigvee_{i : f(x^i)=1} x_i$ (which is equivalent to a decision list).
- If $f(x^0) = 1$ then $f$ agrees with the term $\bigwedge_{i : f(x^i)=0} \overline{x_i}$ (which is equivalent to a decision list).

**Exercise 13.18.b:** Consider any $n$-variable decision list $c$ with $k \geq 1$ internal nodes. We design an equivalent halfspace "$w \cdot x \geq \theta$". The idea is that each variable $x_j$ in $c$ can have a coefficient $w_j$ large enough (in absolute value) to override all variables that appear later in $c$. Assume the terminal leaf outputs 1 (by swapping it with its sibling if necessary).

- Initially, let $\theta = 0$.
- If the $i^{\text{th}}$ internal node is $(j, 1, 1)$ ("if $x_j = 1$ then output 1"), let $w_j = 2^{k-i}$.
- If the $i^{\text{th}}$ internal node is $(j, 0, 1)$ ("if $x_j = 0$ then output 1"), let $w_j = -2^{k-i}$ and subtract $2^{k-i}$ from $\theta$. This is equivalent to including $2^{k-i}(1 - x_j)$ on the left side and not changing $\theta$.
- If the $i^{\text{th}}$ internal node is $(j, 1, 0)$ ("if $x_j = 1$ then output 0"), let $w_j = -2^{k-i}$.
- If the $i^{\text{th}}$ internal node is $(j, 0, 0)$ ("if $x_j = 0$ then output 0"), let $w_j = 2^{k-i}$ and add $2^{k-i}$ to $\theta$. This is equivalent to including $-2^{k-i}(1 - x_j)$ on the left side and not changing $\theta$.
- Let $w_j = 0$ for all other $j$.

Consider the perspective where the right side is $\theta = 0$ and the left side consists of coefficients ($2^{k-i}$ or $-2^{k-i}$) times literals ($x_j$ or $1 - x_j$). We claim that for every assignment to $x$, this halfspace outputs $c(x)$. If the computation of $c(x)$ reaches the terminal leaf (which outputs 1), then all literals on the left side evaluate to 0, so the halfspace outputs 1 since $0 \geq 0$. Otherwise, say the $i^{\text{th}}$ internal node $(j, a, b)$ is where the computation branches to the nonterminal leaf child and outputs $b = c(x)$ because $x_j = a$. The corresponding literal evaluates to 1 and has coefficient $2^{k-i}$ if $b = 1$ or $-2^{k-i}$ if $b = 0$. All earlier literals evaluate to 0, and the sum of absolute values of coefficients of later literals is $\leq 2^{k-(i+1)} + \cdots + 2^0 = 2^{k-i} - 1$ (Lemma 0.1). Thus if $c(x) = 1$ then the halfspace outputs 1 since the left side is $\geq 2^{k-i} - (2^{k-i} - 1) = 1 \geq 0$, and if $c(x) = 0$ then the halfspace outputs 0 since the left side is $\leq -2^{k-i} + (2^{k-i} - 1) = -1 < 0$.

Every size-$(n + 2)$ set is not shattered by halfspaces (Lemma 13.14) and thus not shattered by decision lists either.

**Exercise 13.19:** Let $C$ contain all $c$ such that $c(x) = 1$ for at most $d$ many $x \in \{0,1\}^n$ ($c$'s truth table has weight $\leq d$). Observe that $C$'s VC dimension is $\geq d$, and is $\leq d$ since for every $S \subseteq \{0,1\}^n$ with $|S| > d$, there's no $c \in C$ such that $c_S$ is the constant 1 function. For every $S$ with $|S| = m > d$, we have $|C_S| = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$ since $C_S$ contains all functions $f : S \to \{0,1\}$ of weight $\leq d$, and there are $\binom{m}{k}$ many weight-$k$ functions in $C_S$ for each $k \in \{0,1,\ldots,d\}$. Thus $g_C(m) = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$ for $m > d$.

**Exercise 13.20:**

| $x$ | $C_T$ | | |
|---|---|---|---|
| 00 | 0 | 1 | 1 |
| 01 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 |

| $x$ | $C_T^*$ | |
|---|---|---|
| 00 | 0 | 1 |
| 01 | 1 | 0 |
| 10 | 1 | 0 |

find-shattered($C$) returns:

$$\{\emptyset, \{00\}, \{10\}\} \cup \{\emptyset \cup \{11\}, \{00\} \cup \{11\}\} = \{\emptyset, \{00\}, \{10\}, \{11\}, \{00, 11\}\}$$

**Exercise 13.21:** Consider any $S \subseteq \{0,1\}^n$ of size $m = 2dt\log(3t)$. To show that $C'$ doesn't shatter $S$, we show $|C'_S| < 2^m$. For each $i \in [t]$, there are $|C_S| \leq g_C(m) \leq (\mathrm{e}m/d)^d$ possibilities of $c^i_S$ with $c^i \in C$ (Lemma 13.15). Thus there are $\leq (\mathrm{e}m/d)^{dt}$ possibilities of the tuple $(c^1_S, \ldots, c^t_S)$. Thus there are $|C'_S| \leq (\mathrm{e}m/d)^{dt}$ possibilities of $c'_S$ with $c' \in C'$. To conclude that $|C'_S| < 2^m$, by taking the log of both sides, we need to show $m$ is large enough that $dt\log(\mathrm{e}m/d) < m$. Plugging in the definition of $m$, we indeed have $dt\log(\mathrm{e}2t\log(3t)) < 2dt\log(3t)$.

**Exercise 13.22:** Assume $C$ has VC dimension $d \geq 2$ and $\varepsilon < 0.1$. Consider any $L : \big(\{0,1\}^n \times \{0,1\}\big)^m \to H$ with $m < 0.01d/\varepsilon$. We claim that $L$ isn't a $(0.01, \varepsilon)$-PAC learner for $(C, H)$. That is, there exist a concept $c \in C$ and a distribution $D$ over $\{0,1\}^n$ such that:

$$\mathbf{Pr}_{(x^1,\ldots,x^m)\sim D^m}\left[\begin{array}{l}\mathbf{Pr}_{y\sim D}\big[h(y) = c(y)\big] \geq 1-\varepsilon \\ \text{where } h = L\big((x^1, c(x^1)), \ldots, (x^m, c(x^m))\big)\end{array}\right] < 0.99$$

Let $S \subseteq \{0,1\}^n$ be a size-$d$ set that $C$ shatters, and let $z \in S$ be an arbitrary distinguished element. Define $D$ by $D(z) = 1 - 10\varepsilon$ and $D(y) = 10\varepsilon/(d-1)$ for all $y \in S \setminus \{z\}$. We use the probabilistic method to show the existence of $c$. Since $C_S$ is the set of all functions $f : S \to \{0,1\}$, we can sample a uniformly random $f : S \to \{0,1\}$ and let $c \in C$ be such that $c_S = f$. In other words, we toss a fair coin $d$ times to specify $c(x)$ for all $x \in S$, and then we define $c(x)$ for all $x \notin S$ in such a way that $c \in C$.

For $(x^1, \ldots, x^m) \sim D^m$, let the random variable $X$ be the number of $i \in [m]$ such that $x^i \neq z$. Then $X$ is a binomial random variable that counts the number of heads in $m$ tosses of a coin with heads probability $10\varepsilon$, where heads means $x^i \neq z$. Thus $\mathbf{E}[X] = 10\varepsilon m < d/8 \leq (d-1)/4$. Say $(x^1, \ldots, x^m)$ is *good* if $X \leq (d-1)/2$ and *bad* otherwise. By Lemma 7.13:

$$\mathbf{Pr}\big[(x^1, \ldots, x^m) \text{ is bad}\big] \;=\; \mathbf{Pr}\big[X > (d-1)/2\big] \;\leq\; \mathbf{E}[X]/((d-1)/2) \;\leq\; 1/2$$

Let $D'$ be the uniform distribution over $S \setminus \{z\}$. For random $c$ and $(x^1, \ldots, x^m) \sim D^m$ and $y \sim D'$ (we return to $y \sim D$ later):

$$\begin{aligned}
& \mathbf{Pr}_{c,x^1,\ldots,x^m,y}\big[h(y) \neq c(y)\big] \\
\geq\; & \mathbf{Pr}_{c,x^1,\ldots,x^m,y}\big[(x^1, \ldots, x^m) \text{ is good and } y \notin \{x^1, \ldots, x^m\} \text{ and } h(y) \neq c(y)\big] \\
=\; & \mathbf{Pr}_{x^1,\ldots,x^m}\big[(x^1, \ldots, x^m) \text{ is good}\big] \cdot \\
& \mathbf{Pr}_{x^1,\ldots,x^m,y}\big[y \notin \{x^1, \ldots, x^m\} \,\big|\, (x^1, \ldots, x^m) \text{ is good}\big] \cdot \\
& \mathbf{Pr}_{c,x^1,\ldots,x^m,y}\big[h(y) \neq c(y) \,\big|\, (x^1, \ldots, x^m) \text{ is good and } y \notin \{x^1, \ldots, x^m\}\big] \qquad \text{(chain rule)} \\
\geq\; & (1/2) \cdot (1/2) \cdot (1/2) \\
=\; & 1/8
\end{aligned}$$

since for any $(x^1, \ldots, x^m)$, $h(y)$ depends only on $c(x^1), \ldots, c(x^m)$ and so is independent of $c(y)$ if $y \in S \setminus \{x^1, \ldots, x^m\}$. Therefore:

$$\min_c\Big(\mathbf{E}_{x^1,\ldots,x^m}\big[\mathbf{Pr}_y\big[h(y) = c(y)\big]\big]\Big) \;\leq\; \mathbf{Pr}_{c,x^1,\ldots,x^m,y}\big[h(y) = c(y)\big] \;\leq\; 7/8$$

Thus there exists a concept $c \in C$ such that:

$$\mathbf{E}_{x^1,\ldots,x^m}\big[\mathbf{Pr}_y\big[h(y) = c(y)\big]\big] \;\leq\; 7/8$$

By Lemma 7.13:

$$\mathbf{Pr}_{x^1,\ldots,x^m}\big[\mathbf{Pr}_y\big[h(y) = c(y)\big] \geq 9/10\big] \;\leq\; (7/8)/(9/10) \;<\; 0.99$$

For every $h$, we have

$$\mathbf{Pr}_{y\sim D}\left[h(y)=c(y)\right] \leq (1-10\varepsilon)+10\varepsilon\cdot\mathbf{Pr}_{y\sim D'}\left[h(y)=c(y)\right]$$

and thus if $\mathbf{Pr}_{y\sim D}\left[h(y)=c(y)\right]\geq 1-\varepsilon$ then $\mathbf{Pr}_{y\sim D'}\left[h(y)=c(y)\right]\geq 9/10$. Therefore:

$$\mathbf{Pr}_{x^1,\ldots,x^m}\left[\mathbf{Pr}_{y\sim D}\left[h(y)=c(y)\right]\geq 1-\varepsilon\right] \leq \mathbf{Pr}_{x^1,\ldots,x^m}\left[\mathbf{Pr}_{y\sim D'}\left[h(y)=c(y)\right]\geq 9/10\right] < 0.99$$

**Exercise 13.23:** As in the proof of Theorem 13.23, assume that $\varepsilon_i \leq \varepsilon$ for all $i$, and that $L'$ outputs $h'$ at the end. For each $j \in [m]$ with $h'(x^j) \neq c(x^j)$, we showed that:

$$1 \leq e^{-\sum_{i=1}^{t} w_i h^i(x^j) c(x^j)} = z_1 \cdots z_t m J_{t+1}(j) < \left(e^{-2\gamma^2}\right)^t m J_{t+1}(j) \leq \beta m J_{t+1}(j)$$

It follows that the number of $j \in [m]$ with $h'(x^j) \neq c(x^j)$ is

$$\leq \sum_{j=1}^{m} \beta m J_{t+1}(j) = \beta m \sum_{j=1}^{m} J_{t+1}(j) = \beta m$$

since $J_{t+1}$ is a distribution. Thus $h'$ agrees with $c$ on $\geq 1 - \beta$ fraction of the training examples $x^1, \ldots, x^m$.

**Exercise 13.24:** The learner maintains the set $B$ (initially $C$) of remaining concepts. In each iteration, the learner picks an arbitrary remaining hypothesis $h \in B$ (it doesn't matter which), receives a counterexample $a$ (if $h$ isn't a success), and removes from $B$ every concept $b$ with $b(a) = h(a)$. This maintains the invariant that for each prior counterexample, all concepts in $B$ have the same value. Thus no prior counterexample can be reused, since otherwise it would make $B$ empty. After $2^n$ mistakes, all $2^n$ strings $a \in \{0,1\}^n$ have been counterexamples, so $B$ has only one remaining concept (since by the invariant, all concepts in $B$ agree with each other on all strings) and success is ensured.

**Exercise 13.25:** Consider any concept $c \in C$ and distribution $D$ over $\{0,1\}^n$. Say a hypothesis $h \in H$ is *bad* iff $\mathbf{Pr}_{y \sim D}\big[h(y) = c(y)\big] < 1-\varepsilon$. Let $h^1, h^2, \ldots$ denote $L$'s sequence of hypotheses. If $L'$ finds $k$ counterexamples for $L$, then $L'$ outputs $h^{k+1}$, which is equivalent to $c$ and therefore good. Thus if $L'$ outputs a bad hypothesis, it must be one of $h^1, \ldots, h^k$, for which no counterexample was found. Let the event "$h^i$ is bad" mean "$h^i$ exists (since $L'$ doesn't halt earlier) and is bad."

$$
\begin{aligned}
& \mathbf{Pr}_{x^{i,j} \sim D \text{ for all } i,j}\big[L' \text{ outputs a bad hypothesis}\big] \\
\leq\ & \textstyle\sum_{i=1}^{k} \mathbf{Pr}\big[h^i \text{ is bad and } h^i(x^{i,j}) = c(x^{i,j}) \text{ for all } j \in [\ell]\big] \\
\leq\ & \textstyle\sum_{i=1}^{k} \mathbf{Pr}\big[h^i(x^{i,j}) = c(x^{i,j}) \text{ for all } j \in [\ell] \,\big|\, h^i \text{ is bad}\big] \\
=\ & \textstyle\sum_{i=1}^{k} \sum_{\text{bad } h} \mathbf{Pr}\big[h^i = h \,\big|\, h^i \text{ is bad}\big] \cdot \mathbf{Pr}\big[h(x^{i,j}) = c(x^{i,j}) \text{ for all } j \in [\ell]\big] \\
=\ & \textstyle\sum_{i=1}^{k} \sum_{\text{bad } h} \mathbf{Pr}\big[h^i = h \,\big|\, h^i \text{ is bad}\big] \cdot \prod_{j=1}^{\ell} \mathbf{Pr}\big[h(x^{i,j}) = c(x^{i,j})\big] \\
\leq\ & \textstyle\sum_{i=1}^{k} \sum_{\text{bad } h} \mathbf{Pr}\big[h^i = h \,\big|\, h^i \text{ is bad}\big] \cdot \prod_{j=1}^{\ell} (1-\varepsilon) \\
=\ & k(1-\varepsilon)^{\ell} \\
\leq\ & k\mathrm{e}^{-\varepsilon\ell} \\
\leq\ & \delta
\end{aligned}
$$

**Exercise 13.26.a:** The mistake complexity is $\leq 2^n$ (Exercise 13.24). To show it's $> 2^n - 2$, we exhibit a teacher strategy that makes the game last for $> 2^n - 2$ rounds. When the learner picks a hypothesis $h$, the teacher picks any $a$ such that $h(a) = 1$ (it doesn't matter which). The only width-$n$ term (point function) $b$ this eliminates is the one with $b(a) = 1$. After round $2^n - 2$, at least two of the $2^n$ many width-$n$ terms remain, so the game is not over.

**Exercise 13.26.b:** The mistake complexity is $\geq n$ by Lemma 13.27 since the VC dimension of $C$ is $n$ (Lemma 13.13). The mistake complexity is $\leq n+1$ by a learner similar to the proper one for monotone terms (§13.6.2):

do an equivalence query with the constant 0 function
given counterexample $a \in \{0,1\}^n$: initialize $h$ to the unique width-$n$ term such that $h(a) = 1$
repeat:
    do an equivalence query with hypothesis $h$
    if success: output hypothesis $h$
    else given counterexample $a \in \{0,1\}^n$: remove from $h$ every literal not satisfied by $a$

We just need one improper query at the beginning to find a term that contains all literals of the target concept. The rest of the proof of correctness is completely analogous to the corresponding proof for monotone terms (§13.6.2).

**Exercise 13.27:** Let $c$ be the target decision list. The first item of $c$ never produces erroneous output, so it never moves beyond level 1. Among the inputs that survive past $c$'s first item, $c$'s second item never produces erroneous output, so $c$'s second item never moves beyond level 2. And so on. Since $c$ has $O(n)$ items, its terminal item never moves beyond $O(n)$ levels. Thus the learner's hypothesis never has more than $O(n)$ levels (since items beyond a terminal item are not reached and thus don't produce output—erroneous or otherwise). Since each mistake results in an item moving to a subsequent level, and there are $O(n)$ items and $O(n)$ levels, there must only be $O(n^2)$ mistakes.

**Exercise 13.28:** Suppose $(F_{ex}, F_{con}, F_{hyp})$ is a learning reduction from $(C, H)$ to $(C', H')$ and $L'$ is a $k$-mistake learner for $(C', H')$. We claim that this $L$ is a $k$-mistake learner for $(C, H)$:

```
run L′:
    when it does an equivalence query with h′ ∈ H′:
        do an equivalence query with h = F_hyp(h′) ∈ H
        if success: output h
        else given counterexample a ∈ {0,1}^n:
            give L′ the counterexample a′ = F_ex(a) ∈ {0,1}^n′
```

Consider any $c \in C$, and let $c' = F_{con}(c) \in C'$. When $L$ gets a counterexample $a$, it gives $L'$ a valid counterexample $a'$ since $h'(a') = h(a) \neq c(a) = c'(a')$. Since $L'$ gets as many counterexamples as $L$ does, $L$ must get $\leq k$ counterexamples. Thus $L$ makes $\leq k$ mistakes before success.

**Exercise 13.29.a:** The game maintains a set $B$ (initially $C$) of remaining possibilities for the target concept. In each round, the learner chooses which one of the following happens:

- Membership query: The learner picks $a \in \{0,1\}^n$, and then the teacher picks $z \in \{0,1\}$ and removes from $B$ every concept $b$ with $b(a) = \overline{z}$.
- Equivalence query: The learner picks $h \in H$, and then the teacher picks $a \in \{0,1\}^n$ and removes from $B$ every concept $b$ with $b(a) = h(a)$.

The teacher is not allowed to remove all of $B$. The game ends when only one concept remains in $B$.

**Exercise 13.29.b:** To show the complexity is $> 2^n - 2$, we exhibit a teacher strategy that makes the game last for $> 2^n - 2$ rounds.

- Membership query: When the learner picks $a \in \{0,1\}^n$, the teacher disregards $a$ and picks $z = 0$. This eliminates only the point function $b$ such that $b(x) = 1$ iff $x = a$.
- Equivalence query: When the learner picks $h \in H$, the teacher picks the unique $a$ such that $h(a) = 1$. This eliminates only $h$ itself, because $b(a) = 0$ for all other concepts $b$.

After round $2^n - 2$, at least two of the $2^n$ concepts remain, so the game is not over.

**Exercise 13.29.c:** To show the complexity is $> \lfloor \log n \rfloor - 1$, we exhibit a teacher strategy that makes the game last for $> \lfloor \log n \rfloor - 1$ rounds.

- Membership query: When the learner picks $a \in \{0,1\}^n$, the teacher picks $z =$ the majority value among $b(a)$ over all remaining $b \in B$. This eliminates $\leq |B|/2$ concepts from $B$.
- Equivalence query: When the learner picks $h \in H$, the teacher disregards $h$ and picks an $a$ such that $b(a) = 1$ for exactly $\lceil |B|/2 \rceil$ of the remaining $b \in B$. (That is, $a_j = 1$ for exactly $\lceil |B|/2 \rceil$ of the indices $j$ corresponding to remaining dictatorships $x_j \in B$. The bits of $a$ on other indices don't matter.) If $h(a) = 1$, this eliminates $\lceil |B|/2 \rceil$ concepts from $B$. If $h(a) = 0$, this eliminates $\lfloor |B|/2 \rfloor$ concepts from $B$.

In all cases, $\leq \lceil |B|/2 \rceil$ concepts are removed from $B$, and thus $\geq \lfloor |B|/2 \rfloor$ concepts remain in $B$ in each round. This implies that the teacher maintains the invariant that $|B| \geq 2^{\lfloor \log n \rfloor - i}$ after round $i$: The invariant holds at the beginning ($i = 0$) since $|B| = n \geq 2^{\lfloor \log n \rfloor - 0}$. Round $i$ maintains the invariant because if $|B| \geq 2^{\lfloor \log n \rfloor - i + 1}$ before round $i$, then $\lfloor |B|/2 \rfloor \geq \left\lfloor 2^{\lfloor \log n \rfloor - i + 1}/2 \right\rfloor = 2^{\lfloor \log n \rfloor - i}$ and thus after round $i$ we have $|B| \geq 2^{\lfloor \log n \rfloor - i}$. After round $\lfloor \log n \rfloor - 1$, $B$ has $\geq 2^{\lfloor \log n \rfloor - (\lfloor \log n \rfloor - 1)} = 2$ remaining concepts, so the game is not over.

**Exercise 13.29.d:** To show the complexity is $> n-1$, we exhibit a teacher strategy that makes the game last for $> n-1$ rounds. The teacher maintains the invariant that after round $i$, the set $B$ of remaining concepts corresponds to $\geq 2^{n-i}$ many consecutive thresholds.

- Membership query: When the learner picks $a \in \{0,1\}^n$, the teacher picks $z =$ the majority value among $b(a)$ over all remaining $b \in B$. This eliminates $\leq |B|/2$ concepts from $B$.
- Equivalence query: When the learner picks $h \in H$, the teacher disregards $h$ and picks $a$ to be the $\lfloor |B|/2 \rfloor^{\text{th}}$ smallest threshold among the $|B|$ many remaining thresholds. If $h(a) = 0$, $B$'s upper $\lceil |B|/2 \rceil$ thresholds are eliminated. If $h(a) = 1$, $B$'s lower $\lfloor |B|/2 \rfloor$ thresholds are eliminated.

In all cases, $\leq \lceil |B|/2 \rceil$ concepts are removed from $B$. By the same analysis as in Exercise 13.29.c, this indeed maintains the invariant that $|B| \geq 2^{n-i}$ after round $i$. After round $n-1$, $B$ has $\geq 2^{n-(n-1)} = 2$ remaining concepts, so the game is not over.

**Exercise 13.30.a:** To answer an equivalence query for hypothesis $h$:

do a subset query with $h$
do a superset query with $h$
if either returned a counterexample $a$: return $a$
else: success ($h$ and $c$ are equivalent)

**Exercise 13.30.b:** This is like search-to-decision reductions for NP problems (Theorem 2.14). To answer a subset query for hypothesis $h \in H$:

```
do a weak subset query with h
if it returned yes: return yes
initialize a ← **···*  (partial assignment to h's variables)
for i ← 1,...,n:
    construct a size-(s + 10n) circuit h' such that h'(x) = 1 iff:
        h(x) = 1, and x is consistent with a, and x_i = 1
    do a weak subset query with h'
    if it returned no:  update a_i ← 1
    if it returned yes: update a_i ← 0
return a
```

Assuming $\{x : h(x) = 1\} \not\subseteq \{x : c(x) = 1\}$, this maintains the invariant that there exists $x$ consistent with $a$ such that $h(x) = 1$ and $c(x) = 0$. Thus at the end, $a$ is a full assignment and $h(a) = 1$ and $c(a) = 0$.

Each hypothesis $h' \in H'$ consists of $h$ and $\leq 10n$ gates and wires to check whether $x$ is consistent with $a$ and $x_i = 1$.

**Exercise 13.31:** MAJ's minimal 1-inputs are: $\{011, 101, 110\}$

$f$'s minimal 1-inputs are: $\{\text{MAJ's minimal 1-inputs}\}^2 = \{011011, 011101, 011110,$
$101011, 101101, 101110,$
$110011, 110101, 110110\}$

$f$'s smallest monotone DNF is: $(x_2 \wedge x_3 \wedge x_5 \wedge x_6) \vee (x_2 \wedge x_3 \wedge x_4 \wedge x_6) \vee (x_2 \wedge x_3 \wedge x_4 \wedge x_5) \vee$
$(x_1 \wedge x_3 \wedge x_5 \wedge x_6) \vee (x_1 \wedge x_3 \wedge x_4 \wedge x_6) \vee (x_1 \wedge x_3 \wedge x_4 \wedge x_5) \vee$
$(x_1 \wedge x_2 \wedge x_5 \wedge x_6) \vee (x_1 \wedge x_2 \wedge x_4 \wedge x_6) \vee (x_1 \wedge x_2 \wedge x_4 \wedge x_5)$

**Exercise 13.32:** Let $d = \binom{n}{\lfloor n/2 \rfloor}$.

First, we prove the VC dimension is $\geq d$. The size-$d$ set $S \subseteq \{0,1\}^n$ of all weight-$\lfloor n/2 \rfloor$ strings is shattered because every $f : S \to \{0,1\}$ agrees with the monotone function $c^f$ defined by $c^f(x) = 1$ iff $a \subseteq x$ for at least one $a \in S$ with $f(a) = 1$.

Now, we prove the VC dimension is $\leq d$. Suppose for contradiction there exists a size-$(d+1)$ shattered set $S = \{x^1, \ldots, x^{d+1}\}$ (unrelated to the previous paragraph's $S$). In particular, for each $i \in [d+1]$, there exists a monotone function $c^i$ such that $c^i(x^i) = 1$ and $c^i(x^j) = 0$ for all $j \neq i$. By Lemma 13.31.*(ii)*:

- Since $c^i(x^i) = 1$, $c^i$ has a minimal monotone term $T^i$ such that $T^i(x^i) = 1$.
- Since $c^i(x^j) = 0$ for all $j \neq i$, we have $T^i(x^j) = 0$.

Let $V^i \subseteq [n]$ be the set of indices of variables in $T^i$. By the theorem mentioned in the exercise, there exist distinct indices $i \neq i'$ such that $V^i \subseteq V^{i'}$. Since $T^{i'}(x^{i'}) = 1$, we also have $T^i(x^{i'}) = 1$, which is a contradiction.

**Exercise 13.33:** First, we note that at most $K$ terms have width $\leq k$: The number of width-$i$ terms is $\binom{n}{i}2^i$, and we have:

$$\begin{aligned}
(k/n)^k \textstyle\sum_{i=0}^{k}\binom{n}{i}2^i &\leq \textstyle\sum_{i=0}^{k}\binom{n}{i}2^i(k/n)^i && (k/n \leq 1)\\
&\leq \textstyle\sum_{i=0}^{n}\binom{n}{i}(2k/n)^i 1^{n-i}\\
&= (1+2k/n)^n && \text{(binomial formula)}\\
&\leq (e^{2k/n})^n && \text{(Fact 7.6)}\\
&= e^{2k}
\end{aligned}$$

Dividing both sides by $(k/n)^k$ shows that the number of terms of width $\leq k$ is $\sum_{i=0}^{k}\binom{n}{i}2^i \leq (e^2 n/k)^k \leq n^k$ since $k \geq 8 > e^2$.

We claim that with probability $\geq 1-\delta$ over the training examples,

$$\mathbf{Pr}_{y\sim U_n}\big[\psi(y)=0 \text{ and } \varphi(y)=1\big] \leq \varepsilon/2 \quad \text{and} \quad \mathbf{Pr}_{y\sim U_n}\big[\psi(y)=1 \text{ and } \varphi(y)=0\big] \leq \varepsilon/2$$

and thus $\mathbf{Pr}_{y\sim U_n}\big[\psi(y)\neq\varphi(y)\big] \leq \varepsilon$.

Say that terms of width $\leq k$ are *narrow*, and terms of width $> k$ are *wide*.

Note that $\psi$ contains each of $\varphi$'s narrow terms. Thus if $\psi(y)=0$ and $\varphi(y)=1$, then $T(y)=1$ for some wide term $T$ of $\varphi$. So with probability 1 over the training examples:

$$\begin{aligned}
&\mathbf{Pr}_{y\sim U_n}\big[\psi(y)=0 \text{ and } \varphi(y)=1\big]\\
\leq\ &\textstyle\sum_{\text{wide term } T \text{ of } \varphi} \mathbf{Pr}_{y\sim U_n}\big[T(y)=1\big] && \text{(union bound)}\\
\leq\ &\textstyle\sum_{\text{wide term } T \text{ of } \varphi} 2^{-(k+1)} && \text{(only place we use } D=U_n)\\
\leq\ &t\,2^{-\log(2t/\varepsilon)} && \text{(definition of } k)\\
=\ &\varepsilon/2
\end{aligned}$$

Say a narrow term $T$ is *good* iff $\mathbf{Pr}_{y\sim U_n}\big[T(y)=1 \text{ and } \varphi(y)=0\big] \leq \varepsilon/2K$. If $\psi$ contains no bad narrow terms, then:

$$\begin{aligned}
&\mathbf{Pr}_{y\sim U_n}\big[\psi(y)=1 \text{ and } \varphi(y)=0\big]\\
\leq\ &\textstyle\sum_{\text{term } T \text{ of } \psi} \mathbf{Pr}_{y\sim U_n}\big[T(y)=1 \text{ and } \varphi(y)=0\big] && \text{(union bound)}\\
\leq\ &\textstyle\sum_{\text{term } T \text{ of } \psi} \varepsilon/2K && \text{(each } T \text{ in } \psi \text{ is good)}\\
\leq\ &K\varepsilon/2K && \text{(each } T \text{ in } \psi \text{ is narrow)}\\
=\ &\varepsilon/2
\end{aligned}$$

For each bad narrow term $T$:

$$\begin{aligned}
&\mathbf{Pr}_{(x^1,\ldots,x^m)\sim U_n^m}\big[\psi \text{ contains } T\big]\\
=\ &\mathbf{Pr}_{(x^1,\ldots,x^m)\sim U_n^m}\big[\forall i:\ \neg\big(T(x^i)=1 \text{ and } \varphi(x^i)=0\big)\big]\\
=\ &\textstyle\prod_{i=1}^{m} \mathbf{Pr}_{x^i\sim U_n}\big[\neg\big(T(x^i)=1 \text{ and } \varphi(x^i)=0\big)\big] && (x^1,\ldots,x^m \text{ are independent})
\end{aligned}$$

$$\leq (1-\varepsilon/2K)^m \qquad (T \text{ is bad})$$
$$\leq \mathrm{e}^{-m\varepsilon/2K} \qquad (\text{Fact 7.6})$$
$$\leq \delta/K \qquad (\text{definition of } m)$$

Therefore:

$$\begin{aligned}
&\mathbf{Pr}_{(x^1,\ldots,x^m)\sim U_n^m}\Big[\mathbf{Pr}_{y\sim U_n}\big[\psi(y)=1 \text{ and } \varphi(y)=0\big] > \varepsilon/2\Big] \\
&\leq \mathbf{Pr}_{(x^1,\ldots,x^m)\sim U_n^m}\big[\psi \text{ contains a bad narrow term}\big] \\
&\leq \textstyle\sum_{\text{bad narrow term } T} \mathbf{Pr}_{(x^1,\ldots,x^m)\sim U_n^m}\big[\psi \text{ contains } T\big] \qquad (\text{union bound}) \\
&\leq \textstyle\sum_{\text{bad narrow term } T} \delta/K \\
&\leq K\delta/K \\
&= \delta
\end{aligned}$$

The learner has an outer loop over all narrow terms and an inner loop over all training examples. Thus it runs in time $\mathrm{poly}(Km) = \mathrm{poly}\big(n^{\log(t/\varepsilon)} \log \frac{1}{\delta}\big)$.

**Exercise 13.34.a:** The input size $N$ is essentially $2^{\ell+n}$.

$C$'s VC dimension is $\geq d$ iff there exists a size-$d$ set $S \subseteq \{0,1\}^n$ such that for all $f : S \to \{0,1\}$, there exists $a \in \{0,1\}^\ell$ such that for all $b \in S$, $c^a(b) = f(b)$.

Truth Table VC Dimension $\in$ NP by a verifier that accepts iff its witness is a size-$d$ set $S \subseteq \{0,1\}^n$ such that for all $f : S \to \{0,1\}$, there exists $a \in \{0,1\}^\ell$ such that for all $b \in S$, $c^a(b) = f(b)$. Brute force over all $2^d \leq 2^\ell \leq N$ many $f$, all $2^\ell \leq N$ many $a$, and all $d$ many $b$ takes poly$N$ time.

Truth Table VC Dimension $\in \mathsf{TIME}[N^{O(\log N)}]$ by running the NP verifier on each size-$d$ set $S \subseteq \{0,1\}^n$, of which there are $\binom{2^n}{d} \leq (2^n)^d \leq N^\ell \leq N^{\log N}$ many.

**Exercise 13.34.b:** SUCCINCT VC DIMENSION $\in \Sigma_3$P by a 3-witness verifier $V$ where $V(E, d; S; f; a)$ accepts iff $S \subseteq \{0,1\}^n$ has size $d$ and $f : S \to \{0,1\}$ and $a \in \{0,1\}^\ell$ are such that for all $b \in S$, $E(a,b) = f(b)$.

To prove SUCCINCT VC DIMENSION is $\Sigma_3$P-hard, we show ADJUSTED $\Sigma_3$SAT $\leq_p^m$ SUCCINCT VC DIMENSION by the poly-time mapping reduction from the hint. To show that the reduction is correct, we argue that:

$$\left(\exists x\ \forall y \neq 0^k\ \exists z :\ \varphi(x,y,z) = 1\right) \iff \left(C \text{ determined by } E \text{ has VC dimension} \geq d\right)$$

$\Rightarrow$: Suppose $\exists x\ \forall y \neq 0^k\ \exists z :\ \varphi(x,y,z) = 1$. Consider any particular assignment $x$ such that $\forall y \neq 0^k\ \exists z :\ \varphi(x,y,z) = 1$, and define $S = \left\{(x,i) : i \in [k]\right\} \subseteq \{0,1\}^k \times [k]$, which has size $k = d$. We claim that $C$ shatters $S$, and thus $C$ has VC dimension $\geq d$. Consider any $f : S \to \{0,1\}$. If $f(x,i) = 0$ for all $i$, then $f$ agrees with $c^{x,0^k,0^k}$ on $S$. Otherwise, define $y \in \{0,1\}^k \setminus \{0^k\}$ by $y_i = f(x,i)$ for all $i$. There exists $z$ such that $\varphi(x,y,z) = 1$. Then $f$ agrees with $c^{x,y,z}$ on $S$, since for all $i$:

- If $f(x,i) = 0$ then $c^{x,y,z}(x,i) = 0$ since $y_i = 0$.
- If $f(x,i) = 1$ then $c^{x,y,z}(x,i) = 1$ since $y_i = 1$ and $\varphi(x,y,z) = 1$.

$\Leftarrow$: Suppose $C$ has VC dimension $\geq d$. Let $S \subseteq \{0,1\}^k \times [k]$ be a size-$d$ set that $C$ shatters. First, we claim that $S = \left\{(x,i) : i \in [k]\right\}$ for some assignment $x$. Suppose not. Then there exist $(w,i) \in S$ and $(v,j) \in S$ with $w \neq v$. Consider any $f : S \to \{0,1\}$ with $f(w,i) = f(v,j) = 1$. Since $C$ shatters $S$, there exists $(x,y,z)$ such that $c^{x,y,z}(w,i) = f(w,i) = 1$ and thus $w = x$, and $c^{x,y,z}(v,j) = f(v,j) = 1$ and thus $v = x$, which contradicts $w \neq v$.

We showed that $S = \left\{(x,i) : i \in [k]\right\}$ for some assignment $x$. To show that $\forall y \neq 0^k\ \exists z :\ \varphi(x,y,z) = 1$, consider any $y \neq 0^k$. Define $f : S \to \{0,1\}$ by $f(x,i) = y_i$ for all $i$. Since $C$ shatters $S$, there exists $a$ such that $c^a(x,i) = f(x,i) = y_i$ for all $i$. We claim that $a$ must be $(x,y,z)$ for some $z$: If $a$'s first component were not $x$, then $c^a(x,i) = 0$ for all $i$ (whereas $y_i = 1$ for some $i$). If $a = (x,u,z)$ for some $u$ and $z$, then for some $i$ we have $y_i = 1$ and thus $c^{x,u,z}(x,i) = 1$ and thus $\varphi(x,u,z) = 1$, which implies that for all $i$, $c^{x,u,z}(x,i) = u_i$, and thus $u = y$.

We showed that for every $y \neq 0^k$, there exists $z$ such that $c^{x,y,z}(x,i) = y_i$ for all $i$. For some $i$, we have $y_i = 1$ and thus $c^{x,y,z}(x,i) = 1$ and thus $\varphi(x,y,z) = 1$. Hence $\exists x\ \forall y \neq 0^k\ \exists z :\ \varphi(x,y,z) = 1$.

**Exercise 14.1:** We have Node Cover Search $\leq_p^o$ Node Cover completely analogous to Independent Set Search $\leq_p^o$ Independent Set (Exercise 2.14.a) since node covers are complements of independent sets. Assuming $\mathsf{P} = \mathsf{NP}$, we have Node Cover $\in \mathsf{P}$ and thus Node Cover Search has a poly-time algorithm. To solve Min Node Cover in poly time on input $G$, we repeatedly run the algorithm for Node Cover Search on input $(G, k)$ for increasing values of $k$ until it outputs a node cover.

**Exercise 14.2:** Given such a set system $S = (S_1, \ldots, S_n)$ over $[m]$ as input:

initialize $I \leftarrow \emptyset$
for each element $r \in [m]$ (in arbitrary order):
    if $r \notin \bigcup_{i \in I} S_i$: add to $I$ every $i$ such that $r \in S_i$
output $I$

First, we prove that the algorithm outputs a set cover. Consider any element $r \in [m]$. In $r$'s iteration, if $r \in \bigcup_{i \in I} S_i$ then we still have $r \in \bigcup_{i \in I} S_i$ at the end (since indices are never removed from $I$), and otherwise every (in particular, some) $i$ such that $r \in S_i$ is added to $I$. Either way, the final $I$ covers $r$.

Now, we prove $ALG \leq b \cdot OPT$. Let $D \subseteq [m]$ be the set of elements for which the "if" condition in the loop is true. Note that $ALG = |I| \leq b \cdot |D|$ since each $r \in D$ contributes $\leq b$ many indices $i$ to $I$. We have $|D| \leq OPT$ because the sets $\{i : r \in S_i\}$ for all $r \in D$ are disjoint from each other, and every set cover must contain at least one index from each of these $|D|$ many sets of indices (in order to cover each $r \in D$).

**Exercise 14.3.a:** For an iteration with $|R| > OPT$, we have $|R'| \leq |R|(1-1/OPT)$ as in the proof of Theorem 14.2. For an iteration with $|R| \leq OPT$, we have $|R'| \leq |R|-1$ since at least one element is removed from $R$. After $\lceil \ln \frac{m}{OPT} \rceil \cdot OPT$ many iterations:

$$\begin{aligned} |R| &\leq m(1-1/OPT)^{\lceil \ln(m/OPT) \rceil \cdot OPT} \\ &\leq m(e^{-1/OPT})^{\lceil \ln(m/OPT) \rceil \cdot OPT} && \text{(Fact 7.6)} \\ &= me^{-\lceil \ln(m/OPT) \rceil} \\ &\leq me^{-\ln(m/OPT)} \\ &= OPT \end{aligned}$$

After $OPT$ more iterations, $|R| = 0$ and the algorithm would halt. At the end, $ALG = |I| \leq \lceil \ln \frac{m}{OPT} \rceil \cdot OPT + OPT = \left\lceil 1 + \ln \frac{m}{OPT} \right\rceil \cdot OPT$.

**Exercise 14.3.b:** By Exercise 14.3.a, $ALG \leq \left\lceil 1 + \ln \frac{m}{OPT} \right\rceil \cdot OPT$. We have $OPT \geq m/b$ since each set in $S$ covers $\leq b$ elements, and all $m$ elements must be covered. Therefore $ALG \leq \lceil 1 + \ln b \rceil \cdot OPT$.

**Exercise 14.4:** For any integer $n \geq 4$, let $m = 2^{n-1} - 2$ and:

$$\begin{aligned} S_1 &= \{\text{first two elements}\} \\ S_2 &= \{\text{next four elements}\} \\ S_3 &= \{\text{next eight elements}\} \\ &\vdots \\ S_{n-2} &= \{\text{last } 2^{n-2} \text{ elements}\} \\ \\ S_{n-1} &= \{\text{all odd elements}\} \\ S_n &= \{\text{all even elements}\} \end{aligned}$$

$OPT = 2$ since $S_{n-1} \cup S_n = [m]$. But $ALG = n-2$ since the greedy algorithm picks $S_{n-2}, S_{n-3}, \ldots, S_2, S_1$ in that order: After picking $S_{n-2}, \ldots, S_{i+1}$, the number of uncovered elements is $2+4+8+\cdots+2^i = 2^{i+1}-2$, and $S_i$ covers $2^i$ remaining elements, while the next best options ($S_{n-1}$ and $S_n$) each cover only $(2^{i+1}-2)/2 = 2^i - 1$ remaining elements. Thus $ALG/OPT = (n-2)/2 = \Omega(\log m)$.

**Exercise 14.5:** Given a set system $S = (S_1, \ldots, S_n)$ over $[m]$ and $b$ as input, the following algorithm maintains the invariant that $C = \bigcup_{i \in I} S_i$ is the set of elements covered so far:

initialize $I \leftarrow \emptyset$ and $C \leftarrow \emptyset$
repeat $b$ times:
    pick $i$ with the largest $|S_i \setminus C|$
    add $i$ to $I$, and add $S_i \setminus C$ to $C$
output $I$

$ALG = |C|$ at the end, and $OPT = \left|\bigcup_{j \in J} S_j\right|$ where $J \subseteq [n]$ with $|J| = b$ is an optimal solution.

We claim that for every iteration, $OPT - |C'| \leq \big(OPT - |C|\big)(1 - 1/b)$ where $C$ is from the beginning of the iteration, and $C'$ denotes the updated $C$ after the iteration. There exists $k \in J$ such that:

$$|S_k \setminus C| \;\geq\; \left|\bigcup_{j \in J} S_j \setminus C\right|/b \;\geq\; \big(OPT - |C|\big)/b$$

Whichever $i$ the algorithm picks is no worse than $k$, so $|C'| - |C| = |S_i \setminus C| \geq \big(OPT - |C|\big)/b$. Rearranging this proves the claim.

Hence, after $b$ iterations, we have

$$OPT - ALG \;\leq\; (OPT - 0)(1 - 1/b)^b \;\leq\; OPT(\mathrm{e}^{-1/b})^b \;=\; OPT/\mathrm{e}$$

because initially $|C| = 0$, and $OPT - |C|$ gets multiplied by $\leq 1 - 1/b$ in each iteration. Thus $ALG \geq (1 - 1/\mathrm{e})OPT$.

**Exercise 14.6.a:** Here's the learner:

$L\big((x^1, c(x^1)), \ldots, (x^m, c(x^m))\big)$:
    if $c(x^k) = 1$ for at least one $k$:
        run the learner from §13.2.2 to get a term $t$
        let $K = \big\{k \in [m] : c(x^k) = 0\big\}$
        define the set system $S = (S_1, S_2, \ldots)$ over $K$ where:
            $S_i = \big\{k \in K : t$'s $i^{\text{th}}$ literal evaluates to 0 on $x^k\big\}$
        run the greedy Min Set Cover approximation algorithm to get a set cover $I$ for $S$
        output the term $h$ consisting of $t$'s literals indexed by $I$
    else: output the constant 0 hypothesis

Consider any term $c$ of width $\leq w$ and any $(x^1, \ldots, x^m)$. If $c(x^k) = 0$ for all $k$, then the constant 0 hypothesis is consistent with the training data. Now, assume $c(x^k) = 1$ for at least one $k$. Recall that $t$ contains all of $c$'s literals (and possibly more). Thus $c$ consists of $\leq w$ many literals of $t$, indexed by some set $J$. For every $k \in K$, since $c(x^k) = 0$, there exists $j \in J$ such that $t$'s $j^{\text{th}}$ literal evaluates to 0 on $x^k$, and thus $k \in S_j$. This means $J$ is a set cover for $S$, so $OPT \leq |J| \leq w$. Since the greedy Min Set Cover algorithm achieves a $\lceil \ln |K| \rceil \leq \lceil \ln m \rceil$-approximation (Theorem 14.2), we have $ALG = |I| \leq w\lceil \ln m \rceil$, so $h$ is in the hypothesis class since $h$ is a term of width $\leq w\lceil \ln m \rceil$. We argue that $h$ is consistent with the training data:

- $h$ evaluates to 1 on all positive training examples since $t$ does (and $h$ is a subset of $t$'s literals).
- $h$ evaluates to 0 on all negative training examples because $I$ is a set cover for $S$: For every $k \in K$ there exists $i \in I$ such that $k \in S_i$, in other words, $t$'s $i^{\text{th}}$ literal evaluates to 0 on $x^k$, and therefore $h(x^k) = 0$ since $h$ contains this literal.

**Exercise 14.6.b:** Here's the learner:

$L\big((x^1, c(x^1)), \ldots, (x^m, c(x^m))\big)$:
    if $c(x^k) = 1$ for at least one $k$: run the learner from Exercise 14.6.a
    else:
        initialize $Z \leftarrow \{x^k : k \in [m]\}$ and $h \leftarrow$ empty term
        for $i \leftarrow 1, 2, \ldots$ until $Z = \emptyset$:
            add to $h$ either $x_i$ or $\overline{x_i}$, whichever evaluates to 0 for at least half of all $x \in Z$
            remove from $Z$ every $x$ on which the literal added to $h$ evaluates to 0
        output $h$

Assume $c(x^k) = 0$ for all $k$. The learner maintains the invariant that $Z = \{x^k : h(x^k) = 1\}$. Initially, $|Z| \leq m$. In each iteration, $|Z|$ decreases by at least a factor of 2. So after $\lfloor 1 + \log m \rfloor > \log m$ iterations, $|Z| < m/2^{\log m} = 1$ and thus $|Z| = 0$. The learner halts and outputs the term $h$ of width $\leq \lfloor 1 + \log m \rfloor$ such that $h(x^k) = 0$ for all $k$. This means $h$ is consistent with the training data. (The $i$ loop cannot run out of variables, because initially $|Z| < 2^n$ since the training data is consistent with some term $c$.)

To handle the possibility of $w = 1$, a poly-time learner can simply try all possible literals (and the constant 1 function) to find one consistent with the training data.

**Exercise 14.7.a:** $\varphi|_a = (\overline{x_3} \vee x_5) \wedge (x_3 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_4}) \wedge (x_4 \vee x_5)$ so:

$$\begin{aligned}
\mathbf{E}\big[\text{\# unsatisfied clauses in } \varphi|_a\big] &= \textstyle\sum_{\text{clause } C \text{ in } \varphi|_a} \mathbf{Pr}\big[C \text{ is unsatisfied}\big] \\
&= \textstyle\sum_{\text{clause } C \text{ in } \varphi|_a} 2^{-\text{width of } C} \\
&= 2^{-2} + 2^{-3} + 2^{-1} + 2^{-2} \\
&= 9/8
\end{aligned}$$

**Exercise 14.7.b:** No, because if a clause in $\varphi|_a$ is satisfied by $x_i = 1$, then there is no corresponding clause in $\varphi|_b$.

**Exercise 14.8.a:** Let $E_1, \ldots, E_m$ be linear equations over $\mathbb{Z}_2$ with nonempty left sides. Consider a uniformly random assignment. For each $i \in [m]$, let $Y_i$ be the indicator random variable for $E_i$ being satisfied. Then $\mathbf{E}[Y_i] = \mathbf{Pr}[E_i \text{ is satisfied}] = 1/2$ by the random subsum principle (Lemma 7.2) since $E_i$'s left side is nonempty. The number of satisfied equations is $Y = Y_1 + \cdots + Y_m$. By linearity of expectation, $\mathbf{E}[Y] = \mathbf{E}[Y_1] + \cdots + \mathbf{E}[Y_m] = m/2$. By Lemma 0.15, $Y(a) \geq m/2$ for some assignment $a$, so $a$ satisfies at least half of the equations.

**Exercise 14.8.b:** Consider any system $S$ of linear equations over $\mathbb{Z}_2$ with variables $x_1 \cdots x_n$, and assume without loss of generality that $S$'s equations all have nonempty left sides. Partition $S$ into $S_1, \ldots, S_n$ where $S_i$ is the equations in $S$ with $x_i$ as the highest-index variable on their left sides. Given $S$ as input:

```
for i ← 1, 2, ..., n:
    choose a_i ∈ {0, 1} such that a_1 ··· a_{i−1} a_i satisfies at least half of the equations in S_i
output the assignment a = a_1 ··· a_n
```

In iteration $i$, there indeed exists $a_i \in \{0,1\}$ such that $a_1 \cdots a_{i-1} a_i$ satisfies at least half of $S_i$: Regardless of $a_1 \cdots a_{i-1}$, each equation in $S_i$ is satisfied by exactly one value of $x_i$, because the left side is the sum (in $\mathbb{Z}_2$) of $x_i$ and some (possibly none) of $x_1 \cdots x_{i-1}$. Thus either $a_1 \cdots a_{i-1} 0$ satisfies at least half of the equations in $S_i$, or $a_1 \cdots a_{i-1} 1$ does.

Since $a$ satisfies at least half of $S_i$ for each $i$, and since $S_1, \ldots, S_n$ partitions $S$, we conclude that $a$ satisfies at least half of $S$.

**Exercise 14.9.a:** Let $G = (V, E)$ be an undirected graph with $m$ edges. Consider a uniformly random 3-color assignment $c : V \to [3]$. For each edge $e = \{u, v\} \in E$, let $X_e$ be the indicator random variable for $e$ being multicolored. Then $\mathbf{E}[X_e] = \mathbf{Pr}\big[c(u) \neq c(v)\big] = 2/3$ since of the nine equally likely 3-color assignments to $u$ and $v$, six of them multicolor $e$. The number of multicolored edges is $X = \sum_{e \in E} X_e$. By linearity of expectation, $\mathbf{E}[X] = \sum_{e \in E} \mathbf{E}[X_e] = (2/3)m$. By Lemma 0.15, there exists $c$ such that $X(c) \geq (2/3)m$, so $\geq 2/3$ fraction of $G$'s edges are multicolored.

**Exercise 14.9.b:** Given an undirected graph $G$ as input:

```
for each node v (in arbitrary order):
    assign v the least frequent color among v's neighbors that have already been colored
        (breaking ties arbitrarily)
```

For each edge $e = \{u, v\}$ in $G$, if $u$ gets colored before $v$, then we associate $e$ with $v$. Thus the edges are partitioned according to which node they're associated with. For each node $v$, at least $2/3$ fraction of the edges associated with $v$ are multicolored, because the algorithm assigns $v$'s color to ensure that. Thus at least $2/3$ fraction of $G$'s edges are multicolored.

**Exercise 14.10.a:** Assume the colors are $[d]$. On input $G = ([n], E)$ where every node has degree $\leq d-1$:

```
for v ← 1, 2, ..., n:
    assign c(v) ← an arbitrary color from [d] ∖ {c(u) : {u, v} ∈ E and u < v}
output c
```

This outputs a $d$-color assignment without getting stuck, because each node $v$ has $\leq d-1$ many neighbors and therefore $[d] \setminus \{c(u) : \{u, v\} \in E \text{ and } u < v\} \neq \emptyset$, so there's an option to color $v$ differently than its already-colored neighbors. By design, $c$ is proper: $u$ and $v$ have different colors if $\{u, v\} \in E$.

**Exercise 14.10.b:** Given a properly 3-colorable graph $G$ as input:

initialize $H \leftarrow G$ and define $d = \lceil \sqrt{n} \rceil$
while $H$ has at least one node $v$ of degree $\geq d$:
    assign $v$ color 1
    find a proper 2-coloring of $v$'s neighborhood using two fresh colors
    remove $v$ and $v$'s neighborhood (and their edges) from $H$
run the algorithm from Exercise 14.10.a to find a proper $d$-coloring of $H$ using $d$ fresh colors

In $H$, each node $v$'s neighborhood ($v$'s neighbor nodes and the edges between them) is properly 2-colorable since $G$ and therefore $H$ is properly 3-colorable, and $v$'s neighbors would all have different colors from $v$. Thus the algorithm can always find a proper 2-coloring of $v$'s neighborhood.

The final color assignment is proper (since no two nodes colored 1 are adjacent, and the other cases are straightforward). We argue that $O(\sqrt{n})$ colors are used. The "while" loop has $\leq n/(d+1) \leq \sqrt{n}$ iterations since each iteration removes $\geq d+1$ nodes. Thus the "while" loop uses $\leq 1 + 2\sqrt{n}$ colors. The rest of the algorithm uses $d \leq \sqrt{n} + 1$ colors. So the total number of colors is $\leq 2 + 3\sqrt{n}$.

**Exercise 14.11:** By definition, the algorithm outputs a feasible solution. But we claim it doesn't achieve a $1/o(m)$-approximation. For any $k$, consider an input with distinct terminal nodes $s_1, t_1, \ldots, s_k, t_k$, such that for each $i \in [k]$:

- There's a length-3 path $s_i \to y \to z \to t_i$ where $y$ and $z$ are two special nodes with the edge $y \to z$ shared among all $i$.
- There's a length-4 path from $s_i$ to $t_i$ using three fresh intermediate nodes that are unique to this $i$ (not shared with other $i$).

The number of edges is $m = 6k + 1$. We have $ALG = 1$ since the algorithm lets $P_i$ be the length-3 path for each $i$, and can only choose one of them since they all share the edge $y \to z$. We have $OPT = k$ by the length-4 paths. Thus the algorithm's approximation ratio is no better than $1/k = 6/(m-1)$.

**Exercise 14.12:** Consider an optimal solution $J \subseteq [k]$ with edge disjoint paths $Q_j$ from $s_j$ to $t_j$ for each $j \in J$. Thus $OPT = |J|$ and $ALG = |I|$ at the end. We claim that $|J \setminus I| \leq \sqrt{m|I|}$, which implies $|J| = |J \cap I| + |J \setminus I| \leq |I| + \sqrt{m|I|}$. For each $j \in J \setminus I$, $Q_j$ shares an edge with at least one $P_i$ such that $|P_i| \leq |Q_j|$, because if $Q_j$ were edge disjoint from all such $P_i$, then the algorithm would have chosen this $Q_j$ before choosing any longer $P_i$ or halting. Thus we can partition $J \setminus I$ into $|I|$ many sets $J_i$ indexed by $i \in I$, such that for each $j \in J_i$, we have $|P_i| \leq |Q_j|$ and $Q_j$ shares an edge with $P_i$. We have $|J_i| \leq |P_i|$ since the paths $Q_j$ for $j \in J_i$ are edge disjoint from each other and each shares an edge with $P_i$. Thus $|J_i| \leq |P_i| \leq \min_{j \in J_i} |Q_j| \leq \frac{1}{|J_i|} \sum_{j \in J_i} |Q_j|$, which implies $|J_i| \leq v_i$ where $v_i = \sqrt{\sum_{j \in J_i} |Q_j|}$. Considering a vector with components $v_i$ for $i \in I$:

$$\begin{aligned}
|J \setminus I| &= \textstyle\sum_{i \in I} |J_i| \\
&\leq \textstyle\sum_{i \in I} v_i \\
&\leq \sqrt{|I| \textstyle\sum_{i \in I} v_i^2} && \text{(Corollary 0.27)} \\
&= \sqrt{|I| \textstyle\sum_{i \in I} \sum_{j \in J_i} |Q_j|} \\
&= \sqrt{|I| \textstyle\sum_{j \in J \setminus I} |Q_j|} \\
&\leq \sqrt{|I| m}
\end{aligned}$$

**Exercise 14.13:** It suffices to show that for every constant $c' > 31/32$, $(2n'/3, c'2n'/3)$-Gap Max Shy Set (with $n'$ nodes) is NP-complete. By Corollary 14.10, it suffices to show that for every constant $c' > 31/32$ and $c = 4c' - 3 > 7/8$, there exists a poly-time $(n/3, cn/3, 2n'/3, c'2n'/3)$-gap-preserving reduction from Max Independent Set (with $n$ nodes) to Max Shy Set (with $n'$ nodes).

Map $G$ to $G'$ where $G'$ contains a copy of $G$, and each node $v$ of $G$ has a new neighbor $v'$ (which is only adjacent to $v$) in $G'$. If $G$ has $n$ nodes, then $G'$ has $n' = 2n$ nodes. In the solution to Exercise 2.20, we proved that $OPT' = OPT + n$ where $OPT$ is the maximum independent set size in $G$, and $OPT'$ is the maximum shy set size in $G'$. It follows that:

*(i)* $OPT \geq n/3 \Rightarrow OPT' \geq n/3 + n = 2n'/3$

*(ii)* $OPT < cn/3 \Rightarrow OPT' < cn/3 + n = c'2n'/3$

**Exercise 14.14:** Lemma 14.13: To show $m - OPT \leq m' - OPT'$, consider an assignment to $x, y$ falsifying $m' - OPT'$ clauses of $\varphi'$. This assignment, ignoring $y$, falsifies the same number of clauses of $\varphi$.

To show $m - OPT \geq m' - OPT'$, consider an assignment to $x$ falsifying $m - OPT$ clauses of $\varphi$. This assignment, together with either $y = 0$ or $y = 1$ (it doesn't matter which), falsifies the same number of clauses of $\varphi'$: For each satisfied clause of $\varphi$, both corresponding clauses of $\varphi'$ are satisfied. For each falsified clause of $\varphi$, one corresponding clause of $\varphi'$ is falsified.

Lemma 14.15: To show $m - OPT \leq m' - OPT'$, consider an assignment to $x, y$ falsifying $m' - OPT'$ clauses of $\varphi'$. This assignment, ignoring $y$, falsifies at most as many clauses of $\varphi$.

To show $m - OPT \geq m' - OPT'$, consider an assignment to $x$ falsifying $m - OPT$ clauses of $\varphi$. This assignment, together with some assignment to $y$, falsifies the same number of clauses of $\varphi'$: For each satisfied clause $C$ of $\varphi$, all corresponding clauses in $C'$ are satisfied. For each falsified clause $C$ of $\varphi$, there exists an assignment to $y_C$ such that exactly one clause of $C'$ is falsified, by the construction of $C'$ in Theorem 2.11 and Claim 2.13.

**Exercise 14.15.a:** First, we use the probabilistic method to prove that for every 3-NAE formula, there exists an assignment satisfying $\geq 3/4$ fraction of the NAE-clauses. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a 3-NAE formula. Consider a uniformly random assignment. For each $i \in [m]$, let $Y_i$ be the indicator random variable for $C_i$ being unsatisfied. Then $\mathbf{E}[Y_i] = \mathbf{Pr}[C_i \text{ is unsatisfied}] = 2/8$ since of the eight assignments to the three variables in $C_i$, two of them don't satisfy $C_i$. The number of unsatisfied NAE-clauses is $Y = Y_1 + \cdots + Y_m$. By linearity of expectation, $\mathbf{E}[Y] = \mathbf{E}[Y_1] + \cdots + \mathbf{E}[Y_m] = m/4$. By Lemma 0.15, there exists an assignment $a$ such that $Y(a) \leq m/4$, so $a$ satisfies $\geq 3/4$ fraction of $\varphi$'s NAE-clauses.

Now, we derandomize this using the method of conditional expectations to obtain a poly-time algorithm that, given an $n$-variable 3-NAE formula $\varphi$ with $m$ NAE-clauses, outputs an assignment satisfying $ALG \geq (3/4)m$ NAE-clauses of $\varphi$. Trivially $OPT \leq m$, so $ALG \geq (3/4) \cdot OPT$.

Given any $a_1 \cdots a_i \in \{0,1\}^i$, we can compute

$$\begin{aligned} &\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_i = a_1 \cdots a_i\Big] \\ = \; &\sum_{\text{NAE-clause } C \text{ in } \varphi} \mathbf{Pr}\Big[C \text{ is unsatisfied} \,\Big|\, x_1 \cdots x_i = a_1 \cdots a_i\Big] \end{aligned}$$

(for uniformly random $x_1 \cdots x_n \in \{0,1\}^n$) in poly time: $\mathbf{Pr}\Big[C \text{ is unsatisfied} \,\Big|\, x_1 \cdots x_i = a_1 \cdots a_i\Big]$ is:

- 0 if $a_1 \cdots a_i$ results in a 0 literal and a 1 literal in $C$.
- $2^{-\#\text{ variables with index} > i \text{ in } C}$ if $a_1 \cdots a_i$ assigns at least one variable in $C$, and all assigned literals in $C$ have the same value.
- $1/4$ if $a_1 \cdots a_i$ assigns no variables in $C$.

for $i \leftarrow 1, 2, \ldots, n$:
  compute $\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 1\Big]$
  if this is $\leq m/4$: let $a_i \leftarrow 1$
  else: let $a_i \leftarrow 0$
output $a = a_1 \cdots a_n$

We claim this maintains the invariant that

$$\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_i = a_1 \cdots a_i\Big] \;\leq\; m/4$$

which implies that at the end:

$$\#\text{ NAE-clauses in } \varphi \text{ unsatisfied by } a \;\leq\; m/4$$

(The expectation has no randomness when conditioning on a full assignment $x = a$.) We proved above that the invariant holds at the beginning. To see that the invariant is maintained, assume it holds at the beginning of iteration $i$. By the law of total expectation:

$$\begin{aligned} m/4 \;\geq\; &\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} = a_1 \cdots a_{i-1}\Big] \\ = \; &\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 1\Big] \cdot \\ &\mathbf{Pr}\Big[x_i = 1 \,\Big|\, x_1 \cdots x_{i-1} = a_1 \cdots a_{i-1}\Big] + \\ &\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 0\Big] \cdot \\ &\mathbf{Pr}\Big[x_i = 0 \,\Big|\, x_1 \cdots x_{i-1} = a_1 \cdots a_{i-1}\Big] \end{aligned}$$

$$= \mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 1\Big] \cdot \tfrac{1}{2} + \mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 0\Big] \cdot \tfrac{1}{2}$$

So one of these cases happens:

- $\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 1\Big] \leq m/4$ and the algorithm lets $a_i \leftarrow 1$.
- $\mathbf{E}\Big[\#\text{ unsatisfied NAE-clauses in } \varphi \,\Big|\, x_1 \cdots x_{i-1} x_i = a_1 \cdots a_{i-1} 0\Big] \leq m/4$ and the algorithm lets $a_i \leftarrow 0$.

Either way, the invariant is maintained.

**Exercise 14.15.b:** It suffices to show that for every constant $c' > 15/16$, $(m', c'm')$-Gap Max 3-NAE Sat (with $m'$ NAE-clauses) is NP-complete. For this, it suffices to show that for every constant $c' > 15/16$ and $c = 1-2(1-c') > 7/8$, there exists a poly-time $(m, cm, m', c'm')$-gap-preserving reduction from Max 3-Sat (with $m$ clauses) to Max 3-NAE Sat (with $m'$ NAE-clauses) since $(m, cm)$-Gap Max 3-Sat (with $m$ clauses) is NP-complete (Theorem 14.7).

The reduction from Max 3-Sat (with $m$ clauses) to Max 4-NAE Sat (with $m$ NAE-clauses) in the solution to Exercise 2.30.b is $(m, cm, m, cm)$-gap-preserving.

We claim that the reduction from Max 4-NAE Sat (with $m$ NAE-clauses) to Max 3-NAE Sat (with $m' = 2m$ NAE-clauses) in the solution to Exercise 2.30.c is $(m, cm, m', c'm')$-gap-preserving. Map 4-NAE formula $\varphi$ to a 3-NAE formula $\varphi'$ by replacing the $i^{\text{th}}$ 4-NAE clause $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow \ell_3 \Leftrightarrow \ell_4)$ (where $\ell_1, \ell_2, \ell_3, \ell_4$ are literals) with two 3-NAE clauses $\neg(\ell_1 \Leftrightarrow \ell_2 \Leftrightarrow y_i) \wedge (\ell_3 \Leftrightarrow \ell_4 \Leftrightarrow \overline{y_i})$ where $y_i$ is a fresh variable, for all $i$. We claim that

*(i)* $OPT \geq m \Rightarrow OPT' \geq m'$

*(ii)* $OPT < cm \Rightarrow OPT' < c'm'$

where $OPT$ and $OPT'$ are for $\varphi$ and $\varphi'$ respectively.

*(i)*: In the solution to Exercise 2.30.c, we argued that if $\varphi$ is satisfiable then $\varphi'$ is satisfiable.

*(ii)*: Suppose $OPT' \geq c'm'$. Consider any assignment to $x, y$ satisfying $\geq c'm'$ NAE-clauses of $\varphi'$. Since $\leq (1-c')m' = (1-c)m$ NAE-clauses of $\varphi'$ are falsified, $\leq (1-c)m$ NAE-clauses of $\varphi$ have at least one of their two associated NAE-clauses of $\varphi'$ falsified. Thus $\geq cm$ NAE-clauses of $\varphi$ have both associated NAE-clauses of $\varphi'$ satisfied. These NAE-clauses of $\varphi$ are satisfied by the assignment to $x$ (ignoring $y$) as we argued in the solution to Exercise 2.30.c, so $OPT \geq cm$.

**Exercise 14.16:** We exhibit a poly-time mapping reduction from Two Edge Disjoint Paths to the strongly connected restriction of $(k, k/m^d)$-Gap Max Edge Disjoint Paths (with $m$ edges and $k$ pairs of terminals). Since the former problem is NP-complete (Theorem 14.8), so is the latter, which implies the theorem.

Given input $\big(G, (q_1, r_1), (q_2, r_2)\big)$ to Two Edge Disjoint Paths, we first check that $r_1$ is reachable from $q_1$ and that $r_2$ is reachable from $q_2$. If not both, then it's a 0-input. Thus we may assume $r_1$ is reachable from $q_1$ and that $r_2$ is reachable from $q_2$. Define graph $H$ from $G$ as follows:

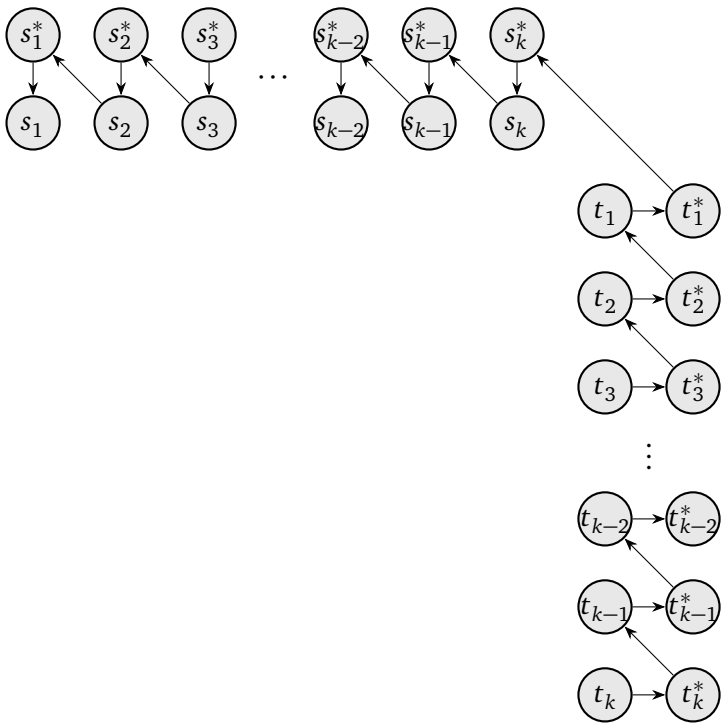
- Add new nodes $q_1^*, r_1^*, q_2^*, r_2^*$ and edges $(q_1^*, q_1)$, $(r_1, r_1^*)$, $(q_2^*, q_2)$, $(r_2, r_2^*)$, and $(q_2^*, r_1^*)$.
- Delete any node $v$ if either:
  - $v$ is reachable from neither $q_1^*$ nor $q_2^*$, or
  - neither $r_1^*$ nor $r_2^*$ is reachable from $v$.

Note that $\big(H, (q_1^*, r_1^*), (q_2^*, r_2^*)\big)$ is a 1-input iff $\big(G, (q_1, r_1), (q_2, r_2)\big)$ is a 1-input:

$\Leftarrow$: If $G$ has edge disjoint paths $q_1 \to \cdots \to r_1$ and $q_2 \to \cdots \to r_2$, then $H$ has edge disjoint paths $q_1^* \to q_1 \to \cdots \to r_1 \to r_1^*$ and $q_2^* \to q_2 \to \cdots \to r_2 \to r_2^*$ without using any deleted nodes.

$\Rightarrow$: If $H$ has edge disjoint paths $q_1^* \to \cdots \to r_1^*$ and $q_2^* \to \cdots \to r_2^*$, then these paths must have the form $q_1^* \to q_1 \to \cdots \to r_1 \to r_1^*$ and $q_2^* \to q_2 \to \cdots \to r_2 \to r_2^*$ without using the edge $(q_2^*, r_1^*)$, since $\text{indeg}(q_1^*) = \text{indeg}(q_2^*) = \text{outdeg}(r_1^*) = \text{outdeg}(r_2^*) = 0$. Thus they yield edge disjoint paths $q_1 \to \cdots \to r_1$ and $q_2 \to \cdots \to r_2$ in $G$.

Define $\big(H', (s_1, t_1), \ldots, (s_k, t_k)\big)$ from $H$ as in the proof of Theorem 14.19. Define $H^*$ from $H'$ by adding nodes $s_1^*, t_1^*, \ldots, s_k^*, t_k^*$ and the following "zigzag" edges:



We claim that:

*(i)* $OPT \geq 2 \Rightarrow OPT' \geq k$

*(ii)* $OPT \leq 1 \Rightarrow OPT' \leq 1 < k/m^d$

where $OPT$ is for $\big(G, (q_1, r_1), (q_2, r_2)\big)$ and thus $\big(H, (q_1^*, r_1^*), (q_2^*, r_2^*)\big)$, and $OPT'$ is for $\big(H^*, (s_1^*, t_1^*), \ldots, (s_k^*, t_k^*)\big)$. This is essentially as in the proof of Theorem 14.19. The zigzag edges don't let edge disjoint paths from $s_i^*$ to $t_i^*$ and from $s_j^*$ to $t_j^*$ cross each other without going through a copy of $H$.

It remains to argue that $H^*$ is strongly connected. We claim that every node $v$ is reachable from $s_1$. Symmetrically, $t_k$ is reachable from every node $u$. Also, $s_1$ is reachable from $t_k$ by the zigzag edges. Thus there exists a walk from $u$ to $t_k$ to $s_1$ to $v$, so $G'$ is strongly connected.

- If $v$ is either some $t_i$ or $t_i^*$ or in a copy of $H$ (in the $i^{\text{th}}$ row) and reachable from $q_2^*$, then a walk from $s_1$ to $v$ can go down the diagonal from top-left toward bottom-right, using the $(q_2^*, r_1^*)$ edges, until reaching $q_2^*$ in the leftmost copy of $H$ in the $i^{\text{th}}$ row, and then go rightward using the $q_2^*$ to $r_2^*$ paths in the copies of $H$.
- If $v$ is either some $s_i$ or $s_i^*$ or in a copy of $H$ (in the $i^{\text{th}}$ column) and reachable from $q_1^*$, then a walk from $s_1$ to $v$ can go to $t_1$ (as above), then to $s_i^*$ using the zigzag edges, then downward using the $q_1^*$ to $r_1^*$ paths in the copies of $H$.

**Exercise 14.17:** To see that $OPT \leq 2$, consider any assignment $a_1 : \{0,1\} \to \{1,2\} \times \{0,1\}$ to the left nodes and $a_2 : \{0,1\} \to \{1,2\} \times \{0,1\}$ to the right nodes. We claim that if two edges are satisfied, then the other two edges must be falsified:

- One case is when the two satisfied edges share an endpoint, say edges $(0,0)$ and $(0,1)$ with $a_1(0) = a_2(0) = a_2(1) = (i,w)$. We must have $(i,w) = (1,0)$ since $a_1(0) = (2,1)$ would falsify edge $(0,0)$, and $a_1(0) = (2,0)$ would falsify edge $(0,1)$, and $a_1(0) = (1,1)$ would falsify both. But then $a_2(0) = (1,0)$ falsifies edge $(1,0)$, and $a_2(1) = (1,0)$ falsifies edge $(1,1)$.
- The other case is when the two satisfied edges are disjoint, say edges $(0,0)$ and $(1,1)$ with $a_1(0) = a_2(0) = (i,0)$ and $a_1(1) = a_2(1) = (j,1)$. But then edge $(0,1)$ is falsified since $a_1(0)$ and $a_2(1)$ disagree on $w$, and edge $(1,0)$ is falsified since $a_1(1)$ and $a_2(0)$ disagree on $w$. (The situation is analogous if edges $(0,1)$ and $(1,0)$ are satisfied.)

Now, we show that $OPT' \geq 8$. For each of the 16 pairs of edges $\big((u_1,u_2),(v_1,v_2)\big)$ of the original graph, the new graph has an edge $\big((u_1,v_1),(u_2,v_2)\big)$. Define $a_1(u_1,v_1) = \big((1,u_1),(2,u_1)\big)$ and $a_2(u_2,v_2) = \big((1,v_2),(2,v_2)\big)$. That is:

- Both provers pick prover 1 on the first repetition and prover 2 on the second repetition.
- Prover 1 responds to both repetitions with the bit given to it in repetition 1.
- Prover 2 responds to both repetitions with the bit given to it in repetition 2.

This satisfies each of the 8 edges with $u_1 = v_2$ since in the original graph, edge $(u_1,u_2)$ is satisfied by labels $(1,u_1)$ and $(1,v_2)$, and edge $(v_1,v_2)$ is satisfied by labels $(2,u_1)$ and $(2,v_2)$.

**Exercise 14.18:** Here's a poly-time $(\alpha, \beta, \alpha, 4\beta)$-gap-preserving reduction from the nonbipartite analogue of Max Unique $k$-Label Cover (with $m$ edges) to the original bipartite version (also with $m$ edges): Map directed graph $G = (V, E)$ to bipartite graph $G' = (V', E')$ by replacing each node $v$ with two nodes $v_{\text{out}}$ (on the left side) and $v_{\text{in}}$ (on the right side) and replacing each edge $e = (u, v)$ with $e' = (u_{\text{out}}, v_{\text{in}})$ where $f_{e'} = f_e$. We claim that

*(i)* $OPT \geq \alpha \Rightarrow OPT' \geq \alpha$

*(ii)* $OPT < \beta \Rightarrow OPT' < 4\beta$

where $OPT$ and $OPT'$ are for $G$ and $G'$ respectively.

*(i)*: Suppose $OPT \geq \alpha$. For any assignment $a : V \to [k]$ satisfying $\geq \alpha$ edges of $G$, the assignment $a' : V' \to [k]$ defined by $a'(v_{\text{out}}) = a'(v_{\text{in}}) = a(v)$ satisfies the corresponding edges of $G'$. This is because for every $e = (u, v) \in E$, we have $f_{e'}\big(a'(u_{\text{out}}), a'(v_{\text{in}})\big) = f_e\big(a(u), a(v)\big)$. Thus $OPT' \geq \alpha$.

*(ii)*: Suppose $OPT' \geq 4\beta$. Consider any assignment $a' : V' \to [k]$ satisfying $\geq 4\beta$ edges of $G'$. We use the probabilistic method to show there exists an assignment $a : V \to [k]$ satisfying $\geq \beta$ edges of $G$. For each $v \in V$ independently, assign $a(v) = a'(v_{\text{out}})$ with probability $1/2$, and $a(v) = a'(v_{\text{in}})$ with probability $1/2$. For every $e = (u, v) \in E$, with probability $\geq (1/2)^2 = 1/4$ we have $a(u) = a'(u_{\text{out}})$ and $a(v) = a'(v_{\text{in}})$, in which case $f_e\big(a(u), a(v)\big) = f_{e'}\big(a'(u_{\text{out}}), a'(v_{\text{in}})\big)$ and so $e$ is satisfied if $e'$ is. For each $e \in E$, let $X_e$ be the indicator random variable for $e$ being satisfied, and let $X = \sum_{e \in E} X_e$ be the number of satisfied edges in $G$.

$$\begin{aligned}
\mathbf{E}[X] &= \textstyle\sum_{e \in E} \mathbf{E}[X_e] && \text{(linearity of expectation)} \\
&= \textstyle\sum_{e \in E} \mathbf{Pr}[a \text{ satisfies } e] \\
&\geq \textstyle\sum_{e \in E \,:\, a' \text{ satisfies } e'} 1/4 \\
&\geq (4\beta) \cdot (1/4) \\
&= \beta
\end{aligned}$$

Thus some outcome $a$ satisfies $X(a) \geq \mathbf{E}[X] \geq \beta$ edges of $G$, so $OPT \geq \beta$.

The uniqueness property was irrelevant for this proof.

In conclusion, assuming the nonbipartite unique games conjecture holds, so does the original bipartite version: Consider any constants $c < 1$ and $d' > 0$. Then for $d = d'/4$, there exists a constant $k$ such that nonbipartite $(cm, dm)$-Gap Max Unique $k$-Label Cover (with $m$ edges) is NP-complete, by the nonbipartite unique games conjecture. By the above reduction, bipartite $(cm, d'm)$-Gap Max Unique $k$-Label Cover (with $m$ edges) is NP-complete, so the bipartite unique games conjecture holds.

**Exercise 14.19:** Assume $(m, bm)$-Gap Max 3-Sat is NP-complete for some constant $b < 1$. To see that $\mathsf{NP} \subseteq \mathsf{PCP}[\log N, k]$ for some constant $k$ (depending on $b$), consider any $A \in \mathsf{NP}$. By assumption, $A$ has a poly-time mapping reduction to $(m, bm)$-Gap Max 3-Sat. Consider this randomized verifier $V$ for $A$ on a valid input $x$:

> run the mapping reduction on input $x$ to get an $m$-clause 3-CNF $\varphi_x$
> view the purported witness $w$ as an assignment to $\varphi_x$'s variables
> pick a uniformly random clause $C$ of $\varphi_x$
> query the three bits of $w$ corresponding to variables in $C$
> accept iff these three bits of $w$ satisfy $C$

$\mathbf{Pr}_r\big[V(x;w;r) \text{ accepts}\big]$ is the fraction of clauses of $\varphi_x$ satisfied by $w$. This $V$ has completeness 1 and soundness $b$:

$$\begin{aligned} A(x)=1 &\Rightarrow OPT(\varphi_x) \geq m \\ &\Rightarrow \big(\exists w:\ w \text{ satisfies all } m \text{ clauses of } \varphi_x\big) \\ &\Rightarrow \big(\exists w:\ \mathbf{Pr}_r\big[V(x;w;r) \text{ accepts}\big]=1\big) \end{aligned}$$

$$\begin{aligned} A(x)=0 &\Rightarrow OPT(\varphi_x) < bm \\ &\Rightarrow \big(\forall w:\ w \text{ satisfies less than } bm \text{ clauses of } \varphi_x\big) \\ &\Rightarrow \big(\forall w:\ \mathbf{Pr}_r\big[V(x;w;r) \text{ accepts}\big]<b\big) \end{aligned}$$

Technicality: If $m$ isn't a power of 2, then $V$ can't exactly sample a uniformly random clause using uniformly random bits. Instead, $V$ can let $m' < 2m$ be the least power of 2 that's $\geq m$, sample uniformly random $r \in [m']$, and if $r \leq m$ then check whether the $r^{\text{th}}$ clause is satisfied, and if $r > m$ then just accept. The completeness is still 1, and the soundness is $1-(1-b)/2 < 1$ since for all $w$, $V(x;w;r)$ rejects for $> (1-b)m > ((1-b)/2)m'$ many $r$.

To improve the soundness to $1/3$, we run $V$ $\ell$ times independently (with fresh randomness but the same $w$ each time) and accept iff all runs accept. We need $\ell$ to be a large enough constant that $\big(1-(1-b)/2\big)^{\ell} \leq 1/3$. The query efficiency becomes $k = 3\ell$, and the randomness efficiency is $O(\ell \log N) = O(\log N)$. Thus $A \in \mathsf{PCP}[\log N, k]$.

**Exercise 14.20.a:** Here's the algorithm:

```
on input φ(x_1 ··· x_n):
    for each i ∈ [n]:
        if φ contains at least as many (x_i) clauses as (¬x_i) clauses: assign a_i ← 1
        else: assign a_i ← 0
    return a = a_1 ··· a_n
```

No assignment to $x$ satisfies more clauses of $\varphi$ than $a$ does, because for each $i$, no assignment to $x_i$ satisfies more of the $(x_i)$ and $(\overline{x_i})$ clauses than $a_i$ does.

**Exercise 14.20.b:** Suppose $A \in \mathsf{PCP}_{c,s}[\log N, 1]$. Let $V$ be a poly-time randomized verifier for $A$ with completeness $c$, soundness $s$, randomness efficiency $\leq a \log N$ (for an integer $a$), and query efficiency $\leq 1$. Assume $w \in \{0,1\}^n$ where $n = dN^e$ (for integers $d, e$). For each valid input $x$ of size $N$ and each $r \in \{0,1\}^{a \log N}$, either $V$ accepts or rejects without querying $w$, or there exist:

- An index $i_{x,r} \in \{0, \ldots, n-1\}$ of $w$ queried by $V(x; w; r)$.
- A nonconstant function $f_{x,r} : \{0,1\} \to \{0,1\}$ such that $V(x; w; r)$ accepts iff $f_{x,r}\left(w_{i_{x,r}}\right) = 1$. Note that $f_{x,r}$ is one of the literals $w_{i_{x,r}}$ or $\overline{w_{i_{x,r}}}$.

Then $\max_{w \in \{0,1\}^n} \mathbf{Pr}_{r \in \{0,1\}^{a \log N}}\left[V(x; w; r) \text{ accepts}\right]$ equals $1/N^a$ times:

$$\begin{aligned}&\big(\text{number of } r \text{ such that } V(x; w; r) \text{ accepts without querying } w\big) + \\ &\textstyle\sum_{j \in [n]} \max\Big(\big(\text{number of } r \text{ such that } f_{x,r} \text{ is } w_j\big),\ \big(\text{number of } r \text{ such that } f_{x,r} \text{ is } \overline{w_j}\big)\Big)\end{aligned}$$

On input $x$, a poly-time algorithm can compute this quantity and accept iff it's $\geq c$. Thus $A \in \mathsf{P}$.

**Exercise 14.21:** Since SAT is NP-complete via $\leq_\ell^m$ (Theorem 2.40), it suffices to prove SAT $\in$ NLPCP$[\log N]$. Say the input to SAT is an $n$-variable $m$-clause CNF $\varphi(x)$. As a technicality, we may assume $m$ is a power of 2 by duplicating clauses if necessary.

First, we show SAT $\in$ NLPCP$[\operatorname{poly} N]$ (with $\operatorname{poly} N$ randomness efficiency):

```
view w as a sequence of 2m many assignments w^1, ..., w^{2m} to x
for i ← 1, ..., 2m:
    pick a uniformly random clause of φ
    if w^i falsifies the clause: reject
accept
```

We implement this as a log-space randomized streaming verifier $V$: For each assignment $w^i$, $V$ checks whether $w^i$ falsifies the randomly chosen clause by paying attention to the values of the clause's variables as it reads through $w^i$. This has completeness 1 because if $\varphi$ is satisfiable, then $\mathbf{Pr}_r\big[V(\varphi; w; r) \text{ accepts}\big] = 1$ when each $w^i$ is a satisfying assignment. This has soundness $1/3$ because if $\varphi$ is unsatisfiable, then for all $w^1, \ldots, w^{2m}$:

$$\begin{aligned}
\mathbf{Pr}_r\big[V(\varphi; w; r) \text{ accepts}\big] &= \textstyle\prod_{i=1}^{2m} \mathbf{Pr}\big[w^i \text{ satisfies a uniformly random clause of } \varphi\big] \\
&\leq \textstyle\prod_{i=1}^{2m} (1 - 1/m) \\
&\leq (\mathrm{e}^{-1/m})^{2m} && \text{(Fact 7.6)} \\
&= \mathrm{e}^{-2} \\
&\leq 1/3
\end{aligned}$$

The randomness efficiency is $2m \log m$. To prove SAT $\in$ NLPCP$[\log N]$, we improve the randomness efficiency to $O(\log m)$: Identifying $[m]$ and $[4m]$ with $\{0,1\}^{\log m}$ and $\{0,1\}^{\log m + 2}$, let $h\colon \{0,1\}^{3 \log m + 1} \times [4m] \to [m]$ be the pairwise uniform hash function from Lemma 8.9, which is computable in $O(\log m)$ space.

```
view w as a sequence of 3m many assignments w^1, ..., w^{3m} to x
sample r ∈ {0,1}^{3 log m + 1} uniformly at random
for i ← 1, ..., 3m:
    if w^i falsifies φ's h_r(i)^th clause: reject
accept
```

We implement this as a log-space randomized streaming verifier $V'$. This has completeness 1 because if $\varphi$ is satisfiable, then $\mathbf{Pr}_r\big[V'(\varphi; w; r) \text{ accepts}\big] = 1$ when each $w^i$ is a satisfying assignment. To see that this has soundness $1/3$, suppose $\varphi$ is unsatisfiable and consider any $w^1, \ldots, w^{3m}$. Define a function $f\colon [3m] \to [m]$ such that for all $i$, $w^i$ falsifies $\varphi$'s $f(i)^{\text{th}}$ clause. If $h_r(i) = f(i)$ for some $i$, then $V'(\varphi; w; r)$ rejects. Thus by Exercise 8.12:

$$\mathbf{Pr}_r\big[V'(\varphi; w; r) \text{ accepts}\big] \leq \mathbf{Pr}_r\big[\forall i \in [3m] \colon h_r(i) \neq f(i)\big] \leq m/3m = 1/3$$

**Exercise 14.22:** Consider any $b \in \{0,1\}^{n^3}$ such that $b \neq a \otimes a \otimes a$ for all $a \in \{0,1\}^n$. Consider $a$ defined by $a_i = b_{i,i,i}$ for all $i \in [n]$, and let $c = a \otimes a \otimes a \neq b$. Note that

$$u' \odot b \;=\; \textstyle\sum_i u'_{i,i,i} b_{i,i,i} \;=\; \sum_i u_i a_i \;=\; u \odot a$$

and $v' \odot b = v \odot a$ and $w' \odot b = w \odot a$. Note that:

$$\begin{aligned}(u' \odot b)(v' \odot b)(w' \odot b) &= (u \odot a)(v \odot a)(w \odot a)\\ &= \big(\textstyle\sum_i u_i a_i\big)\big(\sum_j v_j a_j\big)\big(\sum_k w_k a_k\big)\\ &= \textstyle\sum_{i,j,k}(u_i v_j w_k)(a_i a_j a_k)\\ &= (u \otimes v \otimes w) \odot (a \otimes a \otimes a)\\ &= (u \otimes v \otimes w) \odot c\end{aligned}$$

Next, we overload the $\odot$ notation for when the operands have different lengths:

- If $y \in \{0,1\}^n$ and $z \in \{0,1\}^{n^3}$, define $y \odot z \in \{0,1\}^{n^2}$ by $(y \odot z)_{i,j} = y \odot (z_{i,j,1}, \ldots, z_{i,j,n})$.
- If $y \in \{0,1\}^n$ and $z \in \{0,1\}^{n^2}$, define $y \odot z \in \{0,1\}^n$ by $(y \odot z)_i = y \odot (z_{i,1}, \ldots, z_{i,n})$.

Note that

$$(u \otimes v \otimes w) \odot b \;=\; \textstyle\sum_{i,j,k} u_i v_j w_k b_{i,j,k} \;=\; \sum_i u_i \sum_j v_j \sum_k w_k b_{i,j,k} \;=\; u \odot \big(v \odot (w \odot b)\big)$$

and $(u \otimes v \otimes w) \odot c = u \odot \big(v \odot (w \odot c)\big)$.

- Since $b \neq c$, we have $b_{i,j,k} \neq c_{i,j,k}$ for some $i, j, k$, so by the random subsum principle:

$$\mathbf{Pr}_w\big[(w \odot b)_{i,j} \neq (w \odot c)_{i,j}\big] \;=\; 1/2$$

- If $(w \odot b)_{i,j} \neq (w \odot c)_{i,j}$ then by the random subsum principle:

$$\mathbf{Pr}_v\Big[\big(v \odot (w \odot b)\big)_i \neq \big(v \odot (w \odot c)\big)_i\Big] \;=\; 1/2$$

- If $\big(v \odot (w \odot b)\big)_i \neq \big(v \odot (w \odot c)\big)_i$ then by the random subsum principle:

$$\mathbf{Pr}_u\Big[u \odot \big(v \odot (w \odot b)\big) \neq u \odot \big(v \odot (w \odot c)\big)\Big] \;=\; 1/2$$

By the chain rule:

$$\begin{aligned}&\mathbf{Pr}_{u,v,w}\big[(u \otimes v \otimes w) \odot b \neq (u' \odot b)(v' \odot b)(w' \odot b)\big]\\ =\;& \mathbf{Pr}_{u,v,w}\big[(u \otimes v \otimes w) \odot b \neq (u \otimes v \otimes w) \odot c\big]\\ =\;& \mathbf{Pr}_{u,v,w}\Big[u \odot \big(v \odot (w \odot b)\big) \neq u \odot \big(v \odot (w \odot c)\big)\Big]\\ \geq\;& \mathbf{Pr}_{u,v,w}\Big[u \odot \big(v \odot (w \odot b)\big) \neq u \odot \big(v \odot (w \odot c)\big) \;\Big|\; (w \odot b)_{i,j} \neq (w \odot c)_{i,j} \text{ and } \big(v \odot (w \odot b)\big)_i \neq \big(v \odot (w \odot c)\big)_i\Big] \cdot\\ &\mathbf{Pr}_{v,w}\Big[\big(v \odot (w \odot b)\big)_i \neq \big(v \odot (w \odot c)\big)_i \;\Big|\; (w \odot b)_{i,j} \neq (w \odot c)_{i,j}\Big] \cdot\\ &\mathbf{Pr}_w\big[(w \odot b)_{i,j} \neq (w \odot c)_{i,j}\big]\\ =\;& (1/2) \cdot (1/2) \cdot (1/2) \;=\; 1/8\end{aligned}$$

**Exercise 14.23:** On input $(M, d)$, the purported witness is some $(w^1, w^2)$ where $w^1 : \{0,1\}^n \to \{0,1\}$ and $w^2 : \{0,1\}^{n^2} \to \{0,1\}$, and the randomness is $(q, r, s^1, t^1, s^2, t^2, u, v)$ where:

$$\begin{aligned} q &\in \{0,1\}^{n^2} && \text{(for locally correcting } w^2\text{)} \\ r &\in \{0,1\}^m && \text{(for the linear combination of equations; } r \text{ is a } 1 \times m \text{ vector)} \\ s^1, t^1 &\in \{0,1\}^n && \text{(for locally testing linearity of } w^1\text{)} \\ s^2, t^2 &\in \{0,1\}^{n^2} && \text{(for locally testing linearity of } w^2\text{)} \\ u, v &\in \{0,1\}^n && \text{(for checking that } w^2 \text{ is quadratically consistent with } w^1\text{)} \end{aligned}$$

- Define $w^2_q : \{0,1\}^{n^2} \to \{0,1\}$ by $w^2_q(z) = w^2(z \oplus q) \oplus w^2(q)$, which is two queries to $w^2$.
- Define $u \otimes v \in \{0,1\}^{n^2}$ by $(u \otimes v)_{i,j} = u_i v_j$.

$V(M, d; w^1, w^2; q, r, s^1, t^1, s^2, t^2, u, v)$:
accept iff $w^2_q(rM) = rd$
and $w^1(s^1 \oplus t^1) = w^1(s^1) \oplus w^1(t^1)$
and $w^2(s^2 \oplus t^2) = w^2(s^2) \oplus w^2(t^2)$
and $w^2_q(u \otimes v) = w^1(u)w^1(v)$

$V$ has query efficiency 12.

We prove $V$ has completeness 1. Suppose $M(a \otimes a) = d$ for some $a \in \{0,1\}^n$.

- Define $w^1 : \{0,1\}^n \to \{0,1\}$ by $w^1(y) = y \odot a$.
- Define $w^2 : \{0,1\}^{n^2} \to \{0,1\}$ by $w^2(z) = z \odot b$ where $b = a \otimes a$.

Then $\mathbf{Pr}_{q,r,s^1,t^1,s^2,t^2,u,v}\big[V(M, d; w^1, w^2; q, r, s^1, t^1, s^2, t^2, u, v) \text{ accepts}\big] = 1$ because $w^2_q = w^2$ and:

$$\begin{aligned} w^2(rM) &= (rM)b = r(Mb) = rd \\ w^1(s^1 \oplus t^1) &= (s^1 \oplus t^1) \odot a = (s^1 \odot a) \oplus (t^1 \odot a) = w^1(s^1) \oplus w^1(t^1) \\ w^2(s^2 \oplus t^2) &= (s^2 \oplus t^2) \odot b = (s^2 \odot b) \oplus (t^2 \odot b) = w^2(s^2) \oplus w^2(t^2) \\ w^2(u \otimes v) &= (u \otimes v) \odot (a \otimes a) = (u \odot a)(v \odot a) = w^1(u)w^1(v) \end{aligned}$$

We prove $V$ has soundness 19/20. Suppose $M(a \otimes a) \neq d$ for all $a \in \{0,1\}^n$. Consider any $w^1 : \{0,1\}^n \to \{0,1\}$ and $w^2 : \{0,1\}^{n^2} \to \{0,1\}$. In all four of the following cases:

$$\mathbf{Pr}_{q,r,s^1,t^1,s^2,t^2,u,v}\big[V(M, d; w^1, w^2; q, r, s^1, t^1, s^2, t^2, u, v) \text{ accepts}\big] \leq 19/20$$

**Case 1:** For some $a$ and $b = a \otimes a$, $\mathbf{Pr}_q\big[w^2(q) \neq q \odot b\big] \leq 1/20$. Viewing $r$ as a $1 \times m$ vector and $b$ as an $n^2 \times 1$ vector,

$$\begin{aligned} \mathbf{Pr}_{q,r}\big[w^2_q(rM) = rd\big] &\leq \mathbf{Pr}_{q,r}\big[r(Mb) = rd \text{ or } w^2_q(rM) \neq (rM)b\big] \\ &\leq \mathbf{Pr}_r\big[r(Mb) = rd\big] + \mathbf{Pr}_{q,r}\big[w^2_q(rM) \neq (rM)b\big] \\ &\leq 1/2 + 1/10 \leq 19/20 \end{aligned}$$

by a union bound and the random subsum principle, since $Mb \neq d$.

**Case 2:** For all $a$, $\mathbf{Pr}_p\left[w^1(p) \neq p \odot a\right] \geq 1/20$ over uniformly random $p \in \{0,1\}^n$. Then we have $\mathbf{Pr}_{s^1,t^1}\left[w^1(s^1 \oplus t^1) = w^1(s^1) \oplus w^1(t^1)\right] \leq 19/20$ by Theorem 14.28.

**Case 3:** For all $b$ (not necessarily of the form $a \otimes a$), $\mathbf{Pr}_q\left[w^2(q) \neq q \odot b\right] \geq 1/20$. Then we have $\mathbf{Pr}_{s^2,t^2}\left[w^2(s^2 \oplus t^2) = w^2(s^2) \oplus w^2(t^2)\right] \leq 19/20$ by Theorem 14.28.

**Case 4:** For some $a$ and $b \neq a \otimes a$, $\mathbf{Pr}_p\left[w^1(p) \neq p \odot a\right] \leq 1/20$ and $\mathbf{Pr}_q\left[w^2(q) \neq q \odot b\right] \leq 1/20$. By a union bound:

$$\begin{aligned}
& \mathbf{Pr}_{q,u,v}\left[w^2_q(u \otimes v) = w^1(u)w^1(v)\right] \\
\leq\ & \mathbf{Pr}_{q,u,v}\Big[(u \otimes v) \odot b = (u \odot a)(v \odot a) \text{ or} \\
& \qquad w^2_q(u \otimes v) \neq (u \otimes v) \odot b \text{ or } w^1(u) \neq u \odot a \text{ or } w^1(v) \neq v \odot a\Big] \\
\leq\ & \mathbf{Pr}_{u,v}\left[(u \otimes v) \odot b = (u \odot a)(v \odot a)\right] + \\
& \mathbf{Pr}_{q,u,v}\left[w^2_q(u \otimes v) \neq (u \otimes v) \odot b\right] + \mathbf{Pr}_u\left[w^1(u) \neq u \odot a\right] + \mathbf{Pr}_v\left[w^1(v) \neq v \odot a\right] \\
\leq\ & \mathbf{Pr}_{u,v}\left[(u \otimes v) \odot b = (u \odot a)(v \odot a)\right] + 1/10 + 1/20 + 1/20
\end{aligned}$$

Let $c = a \otimes a$, so $(u \odot a)(v \odot a) = (u \otimes v) \odot c$. We have

$$\mathbf{Pr}_{u,v}\left[(u \otimes v) \odot b \neq (u \odot a)(v \odot a)\right] = \mathbf{Pr}_{u,v}\left[(u \otimes v) \odot b \neq (u \otimes v) \odot c\right] \geq 1/4$$

as in the proof of Claim 14.33. Putting everything together:

$$\mathbf{Pr}_{q,u,v}\left[w^2_q(u \otimes v) = w^1(u)w^1(v)\right] \leq 3/4 + 1/10 + 1/20 + 1/20 = 19/20$$

**Exercise 14.24:** The reduction runs in poly time since $m' = 2^{\ell} = 2^{O(\log m)} = \text{poly}\, m$.

Consider any assignment $a \in \{0,1\}^n$. By the random subsum principle (Lemma 7.2.*(ii)*):

$$\mathbf{Pr}_{r \sim U_m}\big[(rM)(a \otimes a) = rd\big] \;=\; \mathbf{Pr}_{r \sim U_m}\big[r(M(a \otimes a)) = rd\big] \begin{cases} = 1 & \text{if } M(a \otimes a) = d \\ = 1/2 & \text{if } M(a \otimes a) \neq d \end{cases}$$

By the derandomized random subsum principle (Observation 11.9.*(ii)*):

$$\mathbf{Pr}_{r \sim G(U_\ell)}\big[(rM)(a \otimes a) = rd\big] \;=\; \mathbf{Pr}_{r \sim G(U_\ell)}\big[r(M(a \otimes a)) = rd\big] \begin{cases} = 1 & \text{if } M(a \otimes a) = d \\ \leq 1/2 + \varepsilon < c & \text{if } M(a \otimes a) \neq d \end{cases}$$

This implies:

- If some $a$ satisfies $M(x \otimes x) = d$, then the same $a$ satisfies each of the equations $(rM)(x \otimes x) = rd$ for $r = G(s)$.
- If every $a$ doesn't satisfy $M(x \otimes x) = d$, then every $a$ satisfies $< c$ fraction of the equations $(rM)(x \otimes x) = rd$ for $r = G(s)$.

Thus for $OPT = \max_{a \in \{0,1\}^n}\big(\text{number of } s \in \{0,1\}^{\ell} \text{ such that } (rM)(a \otimes a) = rd \text{ where } r = G(s)\big)$:

$$OPT \begin{cases} = m' & \text{if } M(x \otimes x) = d \text{ is satisfiable} \\ < cm' & \text{if } M(x \otimes x) = d \text{ is unsatisfiable} \end{cases}$$

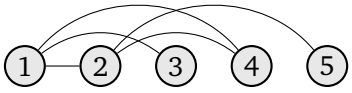
This shows that the reduction is correct.

In conclusion, $(m', cm')$-Gap Max $\mathbb{Z}_2$ Quadratic System Sat is NP-complete since $\mathbb{Z}_2$ Quadratic System Sat is NP-complete (Lemma 14.31), so Max $\mathbb{Z}_2$ Quadratic System Sat has no poly-time $c$-approximation algorithm unless P = NP (Lemma 14.6).

**Exercise 14.25:**

$$\begin{aligned}
\langle f, g \rangle &= \left\langle \textstyle\sum_I \widehat{f}_I \chi_I, \sum_J \widehat{g}_J \chi_J \right\rangle && \text{(Lemma 14.35)} \\
&= \textstyle\sum_{I,J} \widehat{f}_I \widehat{g}_J \langle \chi_I, \chi_J \rangle && \text{(linearity)} \\
&= \textstyle\sum_I \widehat{f}_I \widehat{g}_I \, 1 + \sum_{I, J \neq I} \widehat{f}_I \widehat{g}_J \, 0 && \text{(Lemma 14.34)} \\
&= \textstyle\sum_I \widehat{f}_I \widehat{g}_I
\end{aligned}$$

**Exercise 15.1.a:**



**Exercise 15.1.b:** If $p$ is an automorphism of $G$, then $p(2) = 2$ since 2 is the only node with degree 2. Its neighbors 3 and 5 could be swapped, in which case their respective neighbors 1 and 4 would need to be swapped. So $G$ has two automorphisms:

| $v$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p(v)$ | 1 | 2 | 3 | 4 | 5 |

| $v$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p(v)$ | 4 | 2 | 5 | 1 | 3 |

By Lemma 15.3, $5!/2 = 60$ graphs are isomorphic to $G$.

**Exercise 15.2.a:** Map $(G_1, G_2, p)$ to $(G_1', G_2')$ where $G_1'$ is $G_1$ with a clique of size $n + 2v + 1$ attached (by a single edge) to node $v \in [n]$ for each $v$ such that $p(v)$ is defined, and $G_2'$ is $G_2$ with a clique of size $n + 2v + 1$ attached (by a single edge) to node $p(v)$ for each $v$ such that $p(v)$ is defined. To show this poly-time mapping reduction is correct, we argue that there exists an isomorphism from $G_1$ to $G_2$ that's consistent with $p$ iff there exists an isomorphism from $G_1'$ to $G_2'$:

$\Rightarrow$: If $q$ is an isomorphism from $G_1$ to $G_2$ that's consistent with $p$, then we can turn $q$ into an isomorphism $q'$ from $G_1'$ to $G_2'$ by mapping the clique attached to $v$ in $G_1'$ to the clique attached to $p(v)$ in $G_2'$.

$\Leftarrow$: If $q'$ is an isomorphism from $G_1'$ to $G_2'$, then $q'$ maps the clique attached to $v$ in $G_1'$ to the clique attached to $p(v)$ in $G_2'$ (since those are the only nodes with those degrees) and thus maps $v$ to $p(v)$ for each $v$ such that $p(v)$ is defined. Ignoring the added cliques yields an isomorphism $q$ from $G_1$ to $G_2$ that's consistent with $p$.

**Exercise 15.2.b:** On input $(G_1, G_2, p)$ where $G_1$ and $G_2$ have nodes $[n]$ and $p\colon [n] \to [n]$ is a partial permutation: Define the partial permutation $p'\colon [n] \to [n]$ by $p'(p(v)) = p(v)$ for all $v$ such that $p(v)$ is defined.

Arthur picks $i \in \{1,2\}$ uniformly at random
if $i = 1$: Arthur picks a permutation $q_1\colon [n] \to [n]$ consistent with $p$ uniformly at random
if $i = 2$: Arthur picks a permutation $q_2\colon [n] \to [n]$ consistent with $p'$ uniformly at random
Arthur sends Merlin $q_i(G_i)$
Merlin responds with $j \in \{1,2\}$
Arthur accepts iff $j = i$

Completeness 1: Suppose there's no isomorphism from $G_1$ to $G_2$ that's consistent with $p$. There do not exist permutations $q_1$ consistent with $p$ and $q_2$ consistent with $p'$ such that $q_1(G_1) = q_2(G_2)$, because otherwise $(q_1 q_2^{-1})(G_1) = G_2$ and the permutation $q_1 q_2^{-1}$ is consistent with $p$. Thus Merlin can tell whether the graph Arthur sent is $q_1(G_1)$ for some permutation $q_1$ consistent with $p$, or $q_2(G_2)$ for some permutation $q_2$ consistent with $p'$.

Soundness 1/2: Suppose $r$ is an isomorphism from $G_1$ to $G_2$ that's consistent with $p$. The distribution $q_1(G_1)$ is the same as the distribution $q_2(G_2) = q_2(r(G_1)) = (rq_2)(G_1)$ because if $q_2$ is a uniformly random permutation consistent with $p'$, then $rq_2$ is a uniformly random permutation consistent with $p$. It follows that $q_i(G_i)$ is independent of $i$, so Merlin's response $j$—which depends on Arthur's message $q_i(G_i)$—is also independent of $i$. Therefore Arthur's acceptance probability is:

$$\begin{aligned}
\mathbf{Pr}[j = i] &= \mathbf{Pr}[j = i = 1] + \mathbf{Pr}[j = i = 2] \\
&= \mathbf{Pr}[j = 1] \cdot \mathbf{Pr}[i = 1] + \mathbf{Pr}[j = 2] \cdot \mathbf{Pr}[i = 2] && (j \text{ and } i \text{ are independent}) \\
&= \mathbf{Pr}[j = 1]/2 + \mathbf{Pr}[j = 2]/2 && (i \text{ is uniformly distributed}) \\
&= 1/2
\end{aligned}$$

The soundness can be amplified as in the proof of Theorem 15.1.

**Exercise 15.3:** This is essentially the same as the proof that BPP $\subseteq$ P/poly (Theorem 7.9). Suppose $A \in$ AM and $A$ has word size $W = O(\log N)$, so each input of size $N$ is an element of $(\{0,1\}^W)^N$. By amplification, $A$ has a poly-time randomized verifier $V$ with completeness $> 1-\varepsilon$ and soundness $< \varepsilon$ where $\varepsilon = 2^{-NW}$. Let $B =$ Merlin of $V$.

$$A(x) = 1 \;\Rightarrow\; \mathbf{Pr}_r\big[B(x,r) = 1\big] > 1 - \varepsilon$$
$$A(x) = 0 \;\Rightarrow\; \mathbf{Pr}_r\big[B(x,r) = 1\big] < \varepsilon$$

That is, $\mathbf{Pr}_r\big[B(x,r) \neq A(x)\big] < \varepsilon$ for every valid input $x$. For every $N$, by a union bound:

$$\begin{aligned}\mathbf{Pr}_r\big[\exists \text{ valid } x \text{ of size } N :\; B(x,r) \neq A(x)\big] &\leq \textstyle\sum_{\text{valid } x \text{ of size } N} \mathbf{Pr}_r\big[B(x,r) \neq A(x)\big] \\ &< 2^{NW} \cdot \varepsilon \;=\; 1\end{aligned}$$

Hence for every $N$, there exists an outcome $r$ such that $B(x,r) = A(x)$ for every valid $x$ of size $N$. Treat this particular $r$ as the advice $a_N$. Then $A \in$ NP/poly by a nonuniform verifier $V'$ where $V'(x; w; a_N)$ runs $V(x; a_N; w)$:

$$A(x) = 1 \iff B(x, a_N) = 1 \iff \big(\exists w :\; V(x; a_N; w) \text{ accepts}\big) \iff \big(\exists w :\; V'(x; w; a_N) \text{ accepts}\big)$$

**Exercise 15.4:** First, we prove that MA is unchanged if we require completeness 1. Suppose $A \in \mathsf{MA}$ by a randomized verifier $V$. Let $B = \textsc{Arthur of } V \in \mathsf{BPP}$. In the proof of Theorem 7.10, we designed a poly-time 2-witness verifier $V'(x,w;s;r)$ such that the following hold, where $\widehat{\exists}$ means "for at least 2/3 fraction of":

$$\begin{aligned} B(x,w) = 1 &\Rightarrow \exists s\ \forall r:\ V'(x,w;s;r) \text{ accepts} \\ B(x,w) = 0 &\Rightarrow \forall s\ \widehat{\exists} r:\ V'(x,w;s;r) \text{ rejects} \end{aligned}$$

Then $A \in \mathsf{MA}$ by a randomized verifier $V''$ where $V''(x;w,s;r)$ runs $V'(x,w;s;r)$, and $V''$ has completeness 1 and soundness 1/3:

$$\begin{aligned} A(x) = 1 &\Rightarrow \exists w:\ B(x,w) = 1 \\ &\Rightarrow \exists w\ \exists s\ \forall r:\ V'(x,w;s;r) \text{ accepts} \\ &\Rightarrow \exists w,s:\ \mathbf{Pr}_r\big[V''(x;w,s;r) \text{ accepts}\big] = 1 \end{aligned}$$

$$\begin{aligned} A(x) = 0 &\Rightarrow \forall w:\ B(x,w) = 0 \\ &\Rightarrow \forall w\ \forall s\ \widehat{\exists} r:\ V'(x,w;s;r) \text{ rejects} \\ &\Rightarrow \forall w,s:\ \mathbf{Pr}_r\big[V''(x;w,s;r) \text{ accepts}\big] \leq 1/3 \end{aligned}$$

Now, we prove that AM is unchanged if we require completeness 1. Suppose $A \in \mathsf{AM}$. By amplification, $A$ has a randomized verifier $V$ with completeness $1 - 1/2^N$, soundness 1/3, time efficiency $\leq T = cN^d$ (for integers $c,d$), and log-bounded word size $W$. Let $B = \textsc{Merlin of } V$:

$$\begin{aligned} A(x) = 1 &\Rightarrow \mathbf{Pr}_r\big[B(x,r) = 1\big] \geq 1 - 1/2^N \\ A(x) = 0 &\Rightarrow \mathbf{Pr}_r\big[B(x,r) = 1\big] \leq 1/3 \end{aligned}$$

Denote $s = (s_1, \ldots, s_k)$ where each $s_i \in (\{0,1\}^W)^T$ for $k = TW \in \text{poly}N$. By the same calculations as in the proof of $\mathsf{AM} \subseteq \Pi_2\mathsf{P}$ (Theorem 15.7) except with $\leq 1/3$ instead of $< 1$ in the $A(x) = 0$ case:

$$\begin{aligned} A(x) = 1 &\Rightarrow \forall s\ \exists r\ \forall i:\ B(x, r \oplus s_i) = 1 \\ A(x) = 0 &\Rightarrow \widehat{\exists} s\ \forall r\ \exists i:\ B(x, r \oplus s_i) = 0 \end{aligned}$$

Denote $w = (w_1, \ldots, w_k)$ where each $w_i \in (\{0,1\}^W)^T$. Define a randomized verifier $V'$ where $V'(x;s;r,w)$ runs $V(x;r \oplus s_i;w_i)$ for each $i$ and accepts iff all of these runs accept. For all $s$ and $r$:

$$\begin{aligned} \big(\forall i:\ B(x, r \oplus s_i) = 1\big) &\Leftrightarrow \big(\exists w\ \forall i:\ V(x;r \oplus s_i;w_i) \text{ accepts}\big) \\ &\Leftrightarrow \big(\exists w:\ V'(x;s;r,w) \text{ accepts}\big) \end{aligned}$$

Then $V'$ has completeness 1 and soundness 1/3 for $A \in \mathsf{AM}$:

$$\begin{aligned} A(x) = 1 &\Rightarrow \forall s\ \exists r\ \exists w:\ V'(x;s;r,w) \text{ accepts} \\ &\Rightarrow \mathbf{Pr}_s\big[\exists r,w:\ V'(x;s;r,w) \text{ accepts}\big] = 1 \end{aligned}$$

$$\begin{aligned} A(x) = 0 &\Rightarrow \widehat{\exists} s\ \forall r\ \forall w:\ V'(x;s;r,w) \text{ rejects} \\ &\Rightarrow \mathbf{Pr}_s\big[\exists r,w:\ V'(x;s;r,w) \text{ accepts}\big] \leq 1/3 \end{aligned}$$

**Exercise 15.5.a:** Assume $\mathsf{coNP} \subseteq \mathsf{AM}$. It suffices to prove $\Sigma_2\mathsf{P} \subseteq \Pi_2\mathsf{P}$. Suppose $A \in \Sigma_2\mathsf{P}$ by a 2-witness verifier $V$ with time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$. Let $B = \textsc{Final} \; \forall \; \textsc{of} \; V \in \mathsf{coNP}$. We have $B \in \mathsf{AM}$ by a randomized verifier $V'$, which we assume has been amplified to have completeness $2/3$ and soundness $\varepsilon = 2^{-TW}/3$. Then $A \in \mathsf{AM}$ by a randomized verifier $V''$ where $V''(x; r; w_1, w')$ runs $V'(x, w_1; r; w')$:

$$\begin{aligned}
A(x) = 1 \; &\Rightarrow \; \exists w_1 : \; B(x, w_1) = 1 \\
&\Rightarrow \; \exists w_1 : \; \mathbf{Pr}_r\Big[\exists w' : \; V'(x, w_1; r; w') \text{ accepts}\Big] \geq 2/3 \\
&\Rightarrow \; \mathbf{Pr}_r\Big[\exists w_1 \; \exists w' : \; V'(x, w_1; r; w') \text{ accepts}\Big] \geq 2/3 \\
&\Rightarrow \; \mathbf{Pr}_r\Big[\exists w_1, w' : \; V''(x; r; w_1, w') \text{ accepts}\Big] \geq 2/3
\end{aligned}$$

$$\begin{aligned}
A(x) = 0 \; &\Rightarrow \; \forall w_1 : \; B(x, w_1) = 0 \\
&\Rightarrow \; \forall w_1 : \; \mathbf{Pr}_r\Big[\exists w' : \; V'(x, w_1; r; w') \text{ accepts}\Big] \leq \varepsilon \\
&\Rightarrow \; \mathbf{Pr}_r\Big[\exists w_1 \; \exists w' : \; V'(x, w_1; r; w') \text{ accepts}\Big] \leq 2^{TW} \cdot \varepsilon = 1/3 \qquad \text{(union bound)} \\
&\Rightarrow \; \mathbf{Pr}_r\Big[\exists w_1, w' : \; V''(x; r; w_1, w') \text{ accepts}\Big] \leq 1/3
\end{aligned}$$

Thus $\Sigma_2\mathsf{P} \subseteq \mathsf{AM} \subseteq \Pi_2\mathsf{P}$ (Theorem 15.7).

**Exercise 15.5.b:** If Graph Isomorphism is NP-complete, then Graph Nonisomorphism is coNP-complete and therefore $\mathsf{coNP} \subseteq \mathsf{AM}$ since Graph Nonisomorphism $\in \mathsf{AM}$ (Theorem 15.2), which implies $\Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$ (Exercise 15.5.a) and therefore the polynomial hierarchy collapses (Theorem 4.23).

**Exercise 15.6:** This is essentially the same as the proof that if NP $\subseteq$ P/poly then $\Sigma_2\mathsf{P} = \Pi_2\mathsf{P}$ (Theorem 5.5) but with Arthur in place of $\forall$ and Merlin in place of $\exists$.

Assume NP $\subseteq$ P/poly. Since SAT $\in$ NP $\subseteq$ P/poly and SAT SEARCH $\leq_p^o$ SAT (Theorem 2.14), SAT SEARCH has a poly-time program $\Pi$ with advice $a_1, a_2, \ldots$ (Lemma 5.4). For every satisfiable CNF $\varphi$ of size $M$, $\Pi(\varphi; a_M)$ outputs an assignment that satisfies $\varphi$. We may assume $\Pi$ runs in poly time even with wrong advice.

Since MA $\subseteq$ AM (Theorem 15.4), it suffices to prove AM $\subseteq$ MA. Suppose $A \in$ AM by a randomized verifier $V$. Let $B =$ MERLIN OF $V$. We have $B \in$ NP by a verifier $U$ where $U(x, r; w)$ runs $V(x; r; w)$. Since SAT is NP-complete (Theorem 2.10), $B \leq_p^m$ SAT by a reduction that maps $(x, r)$ to a CNF $\varphi_{x,r}$ such that $B(x, r) = 1$ iff $\varphi_{x,r}$ is satisfiable. We may assume $\varphi_{x,r}$ has the same size $M$ for all $r$. We claim that $A \in$ MA by this poly-time randomized verifier $V'(x; b; r)$:

> run the mapping reduction to get $\varphi_{x,r}$
> run $\Pi(\varphi_{x,r}; b)$ to get an assignment $y$
> if $y$ satisfies $\varphi_{x,r}$: accept
> else: reject

Completeness: For all $x, r$, we have

$$B(x,r) = 1 \;\Rightarrow\; \big(\varphi_{x,r} \text{ is satisfiable}\big) \;\Rightarrow\; \big(\Pi(\varphi_{x,r}; a_M) \text{ satisfies } \varphi_{x,r}\big) \;\Rightarrow\; \big(V'(x; a_M; r) \text{ accepts}\big)$$

and thus:

$$A(x) = 1 \;\Rightarrow\; \big(\mathbf{Pr}_r\big[B(x,r) = 1\big] \geq 2/3\big) \;\Rightarrow\; \big(\exists b : \mathbf{Pr}_r\big[V'(x; b; r) \text{ accepts}\big] \geq 2/3\big)$$

Soundness: For all $x, r, b$, we have

$$\big(V'(x; b; r) \text{ accepts}\big) \;\Rightarrow\; \big(\Pi(\varphi_{x,r}; b) \text{ satisfies } \varphi_{x,r}\big) \;\Rightarrow\; \big(\varphi_{x,r} \text{ is satisfiable}\big) \;\Rightarrow\; B(x,r) = 1$$

and thus:

$$A(x) = 0 \;\Rightarrow\; \big(\mathbf{Pr}_r\big[B(x,r) = 1\big] \leq 1/3\big) \;\Rightarrow\; \big(\forall b : \mathbf{Pr}_r\big[V'(x; b; r) \text{ accepts}\big] \leq 1/3\big)$$

**Exercise 15.7.a:** Suppose $A \in \mathsf{MA}$ by a randomized verifier $V$ with time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$, so a witness is $w \in (\{0,1\}^W)^T$. By amplification, $A$ has a randomized verifier $V'$ with the same witness size and with completeness $2/3$ and soundness $\varepsilon = 2^{-TW}/3$. Then $A \in \mathsf{SBP}$ by a randomized program $\Pi$ where $\Pi(x; w, r)$ runs $V'(x; w; r)$:

$$\begin{aligned} A(x) = 1 \;&\Rightarrow\; \exists w : \; \mathbf{Pr}_r\big[V'(x;w;r) \text{ accepts}\big] \geq 2/3 \\ &\Rightarrow\; \mathbf{Pr}_{w,r}\big[V'(x;w;r) \text{ accepts}\big] \geq 2\varepsilon \\ &\Rightarrow\; \mathbf{Pr}_{w,r}\big[\;\Pi(x;w,r) \text{ accepts}\big] \geq 2\varepsilon \end{aligned}$$

$$\begin{aligned} A(x) = 0 \;&\Rightarrow\; \forall w : \; \mathbf{Pr}_r\big[V'(x;w;r) \text{ accepts}\big] \leq \varepsilon \\ &\Rightarrow\; \mathbf{Pr}_{w,r}\big[V'(x;w;r) \text{ accepts}\big] \leq \varepsilon \\ &\Rightarrow\; \mathbf{Pr}_{w,r}\big[\;\Pi(x;w,r) \text{ accepts}\big] \leq \varepsilon \end{aligned}$$

**Exercise 15.7.b:** Suppose $A \in \mathsf{SBP}$ by a randomized program $\Pi$ and a function $\varepsilon(N)$. Say $\Pi$ has time efficiency $\leq T = cN^d$ (for integers $c, d$) and log-bounded word size $W$, so the randomness is $s \in (\{0,1\}^W)^T$. For each valid input $x$ of size $N$, define $S_x = \{s : \Pi(x;s) \text{ accepts}\}$, and define $k = \varepsilon 2^{TW}$:

$$A(x) = 1 \Rightarrow |S_x| \geq 2k$$
$$A(x) = 0 \Rightarrow |S_x| \leq k$$

Let $m = \lceil \log(8k) \rceil$, so $8k \leq 2^m < 16k$ and thus $1/8 \geq k/2^m > 1/16$. Let $h\colon \{0,1\}^{2m+TW-1} \times (\{0,1\}^W)^T \to \{0,1\}^m$ be the pairwise uniform hash function from Lemma 8.9. (Or, we could use Lemma 8.7.) Then $A \in \mathsf{AM}$ by the following randomized verifier $V$ with completeness $1.5k/2^m$ and soundness $k/2^m$ (which can be amplified to $2/3$ and $1/3$ as in the proof of Theorem 15.2):

> $V(x;r;s)$:
> accept iff $\Pi(x;s)$ accepts and $h_r(s) = 0^m$

$$\begin{aligned}
A(x) = 1 &\Rightarrow |S_x| \geq 2k \\
&\Rightarrow \mathbf{Pr}_r\left[\exists s \in S_x : h_r(s) = 0^m\right] \geq \tfrac{2k}{2^m}\left(1 - \tfrac{2k}{2^m}\right) \geq 1.5k/2^m && \text{(Lemma 8.13)} \\
&\Rightarrow \mathbf{Pr}_r\left[\exists s : V(x;r;s) \text{ accepts}\right] \geq 1.5k/2^m
\end{aligned}$$

$$\begin{aligned}
A(x) = 0 &\Rightarrow |S_x| \leq k \\
&\Rightarrow \mathbf{Pr}_r\left[\exists s \in S_x : h_r(s) = 0^m\right] \leq k/2^m && \text{(Lemma 8.13)} \\
&\Rightarrow \mathbf{Pr}_r\left[\exists s : V(x;r;s) \text{ accepts}\right] \leq k/2^m
\end{aligned}$$

**Exercise 15.8:** Let $\Pi$ be the hypothesized reduction. Assume $B \in \mathsf{NP}$ by a verifier $V$. Then $A \in \mathsf{AM/poly}$ by the following nonuniform randomized verifier $V'$: Define the advice $a_N = \mathbf{Pr}_y\big[B(y)=1\big]$ where $y$ is distributed as the random query made by $\Pi$ on inputs $x$ of size $N$.

$V'(x;r;w;a_N)$:
  view $r$ as $(r_1,\ldots,r_k)$ where $k=717$ and each $r_i$ is randomness for $\Pi$
  run $\Pi(x;r_1),\ldots,\Pi(x;r_k)$ to get queries $y_1,\ldots,y_k$
  view $w$ as:
    bits $b_1,\ldots,b_k$ (Merlin claims $b_i = B(y_i)$ for all $i$)
    for each $i$ such that $b_i=1$: $w$ also includes $w_i$ (a purported witness for $B(y_i)=1$)
  accept iff $b_i=1$ for $\geq (a_N - 1/20)k$ many $i$
    and $V(y_i;w_i)$ accepts for each $i$ such that $b_i=1$
    and $> k/2$ of $\Pi(x;r_1),\ldots,\Pi(x;r_k)$ accept after receiving oracle answers $b_1,\ldots,b_k$

Imagine $k$ tosses of a coin with heads probability $a_N$, where the $i^{\text{th}}$ toss is heads iff $B(y_i)=1$. By Lemma 7.20:

$$\mathbf{Pr}_r\big[(\text{number of } i \text{ such that } B(y_i)=1) < (a_N - 1/20)k\big] \leq \mathrm{e}^{-(1/20)^2 k} \leq 1/6$$
$$\mathbf{Pr}_r\big[(\text{number of } i \text{ such that } B(y_i)=1) > (a_N + 1/20)k\big] \leq \mathrm{e}^{-(1/20)^2 k} \leq 1/6$$

Now, imagine $k$ tosses of a different coin, where the $i^{\text{th}}$ toss is heads iff $\Pi(x;r_i) \neq A(x)$ after receiving the correct oracle answer $B(y_i)$. The heads probability $p_x \leq 1/3$ doesn't depend on $i$. By Lemma 7.20:

$$\mathbf{Pr}_r\big[(\text{number of } i \text{ such that } \Pi(x;r_i) \neq A(x) \text{ after receiving the correct oracle answer } B(y_i)) > (1/3+1/20)k\big]$$
$$\leq \mathrm{e}^{-(1/20)^2 k} \leq 1/6$$

Completeness: Assume $A(x)=1$. For each $r$, define $w$ (depending on $r$) as follows: For each $i$, let $b_i = B(y_i)$, and for each $i$ such that $b_i = 1$, let $w_i$ be such that $V(y_i;w_i)$ accepts. By a union bound:

$$\begin{aligned}\mathbf{Pr}_r\big[V'(x;r;w;a_N) \text{ rejects}\big] &\leq \mathbf{Pr}_r\big[b_i = 1 \text{ for } < (a_N - 1/20)k \text{ many } i\big] + \\ &\quad \mathbf{Pr}_r\big[\leq k/2 \text{ of } \Pi(x;r_1),\ldots,\Pi(x;r_k) \text{ accept} \\ &\qquad \text{after receiving oracle answers } b_1,\ldots,b_k\big] \\ &\leq 1/6 + 1/6 = 1/3\end{aligned}$$

Soundness: Assume $A(x)=0$. Consider any $r,w$ such that $V'(x;r;w;a_N)$ accepts. For each $i$, if $b_i=1$ then $V(y_i;w_i)$ accepts (since $V'$ accepts) and thus $B(y_i)=1$. This means if $b_i \neq B(y_i)$ then $b_i = 0$ and $B(y_i)=1$. Assume for the moment that $B(y_i)=1$ for $\leq (a_N+1/20)k$ many $i$. Then since $b_i=1$ for $\geq (a_N - 1/20)k$ many $i$ (since $V'$ accepts), we have $b_i \neq B(y_i)$ for $\leq (a_N+1/20)k - (a_N - 1/20)k = k/10$ many $i$. Then since $> k/2$ of $\Pi(x;r_1),\ldots,\Pi(x;r_k)$ accept after receiving oracle answers $b_1,\ldots,b_k$ (since $V'$ accepts), it follows that $> k/2 - k/10 >$

$(1/3 + 1/20)k$ of $\Pi(x; r_1), \ldots, \Pi(x; r_k)$ accept after receiving oracle answers $B(y_1), \ldots, B(y_k)$. By a union bound:

$$\begin{aligned}\mathbf{Pr}_r\big[\,\exists w : V'(x; r; w; a_N) \text{ accepts}\big] \;\leq\; & \mathbf{Pr}_r\big[B(y_i) = 1 \text{ for } > (a_N + 1/20)k \text{ many } i\big] + \\ & \mathbf{Pr}_r\big[> (1/3 + 1/20)k \text{ of } \Pi(x; r_1), \ldots, \Pi(x; r_k) \text{ accept} \\ & \qquad \text{after receiving oracle answers } B(y_1), \ldots, B(y_k)\big] \\ \leq\; & 1/6 + 1/6 \;=\; 1/3\end{aligned}$$

**Exercise 15.9:** Suppose $A$ has a poly-time private-coin interactive proof system with completeness $> 0$ and soundness 0. Then $A \in \mathsf{NP}$ by a verifier $V$ that views the purported witness $w$ as Arthur's randomness and a sequence of messages from Merlin, where $V(x;w)$ accepts iff Arthur would accept.

$$\begin{aligned} A(x)=1 &\Rightarrow \exists \text{ Merlin strategy}: \mathbf{Pr}_r[\text{Arthur accepts}] > 0 && \text{(completeness} > 0\text{)} \\ &\Rightarrow \exists w: V(x;w) \text{ accepts} \end{aligned}$$

$$\begin{aligned} A(x)=0 &\Rightarrow \forall \text{ Merlin strategy}: \mathbf{Pr}_r[\text{Arthur accepts}] = 0 && \text{(soundness 0)} \\ &\Rightarrow \forall w: V(x;w) \text{ rejects} \end{aligned}$$

**Exercise 15.10:** Arithmetization gives the polynomial:

$$P(x_1, x_2, x_3, x_4) = \big(1 - x_1(1-x_2)x_4\big) \cdot \big(1 - x_1 x_3 (1 - x_4)\big)$$

Arthur lets $b_0 = 12$.

- Merlin sends the coefficients of:

$$\begin{aligned}
Q_1(x_1) &= \textstyle\sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \sum_{x_4 \in \{0,1\}} P(x_1, x_2, x_3, x_4) \\
&= \big(1 - x_1 \cdot (1-0) \cdot 0\big) \cdot \big(1 - x_1 \cdot 0 \cdot (1-0)\big) + \\
&\quad \big(1 - x_1 \cdot (1-0) \cdot 1\big) \cdot \big(1 - x_1 \cdot 0 \cdot (1-1)\big) + \\
&\quad \big(1 - x_1 \cdot (1-0) \cdot 0\big) \cdot \big(1 - x_1 \cdot 1 \cdot (1-0)\big) + \\
&\quad \big(1 - x_1 \cdot (1-0) \cdot 1\big) \cdot \big(1 - x_1 \cdot 1 \cdot (1-1)\big) + \\
&\quad \big(1 - x_1 \cdot (1-1) \cdot 0\big) \cdot \big(1 - x_1 \cdot 0 \cdot (1-0)\big) + \\
&\quad \big(1 - x_1 \cdot (1-1) \cdot 1\big) \cdot \big(1 - x_1 \cdot 0 \cdot (1-1)\big) + \\
&\quad \big(1 - x_1 \cdot (1-1) \cdot 0\big) \cdot \big(1 - x_1 \cdot 1 \cdot (1-0)\big) + \\
&\quad \big(1 - x_1 \cdot (1-1) \cdot 1\big) \cdot \big(1 - x_1 \cdot 1 \cdot (1-1)\big) \\
&= 1 + (1 - x_1) + (1 - x_1) + (1 - x_1) + 1 + 1 + (1 - x_1) + 1 \\
&= 25x_1 + 8
\end{aligned}$$

  Arthur doesn't reject now, since $Q_1(0) + Q_1(1) = 8 + 4 = 12 = b_0$. Arthur samples $a_1 = 4$ and lets $b_1 = Q_1(a_1) = 21$.

- Merlin sends the coefficients of:

$$\begin{aligned}
Q_2(x_2) &= \textstyle\sum_{x_3 \in \{0,1\}} \sum_{x_4 \in \{0,1\}} P(a_1, x_2, x_3, x_4) \\
&= \big(1 - 4 \cdot (1-x_2) \cdot 0\big) \cdot \big(1 - 4 \cdot 0 \cdot (1-0)\big) + \\
&\quad \big(1 - 4 \cdot (1-x_2) \cdot 1\big) \cdot \big(1 - 4 \cdot 0 \cdot (1-1)\big) + \\
&\quad \big(1 - 4 \cdot (1-x_2) \cdot 0\big) \cdot \big(1 - 4 \cdot 1 \cdot (1-0)\big) + \\
&\quad \big(1 - 4 \cdot (1-x_2) \cdot 1\big) \cdot \big(1 - 4 \cdot 1 \cdot (1-1)\big) \\
&= 1 + (1 - 4 + 4x_2) + (1 - 4) + (1 - 4 + 4x_2) \\
&= 8x_2 + 21
\end{aligned}$$

  Arthur doesn't reject now, since $Q_2(0) + Q_2(1) = 21 + 0 = 21 = b_1$. Arthur samples $a_2 = 18$ and lets $b_2 = Q_2(a_2) = 20$.

- Merlin sends the coefficients of:

$$\begin{aligned}
Q_3(x_3) &= \textstyle\sum_{x_4 \in \{0,1\}} P(a_1, a_2, x_3, x_4) \\
&= \big(1 - 4 \cdot (1-18) \cdot 0\big) \cdot \big(1 - 4 \cdot x_3 \cdot (1-0)\big) + \\
&\quad \big(1 - 4 \cdot (1-18) \cdot 1\big) \cdot \big(1 - 4 \cdot x_3 \cdot (1-1)\big) \\
&= (1 - 4x_3) + 11 \\
&= 25x_3 + 12
\end{aligned}$$

Arthur doesn't reject now, since $Q_3(0)+Q_3(1) = 12+8 = 20 = b_2$. Arthur samples $a_3 = 11$ and lets $b_3 = Q_3(a_3) = 26$.

- Merlin sends the coefficients of:

$$\begin{aligned} Q_4(x_4) &= P(a_1, a_2, a_3, x_4) \\ &= \big(1-4\cdot(1-18)\cdot x_4\big)\cdot\big(1-4\cdot 11\cdot(1-x_4)\big) \\ &= (10x_4+1)\cdot(15x_4+15) \\ &= 5x_4^2+20x_4+15 \end{aligned}$$

Arthur doesn't reject now, since $Q_4(0)+Q_4(1) = 15+11 = 26 = b_3$. Arthur samples $a_4 = 2$ and lets $b_4 = Q_4(a_4) = 17$.

Arthur accepts since $P(4, 18, 11, 2) = 17 = b_4$.

**Exercise 15.11.a:** Suppose $Q(x) = c_m x^m + \cdots + c_2 x^2 + c_1 x + c_0$ and $Q'(x) = (c_m + \cdots + c_2 + c_1)x + c_0$. Since $c_0 = Q(0)$ and $c_m + \cdots + c_1 + c_0 = Q(1)$, we have $Q'(x) = \big(Q(1) - Q(0)\big)x + Q(0) = (1-x) \cdot Q(0) + x \cdot Q(1)$.

Another way to show it: $Q'(x)$ and $(1-x) \cdot Q(0) + x \cdot Q(1)$ have degree $\leq 1$ and agree on the two points 0 and 1, so they're identical (Corollary 0.21).

**Exercise 15.11.b:** The polynomials $P_{i,j}(x_1, \ldots, x_i)$ for $n \geq i \geq j \geq 0$ are defined by:

- $P_{n,n} = P$.
- If $i < n$ then $P_{i,i}(x_1, \ldots, x_i) = 1 - P_{i+1,0}(x_1, \ldots, x_i, 0) \cdot P_{i+1,0}(x_1, \ldots, x_i, 1)$.
- If $i > j$ then $P_{i,j} = P_{i,j+1}$ with $x_{j+1}$ linearized.

This recursive subroutine maintains the invariant that $\text{eval}(i, j, a_1, \ldots, a_i) = P_{i,j}(a_1, \ldots, a_i)$ by Exercise 15.11.a:

```
eval(i, j, a_1, ..., a_i):
    if i = j = n: return P(a_1, ..., a_n)
    if i = j: return 1 − eval(i + 1, 0, a_1, ..., a_i, 0) · eval(i + 1, 0, a_1, ..., a_i, 1)
    return (1 − a_{j+1}) · eval(i, j + 1, a_1, ..., a_j, 0, a_{j+2}, ..., a_i) +
                a_{j+1}  · eval(i, j + 1, a_1, ..., a_j, 1, a_{j+2}, ..., a_i)
```

This has recursion depth $\leq n^2$, and each stack frame uses $\text{poly}N$ space, so the total space is:

$$\big(\text{poly}N \text{ stack frames at a time}\big) \cdot \big(\text{poly}N \text{ bits per stack frame}\big) \;=\; \text{poly}N \text{ bits}$$

**Exercise 15.11.c:** In outer iteration $i$ and inner iteration 0, the honest prover sends the coefficients of $Q_{i,0}(x_i) = P_{i,0}(a_1, \ldots, a_{i-1}, x_i)$ for the current values of $a_1, \ldots, a_{i-1}$ (ignoring the rest of the communication history). Since $Q_{i,0}$ has degree $\leq m$, the prover can recover its coefficients from the values $P_{i,0}(a_1, \ldots, a_{i-1}, x_i)$ for all $x_i \in \{0, \ldots, m\}$, which he obtains by calling $\text{eval}(i, 0, a_1, \ldots, a_{i-1}, x_i)$ for all $x_i \in \{0, \ldots, m\}$.

In outer iteration $i$ and inner iteration $j > 0$, the honest prover sends the coefficients of $Q_{i,j}(x_j) = P_{i,j}(\ldots, a_{j-1}, x_j, a_{j+1}, \ldots)$ for the current values of $a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_i$ (ignoring the rest of the communication history). Since $Q_{i,j}$ has degree $\leq m$, the prover can recover its coefficients from the values $P_{i,j}(\ldots, a_{j-1}, x_j, a_{j+1}, \ldots)$ for all $x_j \in \{0, \ldots, m\}$, which he obtains by calling $\text{eval}(i, j, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots)$ for all $x_j \in \{0, \ldots, m\}$.

**Exercise 15.12:** Suppose EXP $\subseteq$ P/poly. Then EXP $\subseteq \Sigma_2$P (Exercise 5.7) and $\Sigma_2$P $\subseteq$ PSPACE (Lemma 4.20), so EXP = PSPACE. In that case, "EXP $\subseteq$ P/poly implies EXP = MA" follows from "PSPACE $\subseteq$ P/poly implies PSPACE = MA" (Corollary 15.19).

**Exercise 15.13.a:** The reduction changes the number of edges from $m$ to $m' = \binom{n}{2} - m$ and thus changes the input size from $N \approx n + m$ to $N' \approx n + m'$. This yields a correct interactive proof system for #Anti-Triangle, and the honest prover is poly-time, but the verifier runs in time quasi-linear in $N'$ (which could be $\approx N^2$), not quasi-linear in $N$.

**Exercise 15.13.b:** This is like the doubly efficient interactive proof system for #Triangle (Theorem 15.20). First, assume $n$ is a power of 2. Redefine $S(x,y)$ to be:

$$S(x,y) \;=\; 1 - \sum_{(u,v) \in V^2 \,:\, u=v \text{ or } \{u,v\} \in E} R_u(x) \cdot R_v(y)$$

For every boolean assignment to $x,y$, we have $S(x,y) = 1$ if $x \neq y$ and $\{x,y\} \notin E$, and $S(x,y) = 0$ otherwise. With this $S$, define:

$$T(x,y,z) \;=\; S(x,y) \cdot S(x,z) \cdot S(y,z)$$

For every boolean assignment to $x,y,z$, we have $T(x,y,z) = 1$ if $x,y,z$ are distinct and $\{x,y,z\}$ is an anti-triangle in $G$, and $T(x,y,z) = 0$ otherwise. There are $\frac{1}{6}\sum_{(x,y,z) \in (\{0,1\}^{\log n})^3} T(x,y,z)$ anti-triangles in $G$ since $\sum_{(x,y,z) \in (\{0,1\}^{\log n})^3} T(x,y,z)$ counts each anti-triangle $3! = 6$ times (once for each permutation of the three nodes). The doubly efficient interactive proof system for #Anti-Triangle uses the sum-check protocol with $T$, starting with $b_0 \leftarrow 6k$.

If $n$ isn't a power of 2, we add some $\ell < n$ degree-0 nodes so the total number of nodes $n+\ell$ is a power of 2. This increases the number of anti-triangles by exactly $\left(\binom{n}{2} - m\right) \cdot \ell + n \cdot \binom{\ell}{2} + \binom{\ell}{3}$ since $\left(\binom{n}{2} - m\right) \cdot \ell$ anti-triangles have one new node, $n \cdot \binom{\ell}{2}$ anti-triangles have two new nodes, and $\binom{\ell}{3}$ anti-triangles have three new nodes. Thus the original graph has exactly $k$ anti-triangles iff the new graph has exactly $k + \left(\binom{n}{2} - m\right) \cdot \ell + n \cdot \binom{\ell}{2} + \binom{\ell}{3}$ anti-triangles.

**Exercise 15.14:** A poly-time $(c,s)$-checker for $A$ on input $x$ can map $x$ to $x'$ (using $A \leq_p^m A'$) and run the poly-time $(c,s)$-checker for $A'$ on input $x'$. Whenever the latter queries $B'(y')$, the former instead maps $y'$ to $y$ (using $A' \leq_p^m A$) and queries $B(y)$, unless $y' = x'$, in which case the former queries $B(x)$ (noting that possibly $y \neq x$).

Completeness: If $B = A$ then the checker for $A(x)$ simulates the checker for $A'(x')$ with $B' = A'$ since $B'(y') = B(y) = A(y) = A'(y')$ for $y' \neq x'$ and $B'(x') = B(x) = A(x) = A'(x')$, and thus accepts with probability $\geq c$.

Soundness: If $B(x) \neq A(x)$ then the checker for $A(x)$ simulates the checker for $A'(x')$ with $B'$ such that $B'(x') \neq A'(x')$, and thus accepts with probability $\leq s$.

**Exercise 15.15.a:** Since we know Consistent Graph Isomorphism $\leq_p^m$ Graph Isomorphism (Exercise 15.2.a), it suffices to show Graph Isomorphism Search $\leq_p^o$ Consistent Graph Isomorphism:

on input $(G_1, G_2)$ where $G_1$ and $G_2$ have nodes $[n]$:
    initialize $p : [n] \to [n]$ as the partial permutation that's undefined everywhere
    for $v \leftarrow 1, \ldots, n$:
        for each $w \in [n] \setminus \{p(1), \ldots, p(v-1)\}$:
            $p(v) \leftarrow w$
            query the oracle on $(G_1, G_2, p)$
            if it accepted: break out of the $w$ loop, so $p(v) = w$ becomes frozen

This maintains the outer loop invariant that there exists an isomorphism from $G_1$ to $G_2$ that's consistent with $p$.

**Exercise 15.15.b:** Let $A = \text{Graph Isomorphism}$. By Lemma 15.22, it suffices to show that $A$ and $\overline{A}$ have poly-time private-coin interactive proof systems with completeness 1 and soundness $1/3$ where the honest provers are computed by poly-time oracle reductions to $A$.

For $A$, we just use $A \in \mathsf{NP}$ where a witness is an isomorphism. By Exercise 15.15.a, the honest prover is computed by a poly-time oracle reduction to $A$.

For $\overline{A}$, we use the private-coin Arthur–Merlin proof system from Theorem 15.1. The honest prover is sent a graph $H$ that's isomorphic to one of $G_1$ or $G_2$, and he must determine which one. He can make the oracle query $(H, G_1)$ to $A$ to determine whether $H \cong G_1$ holds (and if not, then $H \cong G_2$ must hold).

**Exercise 15.16:** $\Leftarrow$: This holds by the same argument as Lemma 15.22 for single-prover interactive proof systems.

$\Rightarrow$: Assume $A$ has a poly-time $(2/3, 1/3)$-checker. We turn this into a poly-time PCP verifier for $A$ that, on input $x$, runs the checker using the purported witness to answer queries to the oracle $B$, and accepts iff the checker accepts and $B(x) = 1$. This has completeness $2/3$ because if $A(x) = 1$ and the witness $B$ is $A$ (more precisely, $A$'s truth table for relevant input sizes), then the PCP verifier accepts with probability $\geq 2/3$. This has soundness $1/3$ because if $A(x) = 0$ and $B(x) = 0$ then the PCP verifier accepts with probability 0, and if $A(x) = 0$ and $B(x) = 1$ then the PCP verifier accepts with probability $\leq 1/3$. By the proof of Lemma 15.16, $A$ has a poly-time two-prover interactive proof system with completeness $2/3$ and soundness $1/3$ where the honest provers simply answer queries to $A$.

Furthermore, $\overline{A}$ has a poly-time $(2/3, 1/3)$-checker that's the same as the one for $A$ except it negates each answer it receives from the oracle. Thus, $\overline{A}$ also has a poly-time two-prover interactive proof system with completeness $2/3$ and soundness $1/3$ where the honest provers are computed by poly-time oracle reductions to $\overline{A}$ and thus to $A$ (by negating the oracle's answers).

**Exercise 15.17:** Consider the dishonest verifier that sends a uniformly random pair of nodes $\{u, v\}$ (not necessarily an edge). Assuming $G$ has a proper 3-coloring $c$, the simulator can sample a uniformly random pair of nodes $\{u, v\}$ along with the bit indicating whether $c(u) = c(v)$, because the messages the honest prover puts in the boxes labeled $u$ and $v$ would be the same color as each other if $c(u) = c(v)$, and different colors if $c(u) \neq c(v)$. We design a poly-time randomized algorithm that uses this simulator to find a proper 3-coloring of $G$ (assuming one exists), which implies Graph 3-Coloring $\in$ RP and thus NP = RP since Graph 3-Coloring is NP-complete. Running the simulator $2n^2 \log n$ times where $n$ is the number of nodes:

$$
\begin{aligned}
& \mathbf{Pr}\big[\exists\, \{u, v\} \text{ that's sampled by no run of the simulator}\big] \\
\leq\ & \textstyle\sum_{\{u,v\}} \mathbf{Pr}\big[\forall i:\ \text{the } i^{\text{th}} \text{ run of the simulator doesn't sample } \{u, v\}\big] && \text{(union bound)} \\
=\ & \textstyle\sum_{\{u,v\}} \prod_i \mathbf{Pr}\big[\text{the } i^{\text{th}} \text{ run of the simulator doesn't sample } \{u, v\}\big] \\
=\ & \tbinom{n}{2}\big(1 - 1/\tbinom{n}{2}\big)^{2n^2 \log n} \\
\leq\ & \tbinom{n}{2} \mathrm{e}^{-(2n^2 \log n)/\binom{n}{2}} \\
\leq\ & \tbinom{n}{2} \mathrm{e}^{-4 \log n} \\
\leq\ & 1/3
\end{aligned}
$$

Thus with probability $\geq 2/3$, every pair $\{u, v\}$ is sampled by at least one run of the simulator, in which case we know whether $c(u) = c(v)$ for each pair $\{u, v\}$. Assign red to node 1 and to all nodes $u$ such that $c(u) = c(1)$. Assign green to some node $v$ such that $c(v) \neq c(1)$ and to all nodes $u$ such that $c(u) = c(v)$. Assign blue to all remaining nodes. This proper 3-coloring is $c$ but with the colors possibly permuted.

**Exercise 15.18:** Completeness 1: Suppose $G$ has a full cycle $C$, and consider the honest verifier's interaction with the honest prover. If $b = 0$ then the honest verifier accepts upon confirming that $p$ is a permutation and the unlocked boxes form $p(G)$. If $b = 1$ then the honest verifier accepts upon confirming that the boxes corresponding to $p(C)$ form a full cycle in $p(G)$.

Soundness $1/2$: Suppose $G$ has no full cycle, and consider the honest verifier's interaction with an arbitrary prover. Whatever messages the prover puts in the boxes, they either don't form a graph isomorphic to $G$ (in which case the honest verifier rejects if $b = 0$, by the binding property) or don't contain a full cycle (in which case the honest verifier rejects if $b = 1$, by the binding property). Either way, the honest verifier accepts with probability $\leq 1/2$.

Zero-knowledge: Suppose $G$ has a full cycle $C$, and consider an arbitrary poly-time verifier's interaction with the honest prover. The verifier's message $b$ doesn't depend on $p$, by the hiding property. If $b = 0$ then the verifier's view is $p$ and $p(G)$, which is trivial to simulate. If $b = 1$ then the verifier's view is a uniformly random full cycle on nodes $[n]$, which is trivial to simulate by sampling a uniformly random permutation $q\colon [n] \to [n]$ and outputting the full cycle $q(1) — q(2) — q(3) — \cdots — q(n) — q(1)$.

**Exercise 16.1:** Let $a, b \in \{0, 1, *\}^n$ denote partial assignments to $x, y$. This adversary strategy assigns $a_i = b_i = 1$ if $x_i$ is queried before $y_i$, and assigns $a_i = b_i = 0$ if $y_i$ is queried before $x_i$:

repeat:
    if $x_i$ is queried and $y_i$ wasn't previously queried: respond 1
    if $y_i$ is queried and $x_i$ was previously queried: respond 1
    if $y_i$ is queried and $x_i$ wasn't previously queried: respond 0
    if $x_i$ is queried and $y_i$ was previously queried: respond 0

Right before the $(2n)^{\text{th}}$ query, $a_i = b_i$ for all but one $i$, and for the remaining $i$, either $a_i = 1$ and $b_i = *$ (in which case $a > b$ if $b_i = 0$, and $a \not> b$ if $b_i = 1$) or $b_i = 0$ and $a_i = *$ (in which case $a > b$ if $a_i = 1$, and $a \not> b$ if $a_i = 0$).

**Exercise 16.2:** Since $f$ is nonconstant and symmetric, there exist $k \in [n]$ and $b \in \{0,1\}$ such that $f(x) = b$ if $\text{weight}(x) = k$, and $f(x) = \overline{b}$ if $\text{weight}(x) = k-1$. The adversary responds 1 to the first $k-1$ queries and 0 to subsequent queries. After $N-1$ queries in total, the remaining unqueried variable could be assigned 1 to get a $b$-input, or 0 to get a $\overline{b}$-input, so $f|_a$ is nonconstant.

**Exercise 16.3:** An adversary strategy for $f$ can maintain the invariant that $f|_a$ has an odd number of 1-inputs: Assume this holds at the beginning of a round in which $x_i$ is queried. Let $a'$ be $a$ with $a'_i = 0$, and $a''$ be $a$ with $a''_i = 1$. The inputs consistent with $a$ are partitioned into inputs consistent with $a'$ and inputs consistent with $a''$, so

$$\text{number of 1-inputs of } f|_a \;=\; (\text{number of 1-inputs of } f|_{a'}) + (\text{number of 1-inputs of } f|_{a''})$$

and thus either $f|_{a'}$ or $f|_{a''}$ has an odd number of 1-inputs, and the adversary assigns $x_i$ accordingly to maintain the invariant. Until $a$ is a full assignment, an even number of inputs are consistent with $a$ and thus $f|_a$ is nonconstant.

An alternative proof is by the contrapositive: Suppose $f$ has a decision tree of depth $< N$. Partition the set of all inputs according to which leaf they lead to. For a root-to-leaf path of length $k < N$, the number of inputs leading to the leaf is $2^{N-k}$, which is even, and either all these inputs are 0-inputs or all these inputs are 1-inputs. Thus

$$\text{number of 1-inputs of } f \;=\; \sum_{\text{leaves that output 1}} (\text{number of inputs leading to that leaf})$$

which is a sum of even numbers and is therefore even.

**Exercise 16.4.a:** Imagine $x_0 = 1$ is prepended to the input and $x_{N+1} = 0$ is appended to the input. These phantom input bits don't affect whether $x$ contains 01.

query $x_2x_4x_6\cdots x_{N-1}$ and accept if it contains 01
find a $j \in \{1,3,5,\ldots,N\}$ such that $x_{j-1} = 1$ and $x_{j+1} = 0$
query $x_1x_3\cdots x_{j-2}$ and accept if $x_1x_2\cdots x_{j-1}$ contains 01
query $x_{j+2}x_{j+4}\cdots x_N$ and accept if $x_{j+1}x_{j+2}\cdots x_N$ contains 01
reject

If this accepts on the first line, then $F_N(x) = 1$ since for some odd $i$, $x_{i-1}x_{i+1} = 01$ and thus $x_{i-1}x_ix_{i+1} \in \{001, 011\}$, both of which contain 01. If this reaches the second line, then there indeed exists such a $j$ since $x_0x_2x_4\cdots x_{N-1}x_{N+1}$ is 1s followed by 0s (and in this case, $x_j$ never gets queried). If this accepts on the third or fourth lines, then of course $F_N(x) = 1$. If this rejects on the last line, then $F_N(x) = 0$ since $x_1x_2\cdots x_{j-1}x_jx_{j+1}x_{j+2}\cdots x_N = 11\cdots 1x_j00\cdots 0$ is 1s followed by 0s, regardless of $x_j$'s value. In all cases, at most $N-1$ input bits are queried.

**Exercise 16.4.b:** $F_N$ has $N+1$ many 0-inputs, namely $000\cdots00$, $100\cdots00$, $110\cdots00$, …, $111\cdots10$, $111\cdots11$. If $N$ is even then $F_N$ has an odd number $2^N-(N+1)$ of 1-inputs, so $F_N$ is evasive (Exercise 16.3).

Now, assume $N$ is odd. We showed $\mathrm{Det}(F_N) \leq N-1$ (Exercise 16.4.a). We have $\mathrm{Det}(F_N) \geq \mathrm{Det}(F_{N-1}) = N-1$ by a projection reduction (Lemma 16.5) from $F_{N-1}$ to $F_N$ that maps $x_1\cdots x_{N-1}$ to $x_1\cdots x_{N-1}0$.

**Exercise 16.5.a:** Here's a decision tree for $F_N(x_1 \cdots x_N)$ assuming $N = 3k+1$ for an integer $k$:

```
for i ← 1,...,k:
    query x_{3i-1}
    if it's 0:
        query x_{3i} ··· x_N
        if it contains the substring 11: accept
        else: reject
    if it's 1:
        query x_{3i-2} and x_{3i}
        if either is 1: accept
reject
```

If this accepts on the fifth line, then it's correct and didn't query $x_{3i-2}$. If this rejects on the sixth line, then it's correct since $x = 010010010 \cdots 010 x_{3i-2} 0 x_{3i} \cdots x_N$ doesn't contain the substring 11, regardless of the unqueried value of $x_{3i-2}$. If this accepts on the next-to-last line, then it's correct since either $x_{3i-2}x_{3i-1} = 11$ or $x_{3i-1}x_{3i} = 11$, and it didn't query $x_N$. If this rejects on the last line, then it's correct since $x = 010010010 \cdots 010 x_N$ doesn't contain the substring 11, regardless of the unqueried value of $x_N$. In all cases, the decision tree queries $\leq N-1$ bits of $x$.

**Exercise 16.5.b:** Let $w_N$ be the number of 1-inputs of $F_N$ (the weight of its truth table). We have $w_1 = 0$ and $w_2 = 1$ and for each $N > 2$, $w_N = w_{N-1} + w_{N-2} + 2^{N-2}$ since $w_{N-1}$ many 1-inputs have $x_N = 0$, and $w_{N-2}$ many 1-inputs have $x_{N-1}x_N = 01$, and $2^{N-2}$ many 1-inputs have $x_{N-1}x_N = 11$. Let $m_N = w_N \bmod 2$. For each $N > 2$,

$$m_N \;=\; \big(m_{N-1} + m_{N-2} + (2^{N-2} \bmod 2)\big) \bmod 2 \;=\; m_{N-1} \oplus m_{N-2}$$

since $2^{N-2}$ is even.

$m_1 \leftarrow 0$ and $m_2 \leftarrow 1$
for $N \leftarrow 3, 4, 5, \ldots$: $m_N \leftarrow m_{N-1} \oplus m_{N-2}$

We see that $m_1, m_2, m_3, \ldots = 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, \ldots$. Formally, the algorithm maintains the invariant that $m_N = 0$ if $N \bmod 3 = 1$, and $m_N = 1$ if $N \bmod 3 \neq 1$, because $0 \oplus 1 = 1$ and $1 \oplus 1 = 0$ and $1 \oplus 0 = 1$. By Exercise 16.3, $F_N$ is evasive if $N \bmod 3 \neq 1$ since $m_N = 1$ means $F_N$ has an odd number of 1-inputs.

Now, assume $N \bmod 3 = 1$. We showed $\mathrm{Det}(F_N) \leq N - 1$ in Exercise 16.5.a. We have $\mathrm{Det}(F_N) \geq \mathrm{Det}(F_{N-1}) = N - 1$ by a projection reduction (Lemma 16.5) from $F_{N-1}$ to $F_N$ that maps $x_1 \cdots x_{N-1}$ to $x_1 \cdots x_{N-1}0$.

**Exercise 16.6:**

query all possible edges between the first $\lfloor n/2 \rfloor$ nodes and the last $\lceil n/2 \rceil$ nodes
if at least three of these edges are assigned 1 and are incident to a common node $v$:
    query all possible edges incident to $v$ and reject if $\deg(v) \neq n-4$
    for each neighbor $u$ of $v$:
        query all possible edges incident to $u$ and reject if $\deg(u) \neq 1$
    accept
else: reject

If this accepts, then the input graph has a degree-$(n-4)$ node $v$, each of whose neighbors is only adjacent to $v$. Conversely, if the graph has such a node $w$, then we must have $v = w$ since $w$ has $\geq n-4-(\lceil n/2 \rceil - 1) \geq 3$ incident edges between the first $\lfloor n/2 \rfloor$ nodes and the last $\lceil n/2 \rceil$ nodes, and all other nodes have degree $\leq 2$ and thus don't have this property, so the decision tree accepts.

If this accepts, then among $v$'s three nonneighbors, at least two of them are both among the first $\lfloor n/2 \rfloor$ nodes or both among the last $\lceil n/2 \rceil$ nodes, so the edge between them is unqueried. Similarly, if this rejects then not all edges are queried.

**Exercise 16.7:** We design an adversary strategy that maintains the following invariant (for the partial assignment $a$ recording the results of queries so far): The edges $\{u, v\}$ with $a_{\{u,v\}} = 1$ form a forest (every connected component is a tree), and $a_{\{u,v\}} = *$ for all $u$ and $v$ in different connected components.

initialize $a \leftarrow **\cdots*$
repeat:
    the querier picks some $\{u, v\}$
    if $u$ and $v$ are in the same connected component: assign $a_{\{u,v\}} \leftarrow 0$
    else: assign $a_{\{u,v\}} \leftarrow 1$ (thereby merging those two connected components)

Since the invariant holds right before the $N^{\text{th}}$ query $\{u, v\}$, there must be only one connected component (the edges assigned 1 form a tree connecting all nodes) because if there were at least two connected components, then there would be at least two unqueried edges between them (since $n \geq 3$). Assigning $a_{\{u,v\}} \leftarrow 0$ would yield a tree (with no cycles). Assigning $a_{\{u,v\}} \leftarrow 1$ would yield a graph with a cycle consisting of $\{u, v\}$ and a path between $u$ and $v$ of edges previously assigned 1.

**Exercise 16.8.a:** For $N = n(n-1)$, $F_N(g) = \bigvee_{u \in [n]} \bigwedge_{v \in [n] \setminus \{u\}} \overline{g_{(u,v)}}$ is expressed by a read-once formula and is therefore evasive (Theorem 16.1).

**Exercise 16.8.b:** We design an adversary strategy that maintains the following invariant: The edges $\{u, v\}$ with $a_{\{u,v\}} = 1$ form a tree connecting a subset of nodes $S$. For every $\{u, v\} \subseteq S$, we have $a_{\{u,v\}} \neq *$. For every $u \notin S$, there exists $v \in S$ with $a_{\{u,v\}} = *$.

the querier picks some $\{u, v\}$
assign $a_{\{u,v\}} \leftarrow 1$ and let $S \leftarrow \{u, v\}$
repeat:
    the querier picks some $\{u, v\}$ with $a_{\{u,v\}} = *$
    if $u \notin S$ and $v \notin S$: assign $a_{\{u,v\}} \leftarrow 0$
    if $u \notin S$ and $v \in S$ (or vice versa, swapping the roles of $u$ and $v$):
        if there exists $w \in S \setminus \{v\}$ with $a_{\{u,w\}} = *$: assign $a_{\{u,v\}} \leftarrow 0$
        else: assign $a_{\{u,v\}} \leftarrow 1$ and update $S \leftarrow S \cup \{u\}$

Since the invariant holds right before the $N^{\text{th}}$ query, this query must be $\{u, v\}$ for the unique node $u \notin S$ and the unique node $v \in S$ with $a_{\{u,v\}} = *$. Also, $u$ has no incident edge assigned 1, and every node in $S$ has an incident edge assigned 1. Assigning $a_{\{u,v\}} \leftarrow 0$ would yield a graph with a degree-0 node, namely $u$. Assigning $a_{\{u,v\}} \leftarrow 1$ would yield a graph with no degree-0 node.

**Exercise 16.9:** The algorithm from the proof of Theorem 16.4 is uniform and runs in time $O(n) = O(\sqrt{N})$ as long as we can find distinct $u, v \in S$ in constant time. For this, we have variables for $u$ and $v$ and maintain the invariant that $u < v$ and $S = \{u, v, v+1, v+2, v+3, \ldots, n\}$ (so the algorithm need not explicitly maintain $S$). To remove $v$ from $S$: update $v \leftarrow v + 1$. To remove $u$ from $S$: update $u \leftarrow v$ and $v \leftarrow v + 1$.

**Exercise 16.10.a:** The following decision tree maintains the set $S$ of contenders, which are nodes where each possible incoming edge is 1 or $*$, and each possible outgoing edge is 0 or $*$. It computes $\textsc{Celebrity}_N$ as in the proof of Theorem 16.4.

initialize $S \leftarrow [n]$
repeat $n-1$ times:
    query $(u,v)$ for distinct nodes $u,v \in S$ of lowest query-degrees
    if it's 0: remove $v$ from $S$
    if it's 1: remove $u$ from $S$
let $w$ be the only remaining node in $S$
for each node $u \neq w$:
    if $(u,w)$ hasn't been queried: query $(u,w)$ and reject if it's 0
    if $(w,u)$ hasn't been queried: query $(w,u)$ and reject if it's 1
accept

The first phase makes $n-1$ queries. We just need to argue that $\text{qdeg}(w) \geq \lfloor \log n \rfloor$ at the end of the first phase, since then the second phase makes $\leq 2(n-1) - \lfloor \log n \rfloor$ queries, for a total of $\leq 3(n-1) - \lfloor \log n \rfloor$ queries. First, suppose $n$ is a power of 2. The first $n/2$ queries (first stage) are disjoint pairs of nodes of query-degree 0; half of these nodes get removed from $S$ and the other half's query-degrees get incremented to 1. The next $n/4$ queries (second stage) are disjoint pairs of nodes in $S$; half of these nodes get removed from $S$ and the other half's query-degrees get incremented to 2. This continues for $\log n$ stages, after which only one node remains in $S$, and it has query-degree $\log n$. If $n$ isn't a power of 2, then $\text{qdeg}(w)$ after the first phase can be no smaller than if $n$ were rounded down to a power of 2, in other words, $\lfloor \log n \rfloor$.

**Exercise 16.10.b:** Let $T_i$ and $\text{qdeg}_i$ denote $T$ and qdeg after round $i$. We claim that the first phase maintains the invariant that every node in $T_i$ is a contender and $|T_i| = n - i$ and $\sum_{t \in T_i} 2^{\text{qdeg}_i(t)} \leq n$. This holds for $i = 0$ since $\sum_{t \in [n]} 2^{\text{qdeg}_0(t)} = n \cdot 2^0 = n$. To see that the invariant is maintained, assume it holds for $i-1$. It's straightforward to see that every node in $T_i$ remains a contender and $|T_i| = n - i$. If $u, v \in T_{i-1}$ and $\text{qdeg}_{i-1}(u) \leq \text{qdeg}_{i-1}(v)$ then

$$2^{\text{qdeg}_i(u)} \;=\; 2^{\text{qdeg}_{i-1}(u)+1} \;=\; 2^{\text{qdeg}_{i-1}(u)} + 2^{\text{qdeg}_{i-1}(u)} \;\leq\; 2^{\text{qdeg}_{i-1}(u)} + 2^{\text{qdeg}_{i-1}(v)}$$

and thus:

$$\textstyle\sum_{t \in T_i} 2^{\text{qdeg}_i(t)} \;\leq\; \sum_{t \in T_{i-1}} 2^{\text{qdeg}_{i-1}(t)} \;\leq\; n$$

Likewise, if $u, v \in T_{i-1}$ and $\text{qdeg}_{i-1}(u) > \text{qdeg}_{i-1}(v)$ then $\sum_{t \in T_i} 2^{\text{qdeg}_i(t)} \leq n$. If $u \notin T_{i-1}$ or $v \notin T_{i-1}$ then $\text{qdeg}_i(t) = \text{qdeg}_{i-1}(t)$ for every $t \in T_i$, so:

$$\textstyle\sum_{t \in T_i} 2^{\text{qdeg}_i(t)} \;=\; \sum_{t \in T_i} 2^{\text{qdeg}_{i-1}(t)} \;\leq\; \sum_{t \in T_{i-1}} 2^{\text{qdeg}_{i-1}(t)} \;\leq\; n$$

Since the invariant holds after round $n-1$, $w$ is a contender and $2^{\text{qdeg}_{n-1}(w)} \leq n$ and thus $\text{qdeg}_{n-1}(w) \leq \lfloor \log n \rfloor$. After $2(n-1) - \lfloor \log n \rfloor - 1$ more rounds (so round $3(n-1) - \lfloor \log n \rfloor - 1$ overall), $w$ is still a contender and has at least one unqueried edge. At this point, $w$ could be a celebrity (if $w$'s unqueried edges are assigned the "right" bits—1 for incoming and 0 for outgoing), or the graph might have no celebrity (if all but one of $w$'s unqueried edges are assigned the "right" bits, and the other is assigned the "wrong" bit, since then $w$ isn't a celebrity, and no other node can be a celebrity since either $w$'s incoming edges are all assigned 1 or $w$'s outgoing edges are all assigned 0).

**Exercise 16.11:** $\mathrm{Det}(f \circ g^N) \leq \mathrm{Det}(f)\mathrm{Det}(g)$: We combine a querier strategy for $f$ that makes $\leq \mathrm{Det}(f)$ queries, and a querier strategy for $g$ that makes $\leq \mathrm{Det}(g)$ queries, to get a querier strategy for $f \circ g^N$ that makes $\leq \mathrm{Det}(f)\mathrm{Det}(g)$ queries: On input $(y^1, \ldots, y^N)$, whenever $f$'s querier queries $x_i$, run $g$'s querier on $y^i$ to get $g(y^i)$. Then $f$'s querier outputs $f\big(g(y^1), \ldots, g(y^N)\big) = (f \circ g^N)(y^1, \ldots, y^N)$. Since this runs $g$'s querier $\leq \mathrm{Det}(f)$ times, and each run makes $\leq \mathrm{Det}(g)$ queries, overall this makes $\leq \mathrm{Det}(f)\mathrm{Det}(g)$ queries to the bits of $(y^1, \ldots, y^N)$.

$\mathrm{Det}(f \circ g^N) \geq \mathrm{Det}(f)\mathrm{Det}(g)$: We combine an adversary strategy for $f$ that makes the game last $\geq \mathrm{Det}(f)$ rounds, and an adversary strategy for $g$ that makes the game last $\geq \mathrm{Det}(g)$ rounds, to get an adversary strategy for $f \circ g^N$ that makes the game last $\geq \mathrm{Det}(f)\mathrm{Det}(g)$ rounds: Let $a \in \{0, 1, *\}^N$ denote a partial assignment to $x = x_1 \cdots x_N$, and $b^i \in \{0, 1, *\}^M$ denote a partial assignment to $y^i = y^i_1 \cdots y^i_M$. The adversary for $f \circ g^N$ uses $g$'s adversary strategy on each block $y^i$ independently. Whenever some bit $y^i_j$ is queried, if $g$'s adversary would be forced to reveal the value of $g(y^i)$ (that is, assigning $b^i_j \leftarrow 0$ would make $g|_{b^i}$ constant, and assigning $b^i_j \leftarrow 1$ would make $g|_{b^i}$ the other constant), then the adversary for $f \circ g^N$ tells $f$'s adversary that $x_i$ is being queried, and finds out which bit $a_i$ it would assign, and then assigns $b^i_j$ to make $g|_{b^i}$ the constant $a_i$ function. Call such a round *major*. There must be $\geq \mathrm{Det}(f)$ major rounds before $f|_a$ is constant. Before that, $f|_a$ could go either way by assigning appropriate bits to all $a_i = *$, and for each $i$ with $a_i = *$, $g|_{b^i}$ could go either way (either bit for $a_i$) by assigning appropriate bits to all $b^i_j = *$. Thus before $\mathrm{Det}(f)$ major rounds, $(f \circ g^N)|_{b^1, \ldots, b^N}$ could go either way. Each major round corresponds to $\geq \mathrm{Det}(g)$ rounds (queries within the associated block $y^i$), so the game lasts $\geq \mathrm{Det}(f)\mathrm{Det}(g)$ rounds.

**Exercise 16.12:** This is like the proof that every read-once formula expresses an evasive function (Theorem 16.1). View the definition of $f = \text{MAJ TREE}_N$ as a read-once "formula" $\varphi$ with $\text{MAJORITY}_3$ gates ("M gates"). The following adversary strategy maintains the invariant that $\psi$ expresses $f|_a$ and every unqueried variable still appears in $\psi$:

initialize $\psi \leftarrow \varphi$ and $a \leftarrow **\cdots*$
repeat $N-1$ times:
    the querier picks some variable $x_i$
    if $x_i$ feeds into an M gate in $\psi$:
        assign $a_i \leftarrow 0$
        replace the M gate in $\psi$ with an $\wedge$ gate with the other two incoming wires
    if $x_i$ feeds into an $\wedge$ gate:
        assign $a_i \leftarrow 1$
        delete the $\wedge$ gate from $\psi$ and use its other incoming wire as its outgoing wire

When only one variable $x_i$ remains unqueried, $f|_a$ is $x_i$ (since $\psi$ expresses $f|_a$ and $x_i$ still appears in $\psi$ by the invariant), which is nonconstant.

**Exercise 16.13.a:** Given a circuit $C$ computing $f : \{0,1\}^n \to \{0,1\}$, this recursive subroutine maintains the invariant that $\mathrm{qc}(C) = \mathrm{Det}(f)$.

```
qc(C):
    if C(x) is the same for all x ∈ {0,1}^n: return 0
    else: return min_{i∈[n]}(1 + max(qc(C|_{*···*0*···*}), qc(C|_{*···*1*···*})))
          (these are the contractions of C with 0 or 1 plugged in for x_i)
```

If $f$ isn't a constant function, then the algorithm tries all possible variables for the root to query, and picks whichever leads to a shallowest decision tree.

The recursion depth is $\leq n + 1 \leq N$. Each stack frame only needs $O(N)$ bits to repeatedly evaluate $C$ (reusing space for all $x \in \{0,1\}^n$) and store other local data. The space efficiency is:

$$\big(N \text{ stack frames at a time}\big) \cdot \big(O(N) \text{ bits per stack frame}\big) \;=\; O(N^2) \text{ bits}$$

**Exercise 16.13.b:** For any $a \in \{0,1,*\}^n$ and $b \in \{0,1\}$ and $i \in [n]$ such that $a_i = *$, let $a^{i,b}$ be $a$ except that $(a^{i,b})_i = b$. This dynamic programming algorithm stores a dictionary *det* such that $det[a] = \mathrm{Det}(f|_a)$ for each partial assignment $a$.

```
for each a ∈ {0,1,*}^n in increasing order of number of *s:
    if f(x) is the same for all x ∈ {0,1}^n consistent with a: let det[a] ← 0
    else: let det[a] ← min_{i such that a_i = *} (1 + max(det[a^{i,0}], det[a^{i,1}]))
return det[*···*]
```

The "if" inside the loop checks whether $f|_a$ is constant. If it's not, the "else" tries all possible variables for the root to query, and picks whichever leads to a shallowest decision tree.

The input size is $N = 2^n$. The loop has $3^n = N^{\log_2(3)}$ iterations, each of which takes $2^n \operatorname{poly} n = N \operatorname{polylog} N$ time, so the overall running time is $N^{\log_2(3)+1} \operatorname{polylog} N \in \operatorname{poly} N$. (To improve the time efficiency to $N^{\log_2(3)} \operatorname{polylog} N$, we can avoid the loop over all $x$: To check whether $f|_a$ is constant in $\operatorname{polylog} N$ time—assuming $a_j = *$ for some $j$—we can check whether $det[a^{j,0}] = 0 = det[a^{j,1}]$ and $f(y) = f(z)$ for some $y, z \in \{0,1\}^n$ consistent with $a^{j,0}$ and $a^{j,1}$ respectively.)

**Exercise 16.14:** Let $F = \text{Parity of Sorted}$.

$\text{Det}(F) \leq \lceil \log(N+1) \rceil$: First, assume $N+1$ is a power of 2. The following decision tree does binary search to find the index of the last 0 in the input—assuming the input has a phantom 0 at "index 0"—maintaining the invariant that this index is $\geq \ell$ ("lower") and $< u$ ("upper") and that $u - \ell = (N+1)/2^k$ after the $k^{\text{th}}$ iteration:

```
initialize ℓ ← 0 and u ← N + 1
while u − ℓ > 1:
    let m ← (ℓ + u)/2   ("middle")
    query x_m
    if x_m = 0: update ℓ ← m
    if x_m = 1: update u ← m
output (N − ℓ) mod 2
```

After $k = \log(N+1)$ queries, $u - \ell = (N+1)/2^k = 1$, so $\ell$ is the last 0's index and the input has $N - \ell$ many 1s (at indices $\ell + 1, \ldots, N$).

If $N+1$ isn't a power of 2, consider the least $M > N$ such that $M+1$ is a power of 2. Then $\text{Det}(F_N) \leq \text{Det}(F_M) \leq \log(M+1) = \lceil \log(N+1) \rceil$ by a projection reduction (Lemma 16.5) from $F_N$ to $F_M$ that prepends $M - N$ many 0s to the input.

$\text{Det}(F) \geq \lceil \log(N+1) \rceil$: Letting the partial assignment $a \in \{0, 1, *\}^N$ record the results of queries so far—and implicitly $a_0 = 0$—the following adversary strategy maintains the invariant that $a_i \in \{0, *\}$ if $i \leq \ell$, and $a_i = *$ if $\ell < i < u$, and $a_i \in \{1, *\}$ if $i \geq u$, and that $u - \ell \geq (N+1)/2^k$ after the $k^{\text{th}}$ iteration:

```
initialize ℓ ← 0 and u ← N + 1
while u − ℓ > 1:
    let m ← ⌊(ℓ + u)/2⌋
    the querier picks some x_i
    if i ≤ m: respond 0 and update ℓ ← m
    if i > m: respond 1 and update u ← m + 1
```

After $k = \lceil \log(N+1) \rceil - 1$ queries, $u - \ell \geq (N+1)/2^k > 1$, so assigning $a_{\ell+1}$ either 0 or 1—along with $a_i = 0$ for all $i \leq \ell$ and $a_i = 1$ for all $i \geq \ell + 2$—would yield valid inputs consistent with $a$ but with different outputs, so $F_N|_a$ is nonconstant for the $a$ before iteration $\lceil \log(N+1) \rceil$.

**Exercise 16.15:** By Exercise 2.12, since $\varphi$ is unsatisfiable, some of its clauses correspond to a cycle of implications (between literals) of the form $x_i \Rightarrow \cdots \Rightarrow \overline{x_i} \Rightarrow \cdots \Rightarrow x_i$ for some variable $x_i$. To find one of these implications that's unsatisfied, we first query $x_i$. If $x_i = 1$ then we do binary search over the $x_i \Rightarrow \cdots \Rightarrow \overline{x_i}$ part, maintaining the invariant that the current segment of implications has the form $1 \Rightarrow \cdots \Rightarrow 0$, until we narrow it down to an unsatisfied implication $1 \Rightarrow 0$. If $x_i = 0$, we do the same thing with the $\overline{x_i} \Rightarrow \cdots \Rightarrow x_i$ part of the cycle. Each iteration of the binary search queries one variable (corresponding to the literal in the middle of the current segment of implications), and there are $O(\log N)$ iterations, so the decision tree's depth is $O(\log N)$.

**Exercise 16.16.a:** $\text{Rand}_1(f) > d$ if there exists a distribution $D$ over 1-inputs of $f$ such that for every depth-$d$ deterministic decision tree $T$ that rejects all 0-inputs of $f$, $\mathbf{Pr}_{x\sim D}\big[T(x)=0\big] > 1/3$.

We prove the contrapositive. Suppose $\text{Rand}_1(f) \leq d$ by some depth-$d$ randomized decision tree, which is a distribution $R$ over depth-$d$ deterministic decision trees that reject all 0-inputs of $f$. Consider any distribution $D$ over 1-inputs of $f$. Taking probabilities over $T \sim R$ and $x \sim D$:

$$\min_T\big(\mathbf{Pr}_x\big[T(x)=0\big]\big) \;\leq\; \mathbf{Pr}_{T,x}\big[T(x)=0\big] \;\leq\; \max_x\big(\mathbf{Pr}_T\big[T(x)=0\big]\big) \;\leq\; 1/3$$

Thus for some depth-$d$ deterministic decision tree $T$ that rejects all 0-inputs of $f$, $\mathbf{Pr}_x\big[T(x)=0\big] \leq 1/3$.

**Exercise 16.16.b:** The upper bound is Lemma 16.6. We prove the matching lower bound using Exercise 16.16.a. Let $D$ be the uniform distribution over all inputs $x$ with exactly one 1. Consider any depth-$(\lceil 2N/3 \rceil - 1)$ deterministic decision tree $T$ that rejects the all-0s input, so the leftmost root-to-leaf path (where all queries are 0s) outputs 0. If $x$ has exactly one 1 but the leftmost path doesn't query the 1, then $T$ follows this path on $x$ and errs. There are $\geq \lfloor N/3 \rfloor + 1 > N/3$ many such inputs $x$ since the leftmost path queries $\leq \lceil 2N/3 \rceil - 1$ variables. So this happens with probability $> (N/3) \cdot (1/N) = 1/3$.

**Exercise 16.17:** Let $d = \left\lceil \frac{1-2\varepsilon}{1-\varepsilon} N \right\rceil$.

The upper bound $\mathrm{Rand}_{0,1,\varepsilon}(\textsc{Or}) \leq d$ generalizes Lemma 16.7: With probability $\varepsilon$, accept (with no queries). With the remaining probability $1-\varepsilon$, pick a uniformly random $i \in [N]$, query $x_i x_{i+1} x_{i+2} \cdots x_{i+d-1}$ (wrapping around), and accept iff at least one query was a 1. If $\textsc{Or}_N(x) = 0$ then $\mathbf{Pr}[\text{accept}] = \varepsilon$. If $\textsc{Or}_N(x) = 1$ then $\mathbf{Pr}[\text{accept}] \geq \varepsilon + (1-\varepsilon)d/N \geq \varepsilon + (1-\varepsilon)\frac{1-2\varepsilon}{1-\varepsilon} = 1-\varepsilon$.

The lower bound $\mathrm{Rand}_{0,1,\varepsilon}(\textsc{Or}) \geq d$ generalizes Theorem 16.10: We prove this using the natural generalization of Lemma 16.9 to error $\varepsilon$ instead of $1/3$. For some small $\delta > 0$, define a distribution $D$ over $\{0,1\}^N$:

$$D(x) = \begin{cases} \varepsilon + \delta & \text{if } x \text{ is all-0s} \\ (1-\varepsilon-\delta)/N & \text{if } x \text{ has exactly one 1} \\ 0 & \text{otherwise} \end{cases}$$

Consider any depth-$(d-1)$ deterministic decision tree $T$. If the leftmost root-to-leaf path (where all queries are 0s) accepts, then $T$ errs on the all-0s input, which has probability $> \varepsilon$. Now, suppose the leftmost path rejects. If $x$ has exactly one 1 but the leftmost path doesn't query the 1, then $T$ follows this path on $x$ and errs. There are $\geq N-d+1$ many such inputs $x$ since the leftmost path queries $\leq d-1$ variables. So this happens with probability $\geq (N-d+1)\cdot(1-\varepsilon-\delta)/N$, which is $> \varepsilon$ if $\delta$ is small enough, since $(N-d+1)\cdot(1-\varepsilon)/N > \left(1-\frac{1-2\varepsilon}{1-\varepsilon}\right)\cdot(1-\varepsilon) = \varepsilon$. In both cases, $\mathbf{Pr}_{x\sim D}\left[T(x) \neq \textsc{Or}_N(x)\right] > \varepsilon$.

**Exercise 16.18.a:** $\text{Rand}^*(\text{MAJORITY}_5) \leq 4.5$: Query the variables in a uniformly random order until three identical input bits have been seen, and then output that bit. To analyze this, first suppose three input bits are $b$ and the other two are $\neg b$. Let the random variable $Q$ be the number of queries. Then $\mathbf{Pr}[Q = 3] = \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = 1/10$ since the probability the first query is a $b$ is $3/5$, and conditioned on that, the probability the second query is a $b$ is $2/4$, and further conditioned on that, the probability the third query is a $b$ is $1/3$. We have $\mathbf{Pr}[Q = 5] = 3/5$ since $Q = 5$ iff the fifth query is a $b$. Thus $\mathbf{Pr}[Q = 4] = 1 - 1/10 - 3/5 = 3/10$. Finally, $\mathbf{E}[Q] = \frac{1}{10} \cdot 3 + \frac{3}{10} \cdot 4 + \frac{3}{5} \cdot 5 = 4.5$. Similarly, if the input has more than three of the same bit, then $\mathbf{E}[Q]$ is even lower.

$\text{Rand}^*(\text{MAJORITY}_5) \geq 4.5$: We use Lemma 16.11. Let $D$ be the uniform distribution over all length-5 weight-2 bit strings. Consider any deterministic decision tree $T$ that computes $\text{MAJORITY}_5$. Note that $T$ always makes at least three queries, since $T$ only terminates after seeing three of the same bit in the input. With probability $3/5$, the first four queries don't include all three 0s, so $T$ makes five queries. With probability $3/10$, the first four queries include all three 0s but the first three don't, so $T$ makes four queries. Thus:

$$\mathbf{E}_{x \sim D}\big[\text{number of queries } T \text{ makes to } x\big] \geq \tfrac{3}{5} \cdot 5 + \tfrac{3}{10} \cdot 4 + \tfrac{1}{10} \cdot 3 = 4.5$$

**Exercise 16.18.b:** This returns $\text{MAJ}_5\ \text{TREE}_N(x)$ with probability 1:

eval($x$):
  if $N = 1$: query and return $x$
  permute the five fifths of $x$ uniformly at random, then name them $u, v, w, y, z$
  if eval($u$) = eval($v$) = eval($w$): return that bit
  else if eval($y$) equals two of eval($u$), eval($v$), eval($w$): return that bit
  else: return eval($z$)

Let the random variable $Q_x$ be the number of queries on input $x$. For each $d = \log_5(N) \geq 0$ (the depth of $\text{MAJ}_5$ TREE) define:

$$q_d \ = \ \max_{x \in \{0,1\}^{5^d}} \big(\mathbf{E}[Q_x]\big)$$

We claim eval maintains the invariant that $q_d \leq 4.5^d$. This holds for the base case $d = 0$ since $q_0 = 1 = 4.5^0$. To see that the invariant is maintained, assume it holds for $d-1$, and consider any $x \in \{0,1\}^{5^d}$. Consider the values of $\text{MAJ}_5\ \text{TREE}_{N/5}$ on the five fifths of $x$. If three of these bits are the same and the other two are different, then with probability $1/10$, eval($x$) returns after three recursive calls (on $u, v, w$), and with probability $3/10$, eval($x$) returns after four recursive calls (on $u, v, w, y$), and with probability $3/5$, eval($x$) returns after five recursive calls (on $u, v, w, y, z$):

$$\begin{aligned}
\mathbf{E}[Q_x] \ &= \ \tfrac{1}{10}\big(\mathbf{E}[Q_u] + \mathbf{E}[Q_v] + \mathbf{E}[Q_w]\big) + \\
&\quad\ \tfrac{3}{10}\big(\mathbf{E}[Q_u] + \mathbf{E}[Q_v] + \mathbf{E}[Q_w] + \mathbf{E}[Q_y]\big) + \\
&\quad\ \tfrac{3}{5}\big(\mathbf{E}[Q_u] + \mathbf{E}[Q_v] + \mathbf{E}[Q_w] + \mathbf{E}[Q_y] + \mathbf{E}[Q_z]\big) \\
&\leq \ \tfrac{1}{10} \cdot 3q_{d-1} + \tfrac{3}{10} \cdot 4q_{d-1} + \tfrac{3}{5} \cdot 5q_{d-1} \ = \ 4.5q_{d-1} \ \leq \ 4.5^d
\end{aligned}$$

If the values of $\text{MAJ}_5\ \text{TREE}_{N/5}$ on the five fifths of $x$ have more than three identical bits, then $\mathbf{E}[Q_x]$ is even lower. We conclude $\text{Rand}^*(\text{MAJ}_5\ \text{TREE}) \leq q_d \leq 4.5^d = N^{\log_5(4.5)}$.

**Exercise 16.19.a:** We show there exists a depth-$k$ deterministic decision tree $T$ such that:

$$\mathbf{Pr}_{(x,y)\sim D}\Big[T(x,y) \neq \textsc{Index}_N(x,y)\Big] \;\leq\; 1/4 \;\leq\; 1/3$$

Query $y_{k-1}\cdots y_1$ and $x_{y_{k-1}\cdots y_1 0}$ and output the latter. Conditioned on an arbitrary outcome of $y_{k-1}\cdots y_1$, by the law of total probability we have

$$\begin{aligned}\mathbf{Pr}\Big[T(x,y) \neq x_y\Big] \;&=\; \mathbf{Pr}[y_0 = 0]\cdot \mathbf{Pr}\Big[T(x,y) \neq x_y \;\Big|\; y_0 = 0\Big] + \\ &\quad\; \mathbf{Pr}[y_0 = 1]\cdot \mathbf{Pr}\Big[T(x,y) \neq x_y \;\Big|\; y_0 = 1\Big] \\ &=\; \tfrac{1}{2}\cdot 0 + \tfrac{1}{2}\cdot\tfrac{1}{2} \;=\; 1/4\end{aligned}$$

since $\mathbf{Pr}\Big[T(x,y) \neq x_y \;\Big|\; y_0 = 1\Big] = \mathbf{Pr}\Big[x_{y_{k-1}\cdots y_1 0} \neq x_{y_{k-1}\cdots y_1 1}\Big] = 1/2$ by the full independence of all bits of $x$.

**Exercise 16.19.b:** Consider any depth-$k$ deterministic decision tree $T$. We argue that:

$$\mathbf{Pr}_{(x,y)\sim D}\left[T(x,y) \neq \textsc{Index}_N(x,y)\right] \;=\; 1/2 \;>\; 1/3$$

In fact, this holds conditioned on reaching any particular leaf of $T$ (that has positive probability), say corresponding to a partial assignment $a$ with at most $k$ non-$*$s and outputting a bit $c$. If the $y$ part of $a$ is fully assigned—and thus the $x$ part is fully unassigned—then $\mathbf{Pr}[c \neq x_y] = \mathbf{Pr}\left[c \neq \oplus(y) \oplus b\right] = 1/2$ since $b$ is uniformly random (conditioned on $(x,y)$ being consistent with $a$). If the $y$ part of $a$ is not fully assigned, then regardless of the value of $b$ (conditioned on $(x,y)$ being consistent with $a$), $x_i = \oplus(i) \oplus b = 1$ for exactly half of all $i$ consistent with the $y$ part of $a$, and $y$ is uniformly distributed over all such $i$, so $\mathbf{Pr}[x_y = 1] = 1/2$ and thus $\mathbf{Pr}[c \neq x_y] = 1/2$.

**Exercise 16.20.a:** Consider any deterministic decision tree $T$ that computes $\textsc{Tribes}_N$. We show $\mathbf{E}_{x\sim D_0}\big[\text{number of queries } T \text{ makes to } x\big] = \Omega(N)$. Since $T$ computes $\textsc{Tribes}_N$ and $x \sim D_0$ is a 0-input, $T$ must query $x$ until it has seen the 0 in each row. In each row, since the 0's location is uniformly random and independent of the other rows, the number of queries to find it is equally likely to be any of $1, 2, \ldots, n$, which is $(n+1)/2$ in expectation. By linearity of expectation:

$$\mathbf{E}_{x\sim D_0}[\text{number of queries}] = \sum_{i\in[n]} \mathbf{E}_{x\sim D_0}[\text{number of queries in row } i] = n(n+1)/2 = \Omega(N)$$

**Exercise 16.20.b:** Let $d = N/400$ and consider any depth-$d$ deterministic decision tree $T$. We show $\mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Tribes}_N(x)\big] > 1/5$, which implies $\mathrm{Rand}_{0,1}(\textsc{Tribes}) > d = \Omega(N)$ if the error probability is $1/5$ instead of $1/3$ (by Lemma 16.9 with $1/5$), which implies $\mathrm{Rand}_{0,1}(\textsc{Tribes}) = \Omega(N)$ with error probability $1/3$ (by amplification).

If $\mathbf{Pr}_{x\sim D_0}\big[T(x) = 1\big] > 2/5$ then $\mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Tribes}_N(x)\big] > 1/5$ as desired. So assume $\mathbf{Pr}_{x\sim D_0}\big[T(x) = 1\big] \leq 2/5$ and thus $\mathbf{Pr}_{x\sim D_0}\big[T(x) = 0\big] \geq 3/5$. We claim that this implies $\mathbf{Pr}_{x\sim D_1}\big[T(x) = 0\big] > 2/5$, which implies $\mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Tribes}_N(x)\big] > 1/5$ as desired.

To sample $y \sim D_1$, we can sample $x \sim D_0$ and obtain $y$ by picking a uniformly random 0 to flip to 1. This is a joint probability space where the marginal distribution of $x$ is $D_0$ and the marginal distribution of $y$ is $D_1$. To prove the claim, it suffices to prove that

$$\mathbf{Pr}\big[T(x) \text{ queries the bit on which } x \text{ and } y \text{ differ}\big] \;<\; 1/5$$

since then:

$$\begin{aligned}
\mathbf{Pr}\big[T(y)=0\big] \;&\geq\; \mathbf{Pr}\big[T(x)=0 \text{ and } x \text{ and } y \text{ lead to the same leaf}\big] \\
&\geq\; \mathbf{Pr}\big[T(x)=0\big] - \mathbf{Pr}\big[x \text{ and } y \text{ lead to different leaves}\big] \\
&=\; \mathbf{Pr}\big[T(x)=0\big] - \mathbf{Pr}\big[T(x) \text{ queries the bit on which } x \text{ and } y \text{ differ}\big] \\
&>\; 3/5 - 1/5 \;=\; 2/5
\end{aligned}$$

We claim $\mathbf{Pr}\big[T(x) \text{ queries} > n/10 \text{ many 0s}\big] < 1/10$, which implies:

$$\begin{aligned}
&\mathbf{Pr}\big[T(x) \text{ queries the bit on which } x \text{ and } y \text{ differ}\big] \\
\leq\; &\mathbf{Pr}\big[T(x) \text{ queries} > n/10 \text{ many 0s}\big] + \\
&\mathbf{Pr}\big[T(x) \text{ queries the bit on which } x \text{ and } y \text{ differ} \;\big|\; T(x) \text{ queries} \leq n/10 \text{ many 0s}\big] \\
<\; &1/10 + 1/10 \;=\; 1/5
\end{aligned}$$

Say a query is *early* if it's one of the first $n/2$ queries made in its row, and *late* otherwise. At most $n/20$ late queries of $T(x)$ are 0s, since otherwise $T(x)$ would make $> (n/20)\cdot(n/2) \geq d$ queries in total. So to prove the latest claim, it suffices to argue $\mathbf{Pr}\big[> n/20 \text{ early queries of } T(x) \text{ are 0s}\big] < 1/10$. Conditioned on a particular query being early, it's 0 with probability $\leq 1/(n/2)$. By linearity of expectation:

$$\begin{aligned}
&\mathbf{E}\big[\text{number of early queries of } T(x) \text{ that are } 0\big] \\
=\; &\textstyle\sum_{i\in[d]} \mathbf{Pr}\big[i^{\text{th}} \text{ query of } T(x) \text{ is early and } 0\big] \\
\leq\; &\textstyle\sum_{i\in[d]} \mathbf{Pr}\big[i^{\text{th}} \text{ query of } T(x) \text{ is } 0 \;\big|\; \text{it's early}\big] \\
\leq\; &d/(n/2) \\
\leq\; &n/200
\end{aligned}$$

By Lemma 7.13:

$$\begin{aligned}
&\mathbf{Pr}\big[> n/20 \text{ early queries of } T(x) \text{ are 0s}\big] \\
\leq\; &\mathbf{E}\big[\text{number of early queries of } T(x) \text{ that are } 0\big]/(n/20) \\
\leq\; &(n/200)/(n/20) \;=\; 1/10
\end{aligned}$$

**Exercise 16.21.a:** Let $d = \mathrm{Rand}_1(f)$. Consider any depth-$d$ randomized decision tree—which is a distribution $R$ over depth-$d$ deterministic decision trees—such that $\mathbf{Pr}_{T\sim R}\big[T(x)=1\big] \geq 2/3$ if $f(x)=1$ and $\mathbf{Pr}_{T\sim R}\big[T(x)=0\big]=1$ if $f(x)=0$. Define a randomized decision tree for $g$ that samples $T \sim R$ and outputs $T'(x^1, x^2, \ldots, x^M) = T(x^1) \vee T(x^2) \vee \cdots \vee T(x^M)$. This makes $\leq Md$ queries and:

- If $g(x^1, \ldots, x^M) = 1$, say $f(x^i) = 1$, then $\mathbf{Pr}\big[T'(x^1, \ldots, x^M) = 1\big] \geq \mathbf{Pr}\big[T(x^i) = 1\big] \geq 2/3$.
- If $g(x^1, \ldots, x^M) = 0$ then $\mathbf{Pr}\big[T'(x^1, \ldots, x^M) = 0\big] = 1$.

**Exercise 16.21.b:** Let $d = \mathrm{Rand}_0(f)$. Consider any depth-$d$ randomized decision tree—which is a distribution $R$ over depth-$d$ deterministic decision trees—such that $\mathbf{Pr}_{T\sim R}\big[T(x)=1\big]=1$ if $f(x)=1$ and $\mathbf{Pr}_{T\sim R}\big[T(x)=0\big]\geq 2/3$ if $f(x)=0$. First, consider this randomized decision tree, which may run forever:

```
on input x^1, ..., x^M:
    for i ← 1, ..., M:
        repeatedly sample T ~ R until T(x^i) = 0
    output 0
```

- If $g(x^1,\ldots,x^M)=1$, say $f(x^i)=1$, then this runs forever since $\mathbf{Pr}_{T\sim R}\big[T(x^i)=0\big]=0$.
- If $g(x^1,\ldots,x^M)=0$ then by linearity of expectation,

$$\begin{aligned}\mathbf{E}[\text{number of queries}] &= \textstyle\sum_{i=1}^{M}\mathbf{E}[\text{number of queries in outer iteration } i]\\ &\leq \textstyle\sum_{i=1}^{M} 1.5d \;=\; 1.5Md\end{aligned}$$

since $\mathbf{Pr}_{T\sim R}\big[T(x^i)=0\big]\geq 2/3$ and thus $\mathbf{E}\big[\text{number of runs of } T(x^i) \text{ until } T(x^i)=0\big]\leq 3/2$ (Lemma 0.11).

We modify this randomized decision tree to stop and output 1 if it hasn't already halted after $4.5Md$ queries have been made in total.

- If $g(x^1,\ldots,x^M)=1$ then $\mathbf{Pr}[\text{output } 1]=1$.
- If $g(x^1,\ldots,x^M)=0$ then by Lemma 7.13:

$$\mathbf{Pr}[\text{output } 1] = \mathbf{Pr}\big[\text{more than } 4.5Md \text{ queries would be made}\big] \leq 1.5Md/4.5Md = 1/3$$

Thus $\mathrm{Rand}_0(g)\leq 4.5Md$.

**Exercise 16.22:** As shown in §6.1.2, $\textsc{Parity}_N$ has a bounded-fan-in formula with depth $2\log N$ and $N^2$ literals, after pushing negations to the input variables. We just need a version where the layers of gates strictly alternate between $\vee$ and $\wedge$. Then the projection reduction maps $x$ to the values of the literals feeding into this formula for $\textsc{And-Or Tree}_{N^2}$. The following mutually recursive subroutines, which we name $\oplus(N)$ and $\overline{\oplus}(N)$, return such formulas for $\textsc{Parity}_N$ and $\overline{\textsc{Parity}_N}$ respectively.

$\oplus(N)$:
    if $N = 1$: return the literal $x_1$
    return $\big(\oplus(N/2)(x_1 \cdots x_{N/2}) \wedge \overline{\oplus}(N/2)(x_{N/2+1} \cdots x_N)\big) \vee$
        $\big(\overline{\oplus}(N/2)(x_1 \cdots x_{N/2}) \wedge \oplus(N/2)(x_{N/2+1} \cdots x_N)\big)$

$\overline{\oplus}(N)$:
    if $N = 1$: return the literal $\overline{x_1}$
    return $\big(\oplus(N/2)(x_1 \cdots x_{N/2}) \wedge \oplus(N/2)(x_{N/2+1} \cdots x_N)\big) \vee$
        $\big(\overline{\oplus}(N/2)(x_1 \cdots x_{N/2}) \wedge \overline{\oplus}(N/2)(x_{N/2+1} \cdots x_N)\big)$

These maintain the invariant that the returned formulas have depth $2\log N$ and $N^2$ literals and strictly alternate between $\vee$ and $\wedge$ layers: The four recursive calls return formulas with depth $2\log(N/2)$ and $(N/2)^2$ literals, so the original calls return formulas with depth $2\log(N/2)+2 = 2\log N$ and $4(N/2)^2 = N^2$ literals.

**Exercise 16.23.a:** Let $f = \text{INDEX}_N$ where $N = 2^k + k$. We prove $\text{Cert}(f) = k + 1$. We have $\text{Cert}(f) \leq \text{Det}(f) \leq k + 1$ and $\text{Cert}(f) \geq k + 1$ since the proof of Theorem 16.2 shows that every certificate has size $\geq k + 1$.

**Exercise 16.23.b:** Let $f = \text{CONNECTED}_N$ where $N = \binom{n}{2}$ and $n > 1$. We prove $\text{Cert}(f) = \lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$.

$\leq$: To certify $f(g) = 1$, revealing a tree of $n-1 \leq \lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$ edges present in $g$ is sufficient. To certify $f(g) = 0$, revealing all non-edges with one endpoint in some $S \subseteq [n]$ (a union of some but not all connected components) and the other in $[n] \setminus S$ is sufficient, and there are $|S| \cdot (n - |S|) \leq \lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$ such edges.

$\geq$: Let $S$ be the first $\lfloor n/2 \rfloor$ nodes and $T$ be the other $\lceil n/2 \rceil$ nodes. Let $g$ have all possible edges with both endpoints in $S$ or both endpoints in $T$, and no edges with one endpoint in $S$ and the other in $T$. To certify $f(g) = 0$, revealing all $|S| \cdot |T|$ non-edges with one endpoint in $S$ and the other in $T$ is necessary: If such a non-edge isn't revealed, then assigning it 1 and assigning all others according to $g$ would yield a connected graph.

**Exercise 16.23.c:** Let $f = \textsc{Celebrity}_N$ where $N = n(n-1)$ and $n > 1$. We prove $\text{Cert}(f) = 2(n-1)$.

$\leq$: To certify $f(g) = 1$, revealing the celebrity's $n-1$ incoming edges and $n-1$ outgoing non-edges is sufficient. To certify $f(g) = 0$, revealing $\leq n \leq 2(n-1)$ bits is sufficient: For each node, we can certify that it's not a celebrity by revealing an incoming non-edge or an outgoing edge.

$\geq$: To certify $f(g) = 1$, revealing the celebrity $u$'s $n-1$ incoming edges and $n-1$ outgoing non-edges is necessary: Suppose not all are revealed. Assign one of them the opposite bit, and assign the rest according to $g$. This yields a graph $h$ with no celebrity, because $u$ is not a celebrity, and neither is any other node $v$: If $h_{(u,v)} = 1$ and $g_{(u,v)} = 0$ then $h_{(v,u)} = g_{(v,u)} = 1$ and thus $v$ isn't a celebrity in $h$. If $h_{(v,u)} = 0$ and $g_{(v,u)} = 1$ then $h_{(u,v)} = g_{(u,v)} = 0$ and thus $v$ isn't a celebrity in $h$. If $h_{(u,v)} = g_{(u,v)}$ and $h_{(v,u)} = g_{(v,u)}$ then $v$ isn't a celebrity in $h$ since it isn't a celebrity in $g$.

**Exercise 16.23.d:** Let $f = \text{AND-OR TREE}_N$ where $N = 2^d$ for even $d$. We prove $\text{Cert}(f) = 2^{d/2} = \sqrt{N}$.

$\geq$: $\text{Cert}(f) \geq \sqrt{\text{Det}(f)} \geq \sqrt{N}$ by Theorem 16.22 and Theorem 16.1.

$\leq$: To certify $f(x) = 1$: The $\vee$ gate at the root evaluates to 1. Pick one child that also evaluates to 1. Pick both children of that $\wedge$ gate. For both of those $\vee$ gates, pick one child that evaluates to 1, and so on. This forms a tree with $2^{d/2}$ leaves since each $\vee$ gate gets one child and each $\wedge$ gate (of which there are $d/2$ layers) gets two children. Revealing the 1 at each such leaf yields a certificate for $(f, x)$.

To certify $f(x) = 0$: The $\vee$ gate at the root evaluates to 0. Pick both children (which also evaluate to 0). For both of those $\wedge$ gates, pick one child that evaluates to 0. For both of those $\vee$ gates, pick both children, and so on. This forms a tree with $2^{d/2}$ leaves since each $\wedge$ gate gets one child and each $\vee$ gate (of which there are $d/2$ layers) gets two children. Revealing the 0 at each such leaf yields a certificate for $(f, x)$.

**Exercise 16.24.a:** Let $f = F_N$. $\mathrm{Cert}_1(f) \leq 1 \in \mathrm{polylog}\, N$ by the 1-DNF $(x_1) \vee (x_2) \vee \cdots \vee (x_{N/2})$, which accepts all 1-inputs of $f$ and rejects all 0-inputs of $f$. $\mathrm{Cert}_0(f) \leq 1 \in \mathrm{polylog}\, N$ by the 1-DNF $(x_{N/2+1}) \vee (x_{N/2+2}) \vee \cdots \vee (x_N)$, which accepts all 0-inputs of $f$ and rejects all 1-inputs of $f$. $\mathrm{Det}(f) \geq N/2 \notin \mathrm{polylog}\, N$ by the adversary strategy that always responds 0: After $N/2-1$ queries, the left and right halves each still have an unqueried variable, so the input could still be a 1-input or a 0-input. Thus $F \in \mathsf{NP}^{\mathsf{qc}}$ and $F \in \mathsf{coNP}^{\mathsf{qc}}$ and $F \notin \mathsf{P}^{\mathsf{qc}}$, assuming these classes allow partial functions.

**Exercise 16.24.b:** The proof of Theorem 16.22 works in this case because the contradiction in the proof of Lemma 16.23 is that some $x$ (for which $f(x)$ may be undefined) satisfies both $\varphi_1$ and $\varphi_0$.

**Exercise 16.25:** $\Rightarrow$: We prove the contrapositive. Suppose $J$ is disjoint from some block $I \subseteq [N]$ of $x$ that $f$ is sensitive to. Then $x$ and $x^I$ are consistent with $a$, but $f(x^I) \neq f(x)$, so $a$ isn't a certificate for $f$.

$\Leftarrow$: We prove the contrapositive. Suppose $a$ isn't a certificate for $f$, so $f(y) \neq f(x)$ for some $y$ consistent with $a$. Then $I = \{i : y_i \neq x_i\}$ is disjoint from $J$ (since $y$ is consistent with $a$), but $f$ is sensitive to block $I$ since $y = x^I$ and $f(y) \neq f(x)$.

**Exercise 16.26:** Consider any monotone $f$. We know $\mathrm{Sens}(f) \leq \mathrm{BSens}(f) \leq \mathrm{Cert}(f)$, so it suffices to prove $\mathrm{Cert}(f) \leq \mathrm{Sens}(f)$, that is, $\max_x\big(\mathrm{Cert}(f,x)\big) \leq \max_y\big(\mathrm{Sens}(f,y)\big)$, in other words, for every $x$ there exists $y$ such that $\mathrm{Cert}(f,x) \leq \mathrm{Sens}(f,y)$. Assume $f(x) = 1$, since the proof is symmetric if $f(x) = 0$. Let $y$ be a lowest-weight 1-input such that $\{j : y_j = 1\} \subseteq \{j : x_j = 1\}$. For each $i$ with $y_i = 1$, we have $f(y^i) \neq f(y)$ since otherwise $y^i$ would be a 1-input such that $\{j : (y^i)_j = 1\} \subseteq \{j : x_j = 1\}$ of lower weight than $y$. Thus $\mathrm{weight}(y) \leq \mathrm{Sens}(f,y)$. Also, $a$ defined by $a_i = 1$ if $y_i = 1$ and $a_i = *$ if $y_i = 0$ is a 1-certificate (of size $\mathrm{weight}(y)$ and consistent with $x$): Assigning 0s to the $*$s yields a 1-input, and flipping any of those 0s to 1s still yields a 1-input since $f$ is monotone. Thus $\mathrm{Cert}(f,x) \leq \mathrm{weight}(y) \leq \mathrm{Sens}(f,y)$.

**Exercise 16.27:** $\Rightarrow$: This was in the proof of Lemma 16.32.

$\Leftarrow$: Suppose $P$ is a projection reduction from $g$ to $f$. Let $x = P(00\cdots0)$ and for each $j \in [k]$, let $I_j = \left\{i \ : \ P(y)_i \text{ is } y_j \text{ or } \overline{y_j}\right\}$, so $x^{I_j} = P(0\cdots010\cdots0)$ where the 1 is at index $j$. Then $f(x) = g(00\cdots0) = 0$ and for each $j \in [k]$, $f(x^{I_j}) = g(0\cdots010\cdots0) = 1$. Thus $f$ is sensitive to the disjoint blocks $I_1, \ldots, I_k$ on the 0-input $x$, so $\mathrm{BSens}_0(f) \geq k$.

**Exercise 16.28.a:** Let $d = \mathrm{Det}(f)$. Consider a depth-$d$ decision tree computing $f$, and assume every root-to-leaf path makes exactly $d$ queries (by querying and ignoring extra variables as needed). For each accepting leaf, consider the width-$d$ term that accepts exactly those inputs that lead to this leaf. The $\vee$ of these terms is a $d$-DNF that expresses $f$ (it accepts iff $x$ leads to an accepting leaf) and is unambiguous (since each 1-input only leads to one leaf and thus only satisfies one term). Therefore $\mathrm{UCert}_1(f) \leq d$.

**Exercise 16.28.b:** Consider an unambiguous $k$-DNF $\varphi$ for $f$. Partition the set of 1-inputs according to which term of $\varphi$ they satisfy. Since each term accepts exactly $2^{N-k}$ inputs (all of which are 1-inputs), the number of 1-inputs is $2^{N-k}$ times the number of terms.

$\text{OR}_N$ has an odd number $2^N - 1$ of 1-inputs, which can't be a multiple of $2^{N-k}$ unless $k = N$ (since $2^{N-k}$ is even if $k < N$).

**Exercise 16.28.c:** This is similar to the proof that $\mathrm{Det}(f) \leq \mathrm{Cert}_1(f) \cdot \mathrm{Cert}_0(f)$ (Theorem 16.22).

Suppose $\mathrm{UCert}_1(f) = k$ so an unambiguous $k$-DNF $\varphi$ expresses $f$. We design an algorithm that computes $f$ with $\leq k^2$ queries, so $\mathrm{Det}(f) \leq k^2$. Throughout the algorithm, the partial assignment $a$ records the results of queries so far, and $\varphi|_a$ is the contraction of $\varphi$ with $a$ plugged in, which is also an unambiguous DNF.

repeat until $f|_a$ is a constant function:
    pick any term of $\varphi|_a$ and query the term's variables
output the constant value of $f|_a$

We argue that this makes $\leq k + (k-1) + (k-2) + \cdots + 2 + 1 = k(k+1)/2 \leq k^2$ queries. In each iteration, every pair of terms of $\varphi|_a$ overlap each other since otherwise we could define an input that satisfies both terms, contradicting the unambiguity of $\varphi|_a$. So each term of $\varphi|_a$ either disappears or shrinks, since at least one of its variables is queried. Since $\varphi$'s terms have width $k$, and the maximum width of $\varphi|_a$'s terms decreases in each iteration, one of the following happens within $k$ iterations:

- $\varphi|_a$ has a width-0 term and thus always accepts, so $f|_a$ is constant 1.
- $\varphi|_a$ has no terms and thus always rejects, so $f|_a$ is constant 0.

The first iteration queries $\leq k$ variables, and the second iteration queries $\leq k-1$ variables, and so on.

**Exercise 16.28.d:** $\mathrm{Det}(f) > n^2$ by this adversary strategy: In each row, answer the earliest $2n-1$ queries $(1,\mathrm{null}),\ldots,(1,\mathrm{null})$ and the last query 0. After $n^2$ queries in total, some row $i$ has had $\leq n$ queries (since if each of the $n$ rows had $> n$ queries, there would be $> n^2$ queries in total) and thus:

- Some 0-input is consistent with the results of queries so far: Let each row have a 0-entry if it doesn't already. Then no row is special.
- Some 1-input is consistent with the results of queries so far: Continue the adversary strategy to assign all remaining entries outside of row $i$. For each row $h \neq i$, let $j_h$ be the column containing 0. Let $j_i$ be arbitrary. In row $i$, assign $(1,j_1),\ldots,(1,j_n)$ to the first $n$ remaining entries and $(1,\mathrm{null})$ to any further remaining entries. Then row $i$ is special.

$\mathrm{UCert}_1(f) \leq 3n-1$ because each 1-input is consistent with exactly one 1-certificate of size $3n-1$, namely the all-1s row and the 0s it points to.

We can turn $f$ into a boolean function exhibiting a similar separation, like in the proof of Theorem 16.35.

**Exercise 16.29:** Let $S$ be a 1-fooling set for $f$. We claim that in every DNF for $f$, inputs in $S$ must satisfy different terms, so the DNF has at least $|S|$ terms. Suppose for contradiction that for some DNF for $f$, some distinct $x, y \in S$ satisfy the same term. There exists a 0-input $z$ such that $z_i \in \{x_i, y_i\}$ for all $i$. Thus $z$ also satisfies that term, and the DNF incorrectly outputs 1 on input $z$.

**Exercise 16.30.a:** Just like Lemma 16.9: $f$ has randomized decision tree size complexity $> s$ if there exists a distribution $D$ over $\{0,1\}^N$ such that for every size-$s$ deterministic decision tree $T$, $\mathbf{Pr}_{x\sim D}\big[T(x) \neq f(x)\big] > 1/3$.

Let $D$ be the uniform distribution over $\{0,1\}^N$. To prove $\textsc{Parity}_N$ has randomized decision tree size complexity $\geq 2^N/3$, it suffices to show that for every deterministic decision tree $T$ of size $< 2^N/3$, $\mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Parity}_N(x)\big] > 1/3$.

$\mathbf{Pr}_{x\sim D}\big[T(x)$ queries all $N$ bits$\big] < 1/3$ since $T$ has $< 2^N/3$ many leaves at depth $N$. Conditioned on not querying all $N$ bits, $T$ errs with probability exactly $1/2$ since flipping an input bit that $T$ doesn't query would flip $\textsc{Parity}_N$'s value but not flip $T$'s output. By the chain rule:

$$\begin{aligned}
&\mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Parity}_N(x)\big] \\
\geq\ &\mathbf{Pr}_{x\sim D}\big[T(x) \text{ queries} < N \text{ bits and } T(x) \neq \textsc{Parity}_N(x)\big] \\
=\ &\mathbf{Pr}_{x\sim D}\big[T(x) \text{ queries} < N \text{ bits}\big] \cdot \mathbf{Pr}_{x\sim D}\big[T(x) \neq \textsc{Parity}_N(x) \;\big|\; T(x) \text{ queries} < N \text{ bits}\big] \\
>\ &(2/3)\cdot(1/2) \;=\; 1/3
\end{aligned}$$

**Exercise 16.30.b:** Exercise 16.22 gives a projection reduction from $\textsc{Parity}_{\sqrt{N}}$ to $\textsc{And-Or Tree}_N$, so the randomized decision tree size complexity of $\textsc{And-Or Tree}_N$ is at least that of $\textsc{Parity}_{\sqrt{N}}$, which is $\Omega(2^{\sqrt{N}})$ (Exercise 16.30.a), since the proof of Lemma 16.5 also works for randomized decision tree size.

**Exercise 16.31.a:** $\mathrm{Det}^{\oplus}(\textsc{Or}) \geq N$: The adversary responds 0 to each query. After $N-1$ queries, the value of $\textsc{Or}_N$ could be 0, and it could be 1 because if $S_1, S_2, \ldots, S_{N-1} \subseteq [N]$ are the subsets queried so far, then $\oplus(x_{S_1}) = 0,\ \oplus(x_{S_2}) = 0,\ \ldots,\ \oplus(x_{S_{N-1}}) = 0$ is homogeneous system of $N-1$ linear equations in $N$ variables over $\mathbb{Z}_2$ and thus has a nonzero solution (Theorem 0.29), which is a 1-input consistent with the queries so far.

$\mathrm{Rand}_1^{\oplus}(\textsc{Or}) \leq 2$: Consider the randomized $\oplus$-decision tree that queries and outputs the parity of a uniformly random subset of input variables. This accepts the 0-input with probability 0 and accepts each 1-input with probability $1/2$ by the random subsum principle (Lemma 7.2.*(i)*). We amplify the success probability to $3/4 \geq 2/3$ on 1-inputs by making two independent queries and accepting iff either yields 1.

**Exercise 16.31.b:** $\mathrm{Det}^{(0)}(f) \leq \mathrm{Det}^{\wedge}(f)$: Let $d = \mathrm{Det}^{\wedge}(f)$. Consider any depth-$d$ $\wedge$-decision tree for $f$. Turn it into an ordinary decision tree for $f$ by replacing each internal node that queries $\wedge(x_S)$ (for some $S \subseteq [N]$) with a sequence of nodes that query $x_i$ for all $i \in S$ one at a time until a 0 is seen or all of them were 1s. (There are $|S|$ copies of the original node's left subtree, corresponding to the $|S|$ places the first 0 could be seen. This process is continued down the tree.) For each root-to-leaf path in the new decision tree, the number of 0s is the number of 0s along the corresponding path in the original $\wedge$-decision tree, which is $\leq d$. Thus $\mathrm{Det}^{(0)}(f) \leq d$.

$\mathrm{Det}^{\wedge}(f) \leq \mathrm{Det}^{(0)}(f) \cdot O(\log N)$: Let $d = \mathrm{Det}^{(0)}(f)$. Consider any decision tree for $f$ where every root-to-leaf path reads $\leq d$ many 0s. Here's an $\wedge$-decision tree for $f$: First, do binary search on the rightmost root-to-leaf path (which only reads 1s) to find the first edge labeled 0 that would be taken on the given input. That is, if the rightmost path queries $x_{i_1}, \ldots, x_{i_k}$ (where $k \leq N$), then query $(x_{i_1} \wedge \cdots \wedge x_{i_{k/2}})$ and if it's 0 then query $(x_{i_1} \wedge \cdots \wedge x_{i_{k/4}})$ next, but if it was 1 then query $(x_{i_1} \wedge \cdots \wedge x_{i_{3k/4}})$ next, and so on. This is $O(\log N)$ queries. If the given input deviates from the rightmost path by taking an edge labeled 0, then repeat this process from that node, and so on, until reaching a leaf. Along any root-to-leaf path in this $\wedge$-decision tree, there are $O(\log N)$ queries per 0 read along the corresponding root-to-leaf path in the original decision tree. Thus the $\wedge$-decision tree has depth $d \cdot O(\log N)$, so $\mathrm{Det}^{\wedge}(f) \leq d \cdot O(\log N)$.

**Exercise 16.31.c:** $\mathrm{DTSize}(f) \leq (N+1)^{\mathrm{Det}^{\mathrm{Term}}(f)}$: Let $d = \mathrm{Det}^{\mathrm{Term}}(f)$. Consider any depth-$d$ decision tree for $f$ where each internal node can evaluate an arbitrary term. Turn this into an ordinary decision tree for $f$ by replacing each internal node with a decision tree of size $\leq N+1$ that evaluates the associated term. Since this is like a depth-$d$ tree where each internal node has $\leq N+1$ children, it has $\leq (N+1)^d$ leaves. Thus $\mathrm{DTSize}(f) \leq (N+1)^d$.

$\mathrm{Det}^{\mathrm{Term}}(f) \leq \log_{3/2}(\mathrm{DTSize}(f))$: Let $\ell = \mathrm{DTSize}(f)$. Let $T$ be a size-$\ell$ decision tree for $f$. For any non-root node $v$:

- Let $\varphi_v$ be the term that accepts iff the input reaches $v$.
- Let $T_v$ be the decision tree rooted at $v$. Thus, $T_v$ agrees with $T$ on inputs that reach $v$.
- Let $T_{-v}$ be $T$ but with $v$'s subtree deleted and with $v$'s parent deleted and replaced with $v$'s sibling. Thus, $T_{-v}$ agrees with $T$ on inputs that don't reach $v$.

This recursive subroutine returns a decision tree $T'$ for $f$ where each internal node can evaluate an arbitrary term:

```
squash(T):
    if T has only one leaf: return T
    if T has ℓ ≥ 2 leaves:
        find a node v whose subtree has ≤ 2ℓ/3 but > ℓ/3 leaves using Lemma 6.23
        return T' whose root queries φ_v and has
            right subtree squash(T_v) and left subtree squash(T_{-v})
```

Since $T_v$ and $T_{-v}$ each have $\leq 2\ell/3$ leaves, the number of levels of recursive calls—not counting base case calls, which correspond to leaves—is $\leq \log_{3/2}(\ell)$. Thus $T'$ has depth $\leq \log_{3/2}(\ell)$, so $\mathrm{Det}^{\mathrm{Term}}(f) \leq \log_{3/2}(\ell)$.

**Exercise 16.32:** $\Rightarrow$: Nothing to prove.

$\Leftarrow$: We prove the contrapositive. Suppose $B$ is not read-once, so some path $P$ rereads some variable. Let $x_i$ be the first variable to be reread along $P$. Since the section of $P$ before the second $x_i$ node reads each variable at most once, some input traverses this section of $P$ up to the second $x_i$ node (but may deviate from $P$ thereafter). On this input, $B$ rereads $x_i$.

**Exercise 16.33:** We prove $\text{ROBPSize}(f) = K \cdot (N-K+1)$.

$\leq$: A straightforward read-once branching program for $f$ reads the input from left to right, remembering how many 1s and how many 0s there have been, and accepts once it has seen $\geq K$ many 1s, and rejects once it has seen $\geq N-K+1$ many 0s. This has a nonsink node for each pair $(k,\ell) \in \{0,\ldots,K-1\} \times \{0,\ldots,N-K\}$ where $k$ is the number of 1s so far, and $\ell$ is the number of 0s so far.

$\geq$: This is essentially the same argument as Theorem 16.44. Consider any read-once branching program $B$ for $f$. Any path from the start node that reads $\leq K-1$ many 1s and $\leq N-K$ many 0s is traversed by some 0-input and some 1-input (since $B$ is read-once) and thus doesn't reach a sink. For each pair $(k,\ell) \in \{0,\ldots,K-1\} \times \{0,\ldots,N-K\}$, let $v_{k,\ell}$ be the nonsink node reached by the path $P_{k,\ell}$ where the edge labels are $k$ many 1s followed by $\ell$ many 0s. We claim the nodes $v_{k,\ell}$ are distinct, so $B$ has at least $K \cdot (N-K+1)$ nonsink nodes. Suppose for contradiction $v_{k,\ell} = v_{k',\ell'}$ for some $(k,\ell) \neq (k',\ell')$. Suppose $k \neq k'$, say $k < k'$. (The case $\ell \neq \ell'$ is analogous.) Let $Q$ be the path from $v_{k,\ell} = v_{k',\ell'}$ where the edge labels are $K-1-k$ many 1s followed by $N-K+1-\ell$ many 0s. The combined path $P_{k,\ell}Q$ reads $K-1$ many 1s and doesn't read $N-K+1$ many 0s until the very last query, so $Q$ must end at the reject node. But since $B$ is read-once, some input traverses the combined path $P_{k',\ell'}Q$ and is rejected despite having $k'+K-1-k \geq K$ many 1s. Thus $B$ doesn't compute $f$.

**Exercise 17.1:** $S = \{(v,v) : v \in V\}$ is a 1-fooling set for $f$ because for all distinct $(u,u),(v,v)$, either $(u,v)$ or $(v,u)$ is a 0-input since either the path from $u$ to $v$ or the path from $v$ to $u$ has length $\geq n/2$ (otherwise the whole cycle would have length $< n/2 + n/2$). Therefore $\mathrm{Det}(f) > \log |S| = \log n$ (Lemma 17.4).

**Exercise 17.2:** To prove that the size of a largest 0-fooling set is $\geq 3$, we exhibit a 0-fooling set of size 3. Let $a, b, c \in \{0,1\}^N$ be any three distinct strings (which exist since $|\{0,1\}^N| \geq 4$ since $N \geq 2$). Let $S = \{(a,b),(b,c),(c,a)\}$. Every $(x,y) \in S$ is a 0-input since $x \neq y$. Also:

- If $(x,y) = (a,b)$ and $(x',y') = (b,c)$ then $(x',y) = (b,b)$ is a 1-input.
- If $(x,y) = (b,c)$ and $(x',y') = (c,a)$ then $(x',y) = (c,c)$ is a 1-input.
- If $(x,y) = (c,a)$ and $(x',y') = (a,b)$ then $(x',y) = (a,a)$ is a 1-input.

To prove that the size of a largest 0-fooling set is $\leq 3$, we consider any $S \subseteq \{0,1\}^N \times \{0,1\}^N$ such that $|S| \geq 4$ and prove that $S$ isn't a 0-fooling set. Suppose for contradiction that $S$ is a 0-fooling set. Let $(x^1,y^1)$, $(x^2,y^2)$, $(x^3,y^3)$, $(x^4,y^4)$ be four distinct inputs in $S$. Every $(x,y) \in S$ is a 0-input, so $x^i \neq y^i$ for all $i$. Also, for all $i \neq j$, either $x^i = y^j$ or $x^j = y^i$. Thus, either $x^1$ equals at least two of $y^2, y^3, y^4$, or $y^1$ equals at least two of $x^2, x^3, x^4$. Assume the former, say $x^1 = y^2 = y^3$. Now, we see that $x^2 \neq y^2 = y^3 \neq x^3$, so we conclude that $x^2 \neq y^3$ and $x^3 \neq y^2$, contradicting the fact that either $x^2 = y^3$ or $x^3 = y^2$.

**Exercise 17.3:** The accepting leaves $v$ of a protocol for $f$ partition $f^{-1}(1)$ into rectangles $R_v$ (Corollary 17.3). Since $1 = D(f^{-1}(1)) = \sum_{\text{accepting leaf } v} D(R_v) \leq \sum_{\text{accepting leaf } v} \delta$, the number of accepting leaves is $\geq 1/\delta$, so the depth is $\geq \log(1/\delta)$. The 1-fooling set method is a special case: If $S$ is a 1-fooling set of size $s$, then letting $D$ be the uniform distribution over $S$, each rectangle $R \subseteq f^{-1}(1)$ contains at most one input in $S$ and therefore $D(R) \leq 1/s$, so every protocol for $f$ has at least $s$ accepting leaves.

**Exercise 17.4.a:** Consider any $f$. For any rectangle $R$, let $q(R)$ be the number of distinct rows in the submatrix corresponding to $R$.

In any protocol for $f$, consider a root-to-leaf path that repeatedly steps to whichever child has more distinct rows in its rectangle (breaking ties arbitrarily). Consider any node $v$ along this path, with children $v_0$ and $v_1$.

- Suppose Bob speaks at $v$, so $R_v$'s columns are partitioned into $R_{v_0}$'s columns and $R_{v_1}$'s columns. Each row of $R_v$ combines a row of $R_{v_0}$ and a row of $R_{v_1}$, so $q(R_v) \leq q(R_{v_0}) \cdot q(R_{v_1})$ and thus either $q(R_{v_0})$ or $q(R_{v_1})$ is $\geq \sqrt{q(R_v)}$.
- Suppose Alice speaks at $v$, so $R_v$'s rows are partitioned into $R_{v_0}$'s rows and $R_{v_1}$'s rows. Either $q(R_{v_0})$ or $q(R_{v_1})$ is $\geq \lceil q(R_v)/2 \rceil$, which is $\geq \sqrt{q(R_v)}$ if $q(R_v) > 2$.

Thus for each non-root node $v$ along the path, $q(R_v) \geq \sqrt{q(R_{v\text{'s parent}})}$ as long as $q(R_{v\text{'s parent}}) > 2$. If the path has $\ell$ edges from the root to the first node $v$ with $q(R_v) \leq 2$ (which must exist since leaf rectangles have only one distinct row), then $2 \geq q(R_v) \geq q(R_{\text{root}})^{1/2^\ell}$. Therefore the protocol has depth $\geq \ell \geq \log\big(\log\big(q(R_{\text{root}})\big)\big)$.

**Exercise 17.4.b:** $\mathrm{Det}^{\rightarrow}(f) \leq 2^{\mathrm{Det}(f)}$ because for each of the $\leq 2^d$ many leaves in a depth-$d$ two-way protocol for $f$, Alice can send Bob a bit indicating whether Alice's input would lead to that leaf, assuming Bob's input does too. That is, for each leaf $v$ with $R_v = A_v \times B_v$, Alice says whether $x \in A_v$. This is enough information for Bob to determine the leaf they would reach (and hence what to output): Among all $v$ with $y \in B_v$, there's a unique $v$ with $x \in A_v$. Thus $\mathrm{Det}(f) \geq \log\big(\mathrm{Det}^{\rightarrow}(f)\big) \geq \log\big(\log(\text{number of distinct rows in } f\text{'s matrix})\big)$ (Observation 17.23).

**Exercise 17.5:** Let $S$ be a set of at most $2^{k_1}$ rectangles whose union equals $f^{-1}(1)$, and label these rectangles with $k_1$-bit strings. Let $T$ be a set of at most $2^{k_0}$ rectangles whose union equals $f^{-1}(0)$. In each iteration, Alice and Bob try to rule out at least half of the remaining rectangles in $T$, meaning they know that the input $(x, y)$ is in none of the ruled-out rectangles.

```
initialize T' ← T
while T' ≠ ∅:
    if an R ∈ S includes row x and is row-disjoint from ≥ half the rectangles in T':
        Alice sends the label of such an R
        remove from T' every rectangle that's row-disjoint from R
    else:
        Alice says there's no such rectangle
        if an R ∈ S includes column y and is column-disjoint from ≥ half the rectangles in T':
            Bob sends the label of such an R
            remove from T' every rectangle that's column-disjoint from R
        else: Bob says there's no such rectangle; reject
accept
```

The depth is $O(k_1 \cdot k_0)$ because in each iteration, $k_1 + O(1)$ bits are communicated, and there are $\leq k_0 + 1$ iterations since in each iteration, at least half of $T'$ is removed (or Alice and Bob halt). This protocol is correct: If $f(x, y) = 0$ with $(x, y) \in Q \in T$, then they reject since $Q$ is never removed from $T'$. If $f(x, y) = 1$ with $(x, y) \in Q \in S$, then they accept because in each iteration, $Q$ is disjoint from all rectangles in $T'$ and is therefore either row-disjoint from at least half the rectangles in $T'$ or column-disjoint from at least half the rectangles in $T'$, so either Alice or Bob will find a suitable $R$.

**Exercise 17.6:** To compute $(f \circ g^N)\big((x^1,\ldots,x^N),(y^1,\ldots,y^N)\big) = f\big(g(x^1,y^1),\ldots,g(x^N,y^N)\big)$, Alice and Bob run a depth-$\mathrm{Det}^{\mathrm{qc}}(f)$ decision tree for $f$, and whenever it queries its input's $i^{\text{th}}$ bit, Alice and Bob run a depth-$\mathrm{Det}^{\mathrm{cc}}(g)$ protocol for $g$ to compute $g(x^i,y^i)$. This protocol for $f \circ g^N$ communicates $\leq \mathrm{Det}^{\mathrm{qc}}(f) \cdot \mathrm{Det}^{\mathrm{cc}}(g)$ bits since each of the $\leq \mathrm{Det}^{\mathrm{qc}}(f)$ queries entails $\leq \mathrm{Det}^{\mathrm{cc}}(g)$ bits of communication.

**Exercise 17.7:** Let $s = \mathrm{DTSize}(f)$. We turn a size-$s$ decision tree for $f$ into a size-$s$ communication protocol for $f$: At a node that queries $x_i$, Alice sends Bob $x_i$. At a node that queries $y_i$, Bob sends Alice $y_i$. By Theorem 17.13, $\mathrm{Det}(f) \leq 2\log_{3/2}(s) = O(\log s)$, so $s \geq 2^{\Omega(\mathrm{Det}(f))}$.

**Exercise 17.8.a:** Let $N$ be this problem's input size. A shallowest protocol for $f$ has size $\leq |X||Y| \leq N$ (at most one leaf per input). At each node in the protocol, the message function also has size $\leq N$. The verifier treats the purported witness as a protocol with description size poly$N$ bits, and accepts iff the protocol computes $f$ and has depth $\leq d$.

**Exercise 17.8.b:** We design a poly-time algorithm to decide whether $f$ (corresponding to the input partial matrix $M$) has a depth-2 protocol in which the root belongs to Alice, meaning she sends the first bit. (Symmetrically, we can also decide whether $f$ has a depth-2 protocol in which the root belongs to Bob.)

If both children of the root belong to Alice, then $f(x, y)$ only depends on $x$, so $f$ actually has a depth-1 protocol. It's straightforward to check whether each row of $M$ is monochromatic.

If one child of the root belongs to Alice and the other to Bob, then within $M$'s bichromatic rows, each column is monochromatic. It's straightforward to check whether this holds.

The most interesting case is when both children of the root belong to Bob. Such a protocol exists for $f$ iff $M$'s rows can be partitioned into two parts such that within each part, each column is monochromatic. Let $G$ be the undirected graph with set of nodes $X$ where $\{x, x'\}$ is an edge iff there exists $y \in Y$ such that $* \neq f(x, y) \neq f(x', y) \neq *$. A suitable partitioning of $M$'s rows corresponds to a proper 2-coloring of $G$. We can decide whether $G$ is properly 2-colorable in poly time (Theorem 2.1).

**Exercise 17.8.c:** We reduce Graph 4-Coloring (which is NP-complete by Corollary 2.26) to this problem (with $d = 3$) by mapping $G = (V, E)$ to the matrix of $f : V \times V \to \{0, 1\}$ where $f(u, v)$ is 1 if $u = v$, is 0 if $\{u, v\} \in E$, and is undefined otherwise. To show this poly-time mapping reduction is correct, we argue that $G$ has a proper 4-coloring iff $\mathrm{Det}(f) \leq 3$:

$\Rightarrow$: Suppose $G$ has a proper 4-coloring $c : V \to \{0, 1\}^2$ (identifying colors with length-2 bit strings). Here's a depth-3 protocol for $f(u, v)$: Alice sends $c(u)$, and then Bob sends the bit indicating whether $c(u) = c(v)$, and they output that bit. If $u = v$ then of course $c(u) = c(v)$, so they accept. If $\{u, v\} \in E$ then $c(u) \neq c(v)$ since $c$ is a proper 4-coloring, so they reject.

$\Leftarrow$: Suppose $f$ has a depth-3 protocol. Without loss of generality, the protocol has eight leaves, of which four accept and four reject, because Alice and Bob can send extra dummy bits if they would have sent fewer than three bits total, and we may assume the last communicated bit is the output. View the four accepting leaves as colors. Define the 4-color assignment $c$ by $c(u) =$ the accepting leaf reached on input $(u, u)$. This is a proper 4-coloring of $G$ because if $\{u, v\} \in E$ then input $(u, v)$ leads to a rejecting leaf, and therefore $(u, u)$ and $(v, v)$ lead to different accepting leaves (since if they led to the same accepting leaf, then $(u, v)$ would also be in the rectangle of inputs leading to that leaf), which means $c(u) \neq c(v)$.

**Exercise 17.9:** Designate an arbitrary node as the root.

Alice says whether $u$ is the root, or a child of the root, or neither
Bob says whether $v$ is the root, or a child of the root, or neither
if $u$ is the root: accept if $v$ is a child of the root, and reject otherwise
if $v$ is the root: accept if $u$ is a child of the root, and reject otherwise
run an EQUALITY protocol on input ($u$, $v$'s parent)
run an EQUALITY protocol on input ($u$'s parent, $v$)
accept iff at least one of those two runs accepted

Assume the EQUALITY protocol has depth $O(1)$ and error $1/6$ on the side of 0-inputs. If $u$ or $v$ is the root, then our protocol is correct, so assume neither $u$ nor $v$ is the root. If $u$ and $v$ are adjacent, then either $u = v$'s parent or $u$'s parent $= v$, so at least one of the two runs of the EQUALITY protocol accepts with probability 1. If $u$ and $v$ aren't adjacent, then $u \neq v$'s parent and $u$'s parent $\neq v$, so each run of the EQUALITY protocol accepts with probability $\leq 1/6$, and thus our protocol accepts with probability $\leq 1/6 + 1/6 = 1/3$.

**Exercise 17.10.a:** $S = \{(x, y) : x_1 = 0 \text{ and } y_1 = 1 \text{ and } x_2 \cdots x_N = y_2 \cdots y_N\}$ is a 1-fooling set of size $2^{N-1}$ because for all distinct $(x, y), (x', y') \in S$, $(x, y')$ (and $(x', y)$, for that matter) is a 0-input since $x_1 \neq y'_1$ and $x_i \neq y'_i$ for some $i > 1$ and thus $\text{dist}(x, y') \geq 2$. By Lemma 17.4, $\text{Det}(f) > N - 1$.

**Exercise 17.10.b:** Consider the protocol that samples a uniformly random $S \subseteq [N]$ (that is, $\mathbf{Pr}[i \in S] = 1/2$ for each $i \in [N]$ independently), runs an EQUALITY protocol on $(x_S, y_S)$ and on $(x_{\overline{S}}, y_{\overline{S}})$, and accepts iff at least one of those two runs accepts. Assume the EQUALITY protocol has depth $O(1)$ and one-sided error $1/2$ on the side of 0-inputs. Then our protocol for $f$ has depth $O(1)$ and one-sided error $7/8$ on the side of 0-inputs (which can be amplified): If $f(x, y) = 1$ then certainly either $x_S = y_S$ or $x_{\overline{S}} = y_{\overline{S}}$, so $\mathbf{Pr}[\text{accept}] = 1$. If $f(x, y) = 0$ then for some $i \neq j$ we have $x_i \neq y_i$ and $x_j \neq y_j$, and with probability $1/2$, one of $i$ or $j$ is in $S$ and the other is in $\overline{S}$, in which case $x_S \neq y_S$ and $x_{\overline{S}} \neq y_{\overline{S}}$ and so each run of the EQUALITY protocol rejects with probability $\geq 1/2$. Therefore:

$$\begin{aligned}\mathbf{Pr}[\text{reject}] \;\geq\; & \mathbf{Pr}\left[x_S \neq y_S \text{ and } x_{\overline{S}} \neq y_{\overline{S}}\right] \cdot \\ & \mathbf{Pr}\left[\text{EQUALITY protocol rejects } (x_S, y_S) \,\middle|\, x_S \neq y_S\right] \cdot \\ & \mathbf{Pr}\left[\text{EQUALITY protocol rejects } (x_{\overline{S}}, y_{\overline{S}}) \,\middle|\, x_{\overline{S}} \neq y_{\overline{S}}\right] \\ \geq\; & \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \;=\; 1/8\end{aligned}$$

**Exercise 17.10.c:** Assume $N$ is a power of 2, and identify $[N] = \{0,1\}^n$ where $n = \log N$. Let $h\colon \{0,1\}^{n+1} \times \{0,1\}^n \to \{0,1\}$ be the pairwise uniform hash function from Lemma 8.7. Sample $S \subseteq [N]$ by sampling uniformly random $r \in \{0,1\}^{n+1}$ and letting $S = \{i : h_r(i) = 1\}$. Then for all $i \neq j$, we have $\mathbf{Pr}\big[i, j \in S \text{ or } i, j \in \overline{S}\big] = \mathbf{Pr}\big[h_r(i) = h_r(j)\big] = 1/2$.

**Exercise 17.11:** Let $E\colon \{0,1\}^N \to \{0,1\}^M$ be an error correcting code with constant relative distance $\delta = \Omega(1)$ and constant rate $N/M = \Omega(1)$, where $M$ is a perfect square. The inputs $x, y \in \{0,1\}^N$ are messages (for the code). View the codewords $E(x), E(y)$ as $\sqrt{M} \times \sqrt{M}$ binary matrices.

- Alice picks a uniformly random $i \in [\sqrt{M}]$ and sends the referee $i$ and $E(x)$'s $i^{\text{th}}$ row.
- Bob picks a uniformly random $j \in [\sqrt{M}]$ and sends the referee $j$ and $E(y)$'s $j^{\text{th}}$ column.
- The referee rejects iff $E(x)_{i,j} \neq E(y)_{i,j}$.

If $x = y$ then $\mathbf{Pr}[\text{accept}] = 1$ since $E(x) = E(y)$ and thus $E(x)_{i,j} = E(y)_{i,j}$. If $x \neq y$ then $\mathbf{Pr}[\text{reject}] \geq \delta$ since $E(x)$ and $E(y)$ disagree on $\geq \delta$ fraction of coordinates, and $(i,j)$ is uniformly random over all coordinates. This can be amplified. The number of bits communicated is $O(\log \sqrt{M} + \sqrt{M}) = O(\sqrt{N})$ since $M = O(N)$.

**Exercise 17.12.a:** $\text{Rand}_1^{\text{pub}}(f) > d$ if there exists a distribution $D$ over 1-inputs of $f$ such that for every depth-$d$ deterministic protocol $T$ that rejects all 0-inputs of $f$, $\mathbf{Pr}_{(x,y)\sim D}\big[T(x,y)=0\big] > 1/3$.

We prove the contrapositive. Suppose $\text{Rand}_1^{\text{pub}}(f) \leq d$ by some depth-$d$ public-coin protocol, which is a distribution $P$ over depth-$d$ deterministic protocols that reject all 0-inputs of $f$. Consider any distribution $D$ over 1-inputs of $f$. Taking probabilities over $T \sim P$ and $(x,y) \sim D$:

$$\min_T\big(\mathbf{Pr}_{x,y}\big[T(x,y)=0\big]\big) \;\leq\; \mathbf{Pr}_{T,(x,y)}\big[T(x,y)=0\big] \;\leq\; \max_{x,y}\big(\mathbf{Pr}_T\big[T(x,y)=0\big]\big) \;\leq\; 1/3$$

Thus for some depth-$d$ deterministic protocol $T$ that rejects all 0-inputs of $f$, $\mathbf{Pr}_{x,y}\big[T(x,y)=0\big] \leq 1/3$.

**Exercise 17.12.b:** Let $d = \log(2s/3)$. Suppose for contradiction that $S$ is a 1-fooling set of size $s$ for $f$, but $\text{Rand}_1^{\text{pub}}(f) \leq d$. Let $D$ be the uniform distribution over $S$. By Exercise 17.12.a, there exists a depth-$d$ deterministic protocol $T$ that rejects all 0-inputs of $f$, and such that $\mathbf{Pr}_{(x,y)\sim D}\left[T(x,y)=0\right] \leq 1/3$. Thus the accepting leaves' rectangles contain $\geq 2/3$ fraction of $S$. No two inputs $(x,y),(x',y') \in S$ can be in the same accepting leaf's rectangle, because (like in the proof of Lemma 17.4) otherwise $(x,y'),(x',y)$ would both be in the same rectangle and be accepted, contradicting the assumption that one of them is a 0-input and is therefore rejected. Thus $T$ has $\geq 2s/3$ many accepting leaves and at least one rejecting leaf, and so has size $> 2s/3$ and depth $> \log(2s/3) = d$, contradicting the assumption that $T$ is depth-$d$.

Thus $\text{Rand}_1^{\text{pub}}(\textsc{Equality}) > \log(2 \cdot 2^N/3) > \log(2^{N-1}) = N-1$, so $\text{Rand}_1^{\text{pub}}(\textsc{Equality}) \geq N$, since $\text{EQ}_N$ has a 1-fooling set of size $2^N$ (Theorem 17.5).

**Exercise 17.13.a:** Let $f = \textsc{Disj}_N$ and $d = \lceil 4\varepsilon N \rceil$. Alice and Bob pick a uniformly random $i \in [N]$ and define $I = \{i, i+1, i+2, \ldots, i+d-1\}$ where indices wrap around (so $N+1$ means 1). Alice sends Bob $x_I \in \{0,1\}^d$. Bob tells Alice to reject iff $x_j = y_j = 1$ for some $j \in I$. If $f(x,y) = 1$ then $\mathbf{Pr}[\text{accept}] = 1$. If $f(x,y) = 0$ then $\mathbf{Pr}[\text{reject}] \geq d/N \geq 4\varepsilon$. Consider the protocol where Alice and Bob reject with probability $1/2 - \varepsilon$ (without communicating) and otherwise run the above protocol. If $f(x,y) = 1$ then $\mathbf{Pr}[\text{accept}] = 1/2 + \varepsilon$. If $f(x,y) = 0$ then $\mathbf{Pr}[\text{reject}] \geq (1/2 - \varepsilon) + (1/2 + \varepsilon)4\varepsilon \geq 1/2 + \varepsilon$. The depth is $d + 1 = O(\varepsilon N)$ since $\varepsilon \geq 1/N$.

**Exercise 17.13.b:** If $f = \textsc{Disj}_N$ has a depth-$d$ $(1/2-\varepsilon)$-error protocol, then running it $O(1/\varepsilon^2)$ times independently and outputting the majority vote yields a depth-$O(d/\varepsilon^2)$ $1/3$-error protocol for $f$ (§7.3.2). Since any such protocol has depth $\Omega(N)$ (Theorem 17.22), we see that $d/\varepsilon^2 \geq \Omega(N)$ and therefore $d \geq \Omega(\varepsilon^2 N)$.

**Exercise 17.13.c:** Run the original protocol $k$ times independently and accept iff all runs accept.

**Exercise 17.13.d:** This follows from Exercise 17.13.c since:

$$(1/2-\varepsilon)^k = \left(\frac{1/2-\varepsilon}{1/2+\varepsilon}\right)^k \alpha \leq (1-2\varepsilon)^k \alpha \leq e^{-2\varepsilon k}\alpha \leq (c/4)\alpha$$

**Exercise 17.13.e:** Suppose $f$ has a $(dk, \alpha, (c/4)\alpha)$-protocol. For each outcome $r \in \{0,1\}^m$ of the coin tosses, let $T_r$ be the corresponding deterministic depth-$dk$ protocol. Consider any distributions $D_1$ and $D_0$ over $f^{-1}(1)$ and $f^{-1}(0)$. Let $p_{1,r} = \mathbf{Pr}_{(x,y)\sim D_1}\big[T_r(x,y)=1\big]$ and $p_{0,r} = \mathbf{Pr}_{(x,y)\sim D_0}\big[T_r(x,y)=1\big]$.

$$\begin{aligned} \tfrac{1}{2^m}\textstyle\sum_r p_{1,r} &= \mathbf{Pr}_{r,(x,y)\sim D_1}\big[T_r(x,y)=1\big] \geq \min_{(x,y)\in f^{-1}(1)} \mathbf{Pr}_r\big[T_r(x,y)=1\big] \geq \alpha \\ \tfrac{1}{2^m}\textstyle\sum_r p_{0,r} &= \mathbf{Pr}_{r,(x,y)\sim D_0}\big[T_r(x,y)=1\big] \leq \max_{(x,y)\in f^{-1}(0)} \mathbf{Pr}_r\big[T_r(x,y)=1\big] \leq (c/4)\alpha \end{aligned}$$

Suppose for contradiction that for every $r$, either $p_{1,r} < \alpha/2$ or $p_{0,r} > (c/2)p_{1,r}$. Let $S = \big\{r : p_{0,r} > (c/2)p_{1,r}\big\}$.

$$\begin{aligned} \tfrac{1}{2^m}\textstyle\sum_r p_{0,r} &\geq \tfrac{1}{2^m}\textstyle\sum_{r\in S} p_{0,r} \\ &\geq \tfrac{1}{2^m}\textstyle\sum_{r\in S}(c/2)p_{1,r} \\ &= (c/2)\big(\tfrac{1}{2^m}\textstyle\sum_r p_{1,r} - \tfrac{1}{2^m}\textstyle\sum_{r\notin S} p_{1,r}\big) \\ &\geq (c/2)\big(\alpha - \alpha/2\big) \\ &= (c/4)\alpha \end{aligned}$$

At least one of these inequalities must be strict, yielding the contradiction $\frac{1}{2^m}\sum_r p_{0,r} > (c/4)\alpha$.

**Exercise 17.13.f:** For each leaf $v$ of $T$, let $R_v$ be the rectangle of inputs that lead to $v$. The accepting leaves $v$ partition $A$ into at most $2^{dk}$ rectangles $R_v$.

$$\begin{aligned} D_0(A) &= \textstyle\sum_{\text{accepting leaf } v} D_0(R_v) \\ &\geq \textstyle\sum_{\text{accepting leaf } v} \left(cD_1(R_v) - 1/2^{cN-1}\right) && \text{(Lemma 17.21)} \\ &\geq c\textstyle\sum_{\text{accepting leaf } v} D_1(R_v) - 2^{dk}/2^{cN-1} \\ &= cD_1(A) - 2^{dk}/2^{cN-1} \end{aligned}$$

**Exercise 17.13.g:** Suppose $f = \textsc{Disj}_N$ has a $(d, 1/2+\varepsilon, 1/2-\varepsilon)$-protocol. Let $c$ and $D = D_1/2 + D_0/2$ be from Lemma 17.21.

- By Exercise 17.13.d, $f$ has a $(dk, \alpha, (c/4)\alpha)$-protocol where $k = \lceil \ln(4/c)/2\varepsilon \rceil$ and $\alpha = (1/2+\varepsilon)^k$.
- By Exercise 17.13.e, there exists a depth-$dk$ deterministic protocol $T$ such that $D_1(A) \geq \alpha/2$ and $D_0(A) \leq (c/2)D_1(A)$ where $A = T^{-1}(1)$ is the event that $T$ accepts.
- By Exercise 17.13.f, $D_0(A) \geq cD_1(A) - 2^{dk}/2^{cN-1}$.

Combining these, we get $(c/2)D_1(A) \geq cD_1(A) - 2^{dk}/2^{cN-1}$. Then rearranging yields $2^{dk} \geq 2^{cN-1}(c/2)D_1(A) \geq 2^{cN-3}c\alpha$ and thus $dk \geq cN - 3 + \log(c\alpha) \geq cN - 3 + \log c - k$. It follows that $d \geq cN/k - O(1) = \Omega(\varepsilon N)$.

**Exercise 17.14.a:** The proof of $\mathrm{Det}(F) \leq 2^{O(\mathrm{Rand}^{\mathrm{priv}}_{0,1}(F))}$ (Theorem 17.16) yields a one-way protocol, so $N \leq \mathrm{Det}^{\rightarrow}(\textsc{Index}) \leq 2^{O(\mathrm{Rand}^{\mathrm{priv}}_{0,1}(\textsc{Index}))}$.

**Exercise 17.14.b:** Every $f : [N] \times [N] \to \{0,1\}$ has a rectangular reduction to $\textsc{Index}_N$ that maps $(x,y)$ to $(x',y)$ where $x' = f(x,1)f(x,2)\cdots f(x,N) \in \{0,1\}^N$ since $f(x,y) = x'_y$. With $f = \mathrm{IP}_{\log N}$, we get $\mathrm{Rand}^{\mathrm{pub}}_{0,1}(\textsc{Index}_N) \geq \mathrm{Rand}^{\mathrm{pub}}_{0,1}(\mathrm{IP}_{\log N}) \geq \Omega(\log N)$ (Theorem 17.19).

**Exercise 17.15:** Let $G$ be the $n$-node complete graph, with edges between all pairs of nodes. Alice's clique is any subset of nodes, which we identify with its characteristic bit string $x \in \{0,1\}^n$. Bob's independent set is any individual node, which we view as an index $y \in [n]$ (ignoring the possibility that Bob gets the empty set). In this case, $\text{CIS}_G$ is equivalent to $\text{INDEX}_n$ since Bob's node is in Alice's set of nodes iff $x_y = 1$. By Theorem 17.25, $\text{Rand}_{0,1}^{\text{pub},\rightarrow}(\text{CIS}_G) \geq \text{Rand}_{0,1}^{\text{pub},\rightarrow}(\text{INDEX}_n) \geq \Omega(n)$. Also, $\text{Rand}_{0,1}^{\text{pub},\rightarrow}(\text{CIS}_G) \leq n$ trivially holds.

**Exercise 17.16.a:** $\leq$: Let $N$ be the VC dimension, so some $S \subseteq Y$ of size $N$ is shattered by $\{f^x : x \in X\}$, which means every $g : S \to \{0,1\}$ agrees with some $f^x$ on $S$. Say $S = \{y^1, y^2, \ldots, y^N\}$. We define a rectangular reduction from $\textsc{Index}_N$ to $f$: Map $(a,b) \in \{0,1\}^N \times [N]$ to $(x, y^b)$ such that $f^x$ agrees with $g : S \to \{0,1\}$ defined by $g(y^1)g(y^2)\cdots g(y^N) = a$. This works because $\textsc{Index}_N(a,b) = a_b = g(y^b) = f^x(y^b) = f(x, y^b)$.

$\geq$: Suppose there exists a rectangular reduction $(h_{\text{Alice}}, h_{\text{Bob}})$ from $\textsc{Index}_N$ to $f$. To show the VC dimension is $\geq N$, we show that some $S \subseteq Y$ of size $N$ is shattered by $\{f^x : x \in X\}$. Let $S = \{y^1, y^2, \ldots, y^N\}$ where $y^b = h_{\text{Bob}}(b)$ for all $b \in [N]$. Then $S$ has size $N$ because if $b \neq b'$ then $y^b \neq y^{b'}$ since for some $a \in \{0,1\}^N$ we have $a_b \neq a_{b'}$ and thus $f(x, y^b) \neq f(x, y^{b'})$ where $x = h_{\text{Alice}}(a)$. To see that $S$ is shattered, consider any $g : S \to \{0,1\}$. Define $a = g(y^1)g(y^2)\cdots g(y^N) \in \{0,1\}^N$ and $x = h_{\text{Alice}}(a)$. Then $g$ agrees with $f^x$ since $g(y^b) = a_b = \textsc{Index}_N(a,b) = f(x, y^b) = f^x(y^b)$ for all $b \in [N]$.

**Exercise 17.16.b:** Let $N$ be the VC dimension. Since there exists a rectangular reduction from $\textsc{Index}_N$ to $f$ (Exercise 17.16.a), we have $\mathrm{Rand}_{0,1}^{\mathrm{pub},\rightarrow}(f) \geq \mathrm{Rand}_{0,1}^{\mathrm{pub},\rightarrow}(\textsc{Index}_N) = \Omega(N)$ (Theorem 17.25) because we could turn any public-coin one-way protocol for $f$ into a public-coin one-way protocol for $\textsc{Index}_N$ without increasing the depth or the error probability: On input $(a, b) \in \{0,1\}^N \times [N]$, Alice and Bob run the protocol for $f$ on input $\left(h_{\mathrm{Alice}}(a), h_{\mathrm{Bob}}(b)\right)$.

**Exercise 17.16.c:** Big-$O$: On input $(x, y)$, Bob uses $O(\log N)$ bits to send $y$ to Alice, who outputs $x_y$.

Big-$\Omega$: By Exercise 17.16.b (interchanging the roles of Alice and Bob), for all $f : X \times Y \to \{0,1\}$ we have $\mathrm{Rand}_{0,1}^{\mathrm{pub},\leftarrow}(f) \geq \Omega\big(\text{VC dimension of } \{f^y : y \in Y\}\big)$ where $f^y : X \to \{0,1\}$ is defined by $f^y(x) = f(x,y)$. For $f = \textsc{Index}_N$, this concept class is dictatorships. The VC dimension is $\lfloor \log N \rfloor$ (Lemma 13.12).

**Exercise 17.17:** Assuming $f$ has a public-coin $(a,b)$-protocol with one-sided error $\varepsilon$ on the side of 1-inputs and $f$ is $(s,t)$-rich, we prove that $f$ has a 1-monochromatic $\left\lceil (1-\sqrt{\varepsilon})s/2^{a+b} \right\rceil \times \left\lceil (1-\sqrt{\varepsilon})t/2^{b} \right\rceil$ rectangle.

Since $f$ is $(s,t)$-rich, there exists a set $S$ of $s \cdot t$ many 1-inputs, consisting of exactly $t$ many 1-inputs in each of exactly $s$ many rows. Let $D$ be the uniform distribution over $S$. Like in Exercise 17.12.a, there exists a deterministic $(a,b)$-protocol $T$ that rejects all 0-inputs of $f$ and such that $\mathbf{Pr}_{(x,y)\sim D}\left[T(x,y)=0\right] \leq \varepsilon$. Since $T$ accepts $\geq 1-\varepsilon$ fraction of $S$, there exist $\geq (1-\sqrt{\varepsilon})s$ many rows each of which has $\geq (1-\sqrt{\varepsilon})t$ many inputs that $T$ accepts (by contrapositive reasoning). In other words, the function computed by $T$ is $\left((1-\sqrt{\varepsilon})s, (1-\sqrt{\varepsilon})t\right)$-rich. By Lemma 17.27, the function computed by $T$ has a 1-monochromatic $\left\lceil (1-\sqrt{\varepsilon})s/2^{a+b} \right\rceil \times \left\lceil (1-\sqrt{\varepsilon})t/2^{b} \right\rceil$ rectangle. This rectangle is also 1-monochromatic for $f$, because $T$ rejects all 0-inputs of $f$.

**Exercise 17.18:** Identify the bit strings $x$ and $y$ with the sets $\{i : x_i = 1\}$ and $\{i : y_i = 1\}$, so weight corresponds to size.

Let $a = k/4$ and $b = \ell/4$ and $s = \binom{N}{k}$ and $t = \binom{N-k}{\ell}$.

$f$ is $(s,t)$-rich because each of the $s$ many size-$k$ subsets of $[N]$ is disjoint from $t$ many size-$\ell$ subsets of $[N]$.

We show that $f$ has no 1-monochromatic $\lceil s/2^{a+b} \rceil \times \lceil t/2^b \rceil$ rectangle. Consider any $\lceil s/2^{a+b} \rceil \times \lceil t/2^b \rceil$ rectangle $A \times B$. Note that

$$|A| \geq s/2^{k/2} = \binom{N}{k}/\sqrt{2}^k \geq \binom{N/\sqrt{2}}{k} > \binom{0.7N}{k}$$

since $\ell \leq k$ and $k < 0.7N$, and

$$|B| \geq \binom{N-k}{\ell}/\sqrt[4]{2}^\ell \geq \binom{(N-k)/\sqrt[4]{2}}{\ell} \geq \binom{N/2\sqrt[4]{2}}{\ell} > \binom{0.4N}{\ell}$$

since $k \leq N/2$ and $\ell < 0.4N$. The union of all $x \in A$ (viewing $x$ as a subset of $[N]$) has size $> 0.7N$ since otherwise, some set of size $0.7N$ would be a superset of every $x \in A$, implying $|A| \leq \binom{0.7N}{k}$. The union of all $y \in B$ (viewing $y$ as a subset of $[N]$) has size $> 0.4N$ since otherwise, some set of size $0.4N$ would be a superset of every $y \in B$, implying $|B| \leq \binom{0.4N}{\ell}$. Since $0.7N + 0.4N > N$, there exists $i \in [N]$ that's in both the union of all $x \in A$ and the union of all $y \in B$. That is, for some $x \in A$ and some $y \in B$, we have $i \in x$ and $i \in y$ and therefore $f(x,y) = 0$. Thus $A \times B$ is not 1-monochromatic.

By Lemma 17.27, $f$ has no $(a,b)$-protocol.

**Exercise 17.19:** We reduce from DISJOINTNESS. Suppose $\Pi$ is a $p$-pass space-$s$ randomized streaming algorithm that approximates the highest frequency within factor $< \sqrt{2}$ when the elements are from $[n]$. We turn $\Pi$ into a depth-$2ps$ public-coin two-way protocol for $\text{DISJ}_n$, which implies $2ps \geq \Omega(n)$ (Theorem 17.22), so $s \geq \Omega(n)$ if $p$ is a constant. On input $(x, y) \in \{0,1\}^n \times \{0,1\}^n$: Alice runs $\Pi$ with the stream being the indices $i_1, i_2, \ldots, i_k$ of 1s in $x$ (using public coins for $\Pi$'s randomness), then sends Bob $\Pi$'s memory contents. Bob continues running $\Pi$ with the indices $j_1, j_2, \ldots, j_\ell$ of 1s in $y$ appended to the stream, then sends Alice $\Pi$'s memory contents. They repeat this for each subsequent pass that $\Pi$ takes. Bob accepts (and tells Alice to accept) iff $\Pi$'s output is $\leq \sqrt{2}$. This is correct because:

- If $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_\ell\} = \emptyset$ then $(i_1, \ldots, i_k, j_1, \ldots, j_\ell)$ has highest frequency 1, so $\Pi$'s output is $< \sqrt{2} \cdot 1 = \sqrt{2}$ with probability $\geq 2/3$.
- If $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_\ell\} \neq \emptyset$ then $(i_1, \ldots, i_k, j_1, \ldots, j_\ell)$ has highest frequency 2, so $\Pi$'s output is $> 2/\sqrt{2} = \sqrt{2}$ with probability $\geq 2/3$.

The depth is $\leq 2ps$ since for each of the $p$ passes, Alice sends $\leq s$ bits and Bob sends $\leq s$ bits.

**Exercise 17.20:** Let $f = \text{Opposite Parity}_N$ with domain $X \times Y$. Let $D$ be the uniform distribution over the set $Z$ of all inputs $(x, y)$ with $\text{dist}(x, y) = 1$. Like Lemma 17.17, it suffices to prove that every deterministic protocol with $\mathbf{Pr}_{(x,y)\sim D}[\text{output is an acceptable solution}] \geq 2/3$ has depth $\geq 2 \log N - O(1)$. Consider any such depth-$d$ size-$s$ protocol. For each leaf $v$, let $R_v$ be the rectangle of inputs that lead to $v$. Let $Z'$ be the set of inputs in $Z$ on which the protocol outputs an acceptable solution, so $|Z'| \geq (2/3)|Z| = (2/3)N2^{N-1}$. Like in the proof of Theorem 17.33, no two inputs in $Z' \cap R_v$ are in the same row or in the same column. Letting $z_v = |Z' \cap R_v|$, this implies $R_v$ has $\geq z_v$ rows and $\geq z_v$ columns and therefore $\geq z_v^2$ inputs. By Corollary 0.27:

$$|X \times Y| = \sum_{\text{leaf } v} |R_v| \geq \sum_{\text{leaf } v} z_v^2 \geq \left(\sum_{\text{leaf } v} z_v\right)^2 / s = |Z'|^2/s$$

We conclude that

$$s \geq |Z'|^2/|X \times Y| = ((2/3)N2^{N-1})^2/(2^{N-1})^2 = (4/9)N^2$$

so the protocol has depth $d \geq \log s \geq 2 \log N - \log(9/4)$.

**Exercise 17.21:** $\mathrm{Det}(\textsc{Fork}_{2,n}) \geq \mathrm{Det}(\textsc{Opposite Parity}_{n+1}) \geq 2\log(n+1)$ (Theorem 17.33's lower bound holds even when $n+1$ isn't a power of 2) since we can turn any protocol $\Pi$ for $\textsc{Fork}_{2,n}$ into a same-depth protocol $\Pi'$ for $\textsc{Opposite Parity}_{n+1}$ where $\Pi'(x,y)$ runs $\Pi(a,b)$ and adds 1 to the output, where $a_i = x_1 \oplus x_2 \oplus \cdots \oplus x_i$ and $b_i = y_1 \oplus y_2 \oplus \cdots \oplus y_i$ for each $i \in \{1,\ldots,n\}$ (and implicitly, $a_0 = b_0 = 0$ and $a_{n+1} = x_1 \oplus \cdots \oplus x_{n+1} = 1$ and $b_{n+1} = y_1 \oplus \cdots \oplus y_{n+1} = 0$). This works because $\Pi(a,b)$ outputs a fork $i \in \{0,\ldots,n\}$ for $(a,b)$, which means $a_i = b_i$ and $a_{i+1} \neq b_{i+1}$, which implies $1 = a_{i+1} \oplus b_{i+1} = (a_i \oplus x_{i+1}) \oplus (b_i \oplus y_{i+1}) = x_{i+1} \oplus y_{i+1}$, so $x_{i+1} \neq y_{i+1}$ and thus $i+1 \in \{1,\ldots,n+1\}$ is an acceptable solution.

**Exercise 17.22:** This is like the proof that $\mathrm{Det}(\textsc{Fork}) \geq \Omega\big((\log k)(\log n)\big)$ (Theorem 17.34). In the three places where the definition of "fork" is relevant, the switch to "undirected fork" makes virtually no difference:

In Lemma 17.35's counterpart: On input $(x, x) \in S \times S$, the only undirected fork is $n$. On input $(x, y) \in S \times S$, $n$ isn't an undirected fork since $x_n \neq y_n$ and $x_{n+1} \neq y_{n+1}$. Thus $S \times S$ is not monochromatic.

In Lemma 17.37's counterpart, when $|V_u| \geq \sqrt{\delta/2}k^{n/2}$: $\Pi_u$ computes $\textsc{Undirected Fork}_{V_u}$ because for all $x, y \in V_u$ (with $x_0 = y_0 = u_{n/2}$ implicitly), we have $ux, uy \in S$ and so $\Pi(ux, uy)$ outputs an undirected fork $i \in \{0, \ldots, n\}$ for $(ux, uy)$, which means either $(ux)_i = (uy)_i$ and $(ux)_{i+1} \neq (uy)_{i+1}$ or $(ux)_i \neq (uy)_i$ and $(ux)_{i+1} = (uy)_{i+1}$, which implies $i \in \{n/2, \ldots, n\}$ and either $x_{i-n/2} = y_{i-n/2}$ and $x_{i-n/2+1} \neq y_{i-n/2+1}$ or $x_{i-n/2} \neq y_{i-n/2}$ and $x_{i-n/2+1} = y_{i-n/2+1}$, so $i - n/2$ is an undirected fork for $(x, y)$.

In Lemma 17.37's counterpart, when $|U| \geq \sqrt{\delta/2}k^{n/2}$: $\Pi'$ computes $\textsc{Undirected Fork}_{U'}$ because for all $x, y \in U'$ (with $x_{n/2+1} = v_1^x \neq w_1^y = y_{n/2+1}$ implicitly), we have $xv^x, yw^y \in S$ and so $\Pi(xv^x, yw^y)$ outputs an undirected fork $i \in \{0, \ldots, n\}$ for $(xv^x, yw^y)$, which means either $(xv^x)_i = (yw^y)_i$ and $(xv^x)_{i+1} \neq (yw^y)_{i+1}$ or $(xv^x)_i \neq (yw^y)_i$ and $(xv^x)_{i+1} = (yw^y)_{i+1}$, which implies $i \in \{0, \ldots, n/2\}$ (since $v^x$ and $w^y$ differ in every coordinate) and either $x_i = y_i$ and $x_{i+1} \neq y_{i+1}$ or $x_i \neq y_i$ and $x_{i+1} = y_{i+1}$, so $i$ is an undirected fork for $(x, y)$.

**Exercise 17.23.a:** As in the randomized upper bound for GREATER THAN (Theorem 17.14), Alice and Bob use binary search with an EQUALITY oracle to find the first $i$ such that $x_i \neq y_i$. Plugging in an amplified EQUALITY protocol yields $\text{Rand}^{\text{pub}}(\text{UNIVERSAL SEARCH}) \leq O(\log N \log\log N)$.

**Exercise 17.23.b:** $\text{Det}(\text{EQUALITY}_N) \leq \text{Det}(\text{UNIVERSAL SEARCH}_N) + 2$: To compute $\text{EQUALITY}_N$ on input $(x, y)$, Alice and Bob run a protocol for $\text{UNIVERSAL SEARCH}_N$ on input $(x, y)$ to get an index $i$, and then exchange $x_i$ and $y_i$ (two more bits of communication) and reject iff $x_i \neq y_i$. This works because if $x = y$ then even though $(x, y)$ is an invalid input to $\text{UNIVERSAL SEARCH}_N$, Alice and Bob accept since $x_i = y_i$ regardless of $i$, and if $x \neq y$ then the $\text{UNIVERSAL SEARCH}_N$ protocol finds $i$ such that $x_i \neq y_i$, so Alice and Bob reject. Thus $\text{Det}(\text{UNIVERSAL SEARCH}_N) \geq \text{Det}(\text{EQUALITY}_N) - 2 = N - 1$ (Theorem 17.5).

**Exercise 18.1:** This is like Theorem 18.1. For each remainder $r \in \{0,1,2\}$, define $F^r : \{0,1\}^+ \to \{0,1\}$ by $F^r(x) = 1$ iff weight$(x) \bmod 3 = r$, so $F = F^0$.

This is trivial for $N \leq 2$, so assume $N \geq 3$. Consider any circuit $C$ for $F_N^0$ with as few $\wedge/\vee$ gates as possible. Consider the first $\wedge/\vee$ gate $u$ in some topological order. Two possibly-negated variables $x_i$ and $x_j$ feed into $u$. Either $x_i$ or $x_j$, possibly negated, also feeds into another $\wedge/\vee$ gate $v$, because otherwise the three partial assignments with $x_i x_j \in \{00, 01, 11\}$ would yield only two contractions of $C$ (in which $u$ becomes constant 0 or 1) but three restrictions of $F_N^0$ (namely $F_{N-2}^0$, $F_{N-2}^2$, $F_{N-2}^1$, which are different since $N \geq 3$). Say $x_j$, possibly negated, feeds into $v$. Assigning $x_j \leftarrow 0$ and contracting $C$ eliminates $u$ and $v$, yielding a circuit for $F_{N-1}^0$ with at least two fewer $\wedge/\vee$ gates. Repeating this until two variables remain eliminates $\geq 2(N-2)$ many $\wedge/\vee$ gates. Then, an $\wedge/\vee$ gate remains for $F_2^0$. Thus $C$ has $\geq 2(N-2)+1$ many $\wedge/\vee$ gates.

**Exercise 18.2:** This is like Theorem 18.2. Consider any circuit $C$ for $\mathrm{EQ}_N$ with as few $\wedge/\vee$ gates as possible. Consider the first $\wedge/\vee$ gate $u$ in some topological order. Two possibly-negated variables $x_i$ and $x_j$ feed into $u$. Then $x_j$, possibly negated, also feeds into another $\wedge/\vee$ gate $v$, because otherwise assigning $x_i$ a certain bit and contracting $C$ would make $u$ a constant and thus yield a circuit that doesn't depend on $x_j$, which can't happen for $\mathrm{EQ}_N$ (since restricting $\mathrm{EQ}_N$ by assigning one variable yields a function that still depends on all other variables). Neither $v$ nor $\neg v$ is the output, because assigning $x_j$ a certain bit would make $v$ a constant, which can't happen for $\mathrm{EQ}_N$. Thus $v$, possibly negated, feeds into an $\wedge/\vee$ gate $w$, and $w \neq u$ since $u$ is the first $\wedge/\vee$ gate in the topological order. Assigning $x_j$ a certain bit and contracting $C$ makes $v$ a constant and eliminates $u, v, w$. The resulting circuit must still depend on $x_j$'s *partner* ($x_{j+N/2}$ if $j \leq N/2$, or $x_{j-N/2}$ if $j > N/2$), and assigning $x_j$'s partner the same bit as $x_j$ and contracting eliminates another $\wedge/\vee$ gate (if $N \geq 4$), yielding a circuit for $\mathrm{EQ}_{N-2}$ with at least four fewer $\wedge/\vee$ gates than $C$. Repeating this until one pair of variables remains eliminates $\geq 4(N/2-1) = 2N-4$ many $\wedge/\vee$ gates. After eliminating three more $\wedge/\vee$ gates by assigning one more variable, the circuit might have no more $\wedge/\vee$ gates. Thus $C$ has $\geq 2N-1$ many $\wedge/\vee$ gates.

By the way, this lower bound is exactly tight: The obvious circuit for $\mathrm{EQ}_N$ has only $2N-1$ many $\wedge/\vee$ gates.

**Exercise 18.3:** We may assume $N$ is a power of 2. Abbreviate 2-THRESHOLD$_N$ as $F_N^2$. To obtain a monotone formula $\varphi_N$ for $F_N^2$, recursively obtain a monotone formula $\varphi_{N/2}$ for $F_{N/2}^2$ and define:

$$\varphi_N(x) = \varphi_{N/2}(x_1 \cdots x_{N/2}) \vee \varphi_{N/2}(x_{N/2+1} \cdots x_N) \vee \big((x_1 \vee \cdots \vee x_{N/2}) \wedge (x_{N/2+1} \vee \cdots \vee x_N)\big)$$

For the base case, $\varphi_2(x_1 x_2) = x_1 \wedge x_2$ has 2 leaves. Note that $\varphi_N$ directly contributes $N$ leaves, and the 2 copies of $\varphi_{N/2}$ each contribute $N/2$ leaves (totalling $2 \cdot N/2 = N$ leaves for this level), and the 4 copies of $\varphi_{N/4}$ each contribute $N/4$ leaves (totalling $4 \cdot N/4 = N$ leaves for this level), and so on. Each of the $\log N$ levels contributes $N$ leaves, so $\varphi_N$ has $N \log N$ leaves in all.

**Exercise 18.4.a:** We design a "low-degree mapping reduction" from $f\colon \{0,1\}^m \to \{0,1\}$ to $\vee\colon \{0,1\}^{\binom{m}{2}} \to \{0,1\}$: Let $g\colon \{0,1\}^m \to \{0,1\}^{\binom{m}{2}}$ be such that the bits of $g(y)$ are the "and"s of all pairs of bits $y$. Then $f(y) = 1$ iff $\vee(g(y)) = 1$ because $y$ has at least two 1s iff at least one pair of bits of $y$ are both 1s. We take the distribution over $\mathbb{Z}_3$-polynomials of degree 2 for $\vee$ on $\binom{m}{2}$ variables, and combine it with the reduction by replacing each variable with the product of the two relevant bits of $y$. This distribution computes $f$ with error probability $\leq 1/3$. Moreover, this distribution is over polynomials of degree 4, because replacing each of a monomial's variables with a product of two variables doubles the monomial's degree.

**Exercise 18.4.b:** Consider this distribution over $b \in \{0,1\}^m$: Pick $h \in \{1,2,\ldots,\lceil \log m \rceil\}$ uniformly at random, and then for each $i \in [m]$ independently, let $b_i = 1$ with probability $1/2^{h+1}$. If $\vee(y) = 0$ then $b \cdot y = 0$. If $\vee(y) = 1$, with $Y = \{i : y_i = 1\}$ where $2^{\ell-1} \leq |Y| \leq 2^\ell$ for some $\ell \in \{1,2,\ldots,\lceil \log m \rceil\}$, then:

$$\begin{aligned}
& \mathbf{Pr}[b \cdot y = 1] \\
\geq\ & \mathbf{Pr}\Big[b \cdot y = 1 \text{ and } h = \ell\Big] \\
=\ & \tfrac{1}{\lceil \log m \rceil} \mathbf{Pr}\Big[b \cdot y = 1 \,\Big|\, h = \ell\Big] && \text{(chain rule)} \\
=\ & \tfrac{1}{\lceil \log m \rceil} \textstyle\sum_{i \in Y} \mathbf{Pr}\Big[b_i = 1 \text{ and } b_j = 0 \text{ for all } j \in Y \setminus \{i\} \,\Big|\, h = \ell\Big] \\
=\ & \tfrac{1}{\lceil \log m \rceil} \textstyle\sum_{i \in Y} \mathbf{Pr}\Big[b_i = 1 \,\Big|\, h = \ell\Big] \cdot \Big(1 - \mathbf{Pr}\Big[b_j = 1 \text{ for some } j \in Y \setminus \{i\} \,\Big|\, h = \ell\Big]\Big) \\
\geq\ & \tfrac{1}{\lceil \log m \rceil} \textstyle\sum_{i \in Y} \mathbf{Pr}\Big[b_i = 1 \,\Big|\, h = \ell\Big] \cdot \Big(1 - \sum_{j \in Y \setminus \{i\}} \mathbf{Pr}\Big[b_j = 1 \,\Big|\, h = \ell\Big]\Big) && \text{(union bound)} \\
=\ & \tfrac{1}{\lceil \log m \rceil} \textstyle\sum_{i \in Y} \tfrac{1}{2^{\ell+1}} \cdot \Big(1 - \sum_{j \in Y \setminus \{i\}} \tfrac{1}{2^{\ell+1}}\Big) \\
=\ & \tfrac{1}{\lceil \log m \rceil} \tfrac{|Y|}{2^{\ell+1}} \Big(1 - \tfrac{|Y|-1}{2^{\ell+1}}\Big) \\
\geq\ & \tfrac{1}{\lceil \log m \rceil} \tfrac{1}{4} \Big(1 - \tfrac{1}{2}\Big) \\
=\ & \tfrac{1}{8\lceil \log m \rceil}
\end{aligned}$$

For $k = \big\lceil \ln(3) 8 \lceil \log m \rceil \big\rceil$, let $Q(y) = 1 - (1 - b^1 \cdot y) \cdots (1 - b^k \cdot y)$ for independent $b^1 \in \{0,1\}^m, \ldots, b^k \in \{0,1\}^m$, each distributed as above. If $\vee(y) = 0$ then $Q(y) = 0$. If $\vee(y) = 1$ then:

$$\mathbf{Pr}\Big[Q(y) \neq 1\Big] = \mathbf{Pr}\Big[b^1 \cdot y \neq 1 \text{ and } \cdots \text{ and } b^k \cdot y \neq 1\Big] \leq \Big(1 - \tfrac{1}{8\lceil \log m \rceil}\Big)^k \leq \mathrm{e}^{-k/8\lceil \log m \rceil} \leq 1/3$$

$Q$ has degree $\leq k = O(\log m)$, and $\mathbf{Pr}\Big[Q(y) \neq \vee(y)\Big] \leq 1/3$ for every $y \in \{0,1\}^m$.

**Exercise 18.5:** This is like the proof of Theorem 18.3 but using $\mathbb{Z}_5$ instead of $\mathbb{Z}_3$.

**Lemma.** *For every depth-$d$ size-$s$ circuit $C\colon \{0,1\}^N \to \{0,1\}$ (with "mod 5 gates" allowed) and every $k \geq 1$, some $\mathbb{Z}_5$-polynomial of degree $\leq (4k)^d$ agrees with $C$ on $\geq 1-s/5^k$ fraction of $\{0,1\}^N$.*

**Proof.** Like the proof of Lemma 18.4: It suffices to design a distribution over $N$-variate $\mathbb{Z}_5$-polynomials $P$ of degree $\leq (4k)^d$ such that $\mathbf{Pr}_P\big[P(x) \neq C(x)\big] \leq s/5^k$ for every $x \in \{0,1\}^N$. For a "mod 5 gate" with incoming wires $y = (y_1,\ldots,y_m)$, the degree-4 polynomial $Q(y) = 1-(y_1+y_2+\cdots+y_m)^4$ works since $1^4 = 2^4 = 3^4 = 4^4 = 1$. For an $\vee$ gate with incoming wires $y = (y_1,\ldots,y_m)$, the degree-$4k$ polynomial

$$Q(y) \;=\; \vee\big((b^1\cdot y)^4,\ldots,(b^k\cdot y)^4\big) \;=\; 1-\big(1-(b^1\cdot y)^4\big)\cdots\big(1-(b^k\cdot y)^4\big)$$

for independent and uniformly random $b^1 \in \mathbb{Z}_5^m,\ldots,b^k \in \mathbb{Z}_5^m$ works: If $\vee(y) = 0$ then $Q(y) = 0$. If $\vee(y) = 1$ then $\mathbf{Pr}\big[Q(y) \neq 1\big] = 1/5^k$ since $b\cdot y$ is uniformly distributed over $\mathbb{Z}_5$ (for uniformly random $b \in \mathbb{Z}_5^m$) and so $(b\cdot y)^4$ is 0 with probability $1/5$ and 1 with probability $4/5$. For the combined polynomial $P$ on any input $x \in \{0,1\}^N$, we have $\mathbf{Pr}[Q_i \text{ errs}] \leq 1/5^k$ for each gate, and we do a union bound over all $s$ gates. ■

**Lemma.** *Every $\mathbb{Z}_5$-polynomial of degree $\leq \sqrt{N}/2$ agrees with $\textsc{Parity}_N$ on $\leq 4/5$ fraction of $\{0,1\}^N$.*

**Proof.** Like the proof of Lemma 18.5: First, we change perspective: "Sum over $\mathbb{Z}_2$" (parity) becomes "product over $\mathbb{Z}_5$" if we change 0 to 1 and change 1 to $-1 = 4$: For all $x \in \{1,-1\}^N$,

$$x_1x_2\cdots x_N \;=\; 1-2\big((2x_1-2)\oplus(2x_2-2)\oplus\cdots\oplus(2x_N-2)\big)$$

since this is $-1$ iff $x$ has an odd number of $-1$s. Let $P$ be a $\mathbb{Z}_5$-polynomial of degree $\leq \sqrt{N}/2$ that agrees with $\textsc{Parity}_N$ on some $S \subseteq \{0,1\}^N$. Then $Q(x) = 1-2P(2x_1-2,\ldots,2x_N-2)$ is a $\mathbb{Z}_5$-polynomial of degree $\leq \sqrt{N}/2$ that agrees with the product function $x_1\cdots x_N$ on $T = \big\{x : (2x_1-2,\ldots,2x_N-2) \in S\big\} \subseteq \{1,-1\}^N$, which has the same size as $S$. We have $|T| \leq (4/5)2^N$ as in the proof of Lemma 18.5: Every function $f\colon T \to \mathbb{Z}_5$ is expressed by a polynomial of degree $\leq N/2+\sqrt{N}/4$, so:

$$5^{|T|} \;\leq\; 5^{\binom{N}{0}+\binom{N}{1}+\binom{N}{2}+\cdots+\binom{N}{\lfloor N/2+\sqrt{N}/4\rfloor}} \;\leq\; 5^{(4/5)2^N}$$

■

Combining the lemmas: Pick $k = \big\lfloor(\sqrt{N}/2)^{1/d}/4\big\rfloor = \Omega(N^{1/2d})$ so the two degree bounds essentially match: For every depth-$d$ size-$s$ circuit (with "mod 5 gates" allowed) for $\textsc{Parity}_N$, some $\mathbb{Z}_5$-polynomial of degree $\leq (4k)^d \leq \sqrt{N}/2$ agrees with the circuit on $\geq 1-s/5^k$ but $\leq 4/5$ fraction of $\{0,1\}^N$. Rearranging $1-s/5^k \leq 4/5$ yields $s \geq 5^k/5 = 2^{\Omega(N^{1/2d})}$.

**Exercise 18.6.a:** Suppose $\textsc{Majority}_{2N}$ has a depth-$d$ size-$s$ circuit $C$. We have $C(y) = 1$ iff $\text{weight}(y) > N$. We design a circuit for $\textsc{Parity}_N(x)$: For each $k \in [N]$ in parallel, evaluate $w_k \leftarrow C(x0^{k-1}1^{N-k+1})$ to determine whether $\text{weight}(x) \geq k$. Then, output $\bigvee_{\text{odd } k \in [N]}(w_k \wedge \overline{w_{k+1}})$ to determine whether $\text{weight}(x)$ is odd. This circuit has size $O(Ns)$ and depth $d+2$. By Theorem 18.7, $Ns \geq 2^{\Omega(N^{1/(d+2-1)})}$ and thus $s \geq 2^{\Omega((2N)^e)}$ for $e = 1/(d+1)$.

**Exercise 18.6.b:** This is like the proof of Claim 18.6. First, suppose $n$ is even.

$$\binom{n}{n/2}/2^n = \frac{n-1}{n}\cdot\frac{n-3}{n-2}\cdots\frac{5}{6}\cdot\frac{3}{4}\cdot\frac{1}{2} \geq \left(\frac{n-2}{n-1}\cdot\frac{n-4}{n-3}\cdots\frac{4}{5}\cdot\frac{2}{3}\right)\cdot\frac{1}{2}$$

$$\begin{aligned}\left(\binom{n}{n/2}/2^n\right)^2 &\geq \left(\frac{n-1}{n}\cdot\frac{n-3}{n-2}\cdots\frac{5}{6}\cdot\frac{3}{4}\cdot\frac{1}{2}\right)\cdot\left(\frac{n-2}{n-1}\cdot\frac{n-4}{n-3}\cdots\frac{4}{5}\cdot\frac{2}{3}\right)\cdot\frac{1}{2}\\ &= \frac{(n-1)\cdot(n-2)\cdot(n-3)\cdots4\cdot3\cdot2\cdot1}{n\cdot(n-1)\cdot(n-2)\cdot(n-3)\cdots4\cdot3\cdot2}\cdot\frac{1}{2} = \frac{1}{2n}\end{aligned}$$

Thus $\binom{n}{n/2} \geq 2^n/\sqrt{2n}$. If $n$ is odd, then $\binom{n}{\lceil n/2\rceil} = \frac{1}{2}\binom{n+1}{(n+1)/2} \geq \frac{1}{2}2^{n+1}/\sqrt{2(n+1)} = 2^n/\sqrt{2(n+1)}$.

**Exercise 18.6.c:** Let $f = \textsc{Majority}_N$, so $f(x) = 1$ iff $\text{weight}(x) > N/2$. Recall that $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ and $\binom{n}{k} = \binom{n}{n-k}$.

First, suppose $N$ is odd. The number of inputs $x$ such that $f(x) = x_1 = 1$ is:

$$\sum_{k=(N-1)/2}^{N-1} \binom{N-1}{k} = \left(2^{N-1} + \binom{N-1}{(N-1)/2}\right)/2$$

This is also the number of inputs $x$ such that $f(x) = x_1 = 0$. Thus the number of inputs $x$ such that $f(x) = x_1$ is

$$2^{N-1} + \binom{N-1}{(N-1)/2} \geq 2^{N-1} + 2^{N-1}/\sqrt{2(N-1)} \geq 2^N\left(1/2 + \Omega(1/\sqrt{N})\right)$$

by Exercise 18.6.b.

Now, suppose $N$ is even. The number of inputs $x$ such that $f(x) = x_1 = 1$ is:

$$\sum_{k=N/2}^{N-1} \binom{N-1}{k} = 2^{N-1}/2$$

The number of inputs $x$ such that $f(x) = x_1 = 0$ is:

$$\sum_{k=N/2-1}^{N-1} \binom{N-1}{k} = 2^{N-1}/2 + \binom{N-1}{N/2-1} = 2^{N-1}/2 + \binom{N-1}{\lceil (N-1)/2 \rceil}$$

Thus the number of inputs $x$ such that $f(x) = x_1$ is

$$2^{N-1} + \binom{N-1}{\lceil (N-1)/2 \rceil} \geq 2^{N-1} + 2^{N-1}/\sqrt{2N} \geq 2^N\left(1/2 + \Omega(1/\sqrt{N})\right)$$

by Exercise 18.6.b.

**Exercise 18.7.a:** We prove that if $f\colon \{0,1\}^N \to \{0,1\}$ has a $k$-DNF $\varphi$, then for a uniformly random $M$-partial assignment $a$, where $M \leq N/2$:

$$\mathbf{Pr}\big[f|_a \text{ has no depth-}\ell \text{ decision tree}\big] \leq (16kM/N)^\ell$$

Let $R$ be the set of all $M$-partial assignments $a$ such that $f|_a$ has no depth-$\ell$ decision tree. As in the proof of Lemma 18.10, it suffices to exhibit an injection from $R$ to $S$.

Consider any $a \in R$. Since the decision tree that simulates $\varphi|_a$ has depth $> \ell$, it has a rejecting leaf at depth $> \ell$. Let $b$ be the 0-certificate of $f|_a$ that records the results of queries along the path to this leaf. This $b$ might not be minimal, but the encoder and decoder from the proof of Lemma 18.10 still work in the same way. The minimality of $b$ was only used to justify the existence of a surviving term in $\varphi|_{ab^1\cdots b^{i-1}}$, but now that follows from the decision tree's definition. Also, the tweak for ensuring $d$ has exactly $M-\ell$ many $*$s and $e$ has exactly $\ell$ entries still works.

**Exercise 18.7.b:** As in the proof of Lemma 18.11, let $k = \ell = 2\log s$, and let $M_1 = N/32$ and $a^1$ be a uniformly random $M_1$-partial assignment to the $N$ variables, and for $i \leftarrow 2, \ldots, d-1$, let $M_i = M_{i-1}/32k$ and $a^i$ be a uniformly random $M_i$-partial assignment to the $M_{i-1}$ many $*$ variables of $a^1 \cdots a^{i-1}$. Now, let $M_d = M_{d-1}/32k$ and $a^d$ be a uniformly random $M_d$-partial assignment to the $M_{d-1}$ many $*$ variables of $a^1 \cdots a^{d-1}$. For the output gate $g$, let $B_g$ be the *bad* event that $g$ isn't a depth-$k$ decision tree under $a^1 \cdots a^d$ but is a $k$-DNF or $k$-CNF (depending on whether $d$ is even or odd) under $a^1 \cdots a^{d-1}$. By Exercise 18.7.a, $\mathbf{Pr}[B_g] \leq 1/s^2$ for the output gate $g$ as well. By a union bound over all $s$ gates, $\mathbf{Pr}[B_g \text{ holds for at least one } g] \leq \sum_g \mathbf{Pr}[B_g] \leq s/s^2 = 1/s$. If no bad event holds, then the circuit becomes a depth-$k$ decision tree under $a = a^1 \cdots a^d$, which is a uniformly random $N/32(32k)^{d-1}$-partial assignment.

**Exercise 18.7.c:** Let $s = 2^{N^{1/d}/64} - 1$ and $k = 2\log s$. Consider any depth-$d$ size-$s$ circuit exactly computing some $f : \{0,1\}^N \to \{0,1\}$. Let $a$ be a uniformly random $N/32(32k)^{d-1}$-partial assignment, and $b$ be a uniformly random assignment to $a$'s $*$ variables, so $ab \in \{0,1\}^N$ is a uniformly random assignment. Since $s < 2^{N^{1/d}/64}$, we have $k < N/32(32k)^{d-1}$. Thus if $f|_a$ has a depth-$k$ decision tree, then $\mathbf{Pr}_b\big[f|_a(b) = \textsc{Parity}_N|_a(b)\big] = 1/2$ since each root-to-leaf path misses at least one variable of $\textsc{Parity}_N|_a$. By Exercise 18.7.b:

$$\begin{aligned}\mathbf{Pr}\big[f(ab) = \textsc{Parity}_N(ab)\big] \;&\leq\; \mathbf{Pr}\big[f(ab) = \textsc{Parity}_N(ab) \,\big|\, f|_a \text{ has a depth-}k\text{ decision tree}\big] + \\ &\qquad \mathbf{Pr}\big[f|_a \text{ has no depth-}k\text{ decision tree}\big] \\ &\leq\; 1/2 + 1/s\end{aligned}$$

**Exercise 18.8:** This circuit does the job:

sample $y \in \{0,1\}^n$ uniformly at random
output $(y_1,\ y_1 \oplus y_2,\ y_2 \oplus y_3,\ \ldots,\ y_{n-2} \oplus y_{n-1},\ y_{n-1} \oplus y_n,\ y_n)$

This circuit is $\mathsf{NC}^0$-type since each output bit depends on at most two of the random input bits. The output distribution is correct since for each $x \in \{0,1\}^n$, the output $(x, \oplus(x))$ is yielded by exactly one $y \in \{0,1\}^n$, namely $y_i = x_1 \oplus x_2 \oplus \cdots \oplus x_i$ for each $i$.

**Exercise 18.9:** We may assume $N$ is odd. Let $X = \left\{x \in \{0,1\}^N : \text{weight}(x) = \lceil N/2 \rceil\right\}$ and $Y = \left\{y \in \{0,1\}^N : \text{weight}(y) = \lfloor N/2 \rfloor\right\}$. Let $f$ be the search problem $\text{MAJORITY}^*_N$ restricted to domain $X \times Y$. We claim that $f$ has protocol size complexity $\geq N^2/4$, which implies the same lower bound for $\text{MAJORITY}^*_N$ and thus that $\text{MAJORITY}_N$ has formula size complexity $\geq N^2/4$ (Lemma 18.16).

Consider any size-$s$ protocol for $f$. Let $Z \subseteq X \times Y$ be the set of all inputs $(x, y)$ with $\text{dist}(x, y) = 1$, that is, $(x, y)$ has only one acceptable solution. We have $|X| = |Y|$ and $|Z| = \lceil N/2 \rceil \cdot |X|$. The proof of Theorem 17.33 shows that $s \geq |Z|^2/|X \times Y| = \lceil N/2 \rceil^2 \geq N^2/4$.

**Exercise 18.10:** Let $f = \text{DIRECTED REACHABILITY}_{n^2}$. As noted in the proof of Theorem 18.19, $f$ has a depth-$O(\log^2 n)$ bounded-fan-in monotone formula. By Lemma 18.18, $f'$ has a depth-$O(\log^2 n)$ protocol. We describe this protocol explicitly. Alice gets a graph $x$ with a walk (of length $\leq n-1$) from $s$ to $t$. Bob gets a graph $y$ with no walk from $s$ to $t$. They want to find an edge that $x$ has but $y$ doesn't. First, Alice sends the identity of the halfway node $v$ on her walk from $s$ to $t$, with $\log n$ bits of communication. Since $y$ has no walk from $s$ to $t$, either $y$ has no walk from $s$ to $v$, in which case Bob says so and they update $t \leftarrow v$, or $y$ has no walk from $v$ to $t$, in which case Bob says so and they update $s \leftarrow v$. This is one bit of communication from Bob. They repeat this, maintaining the invariant that $x$ has a walk from the current $s$ to the current $t$, and $y$ doesn't. The length of the walk from the current $s$ to the current $t$ in $x$ decreases by at least a factor 2 in each iteration. After at most $\log n$ iterations, $x$ has a length-1 walk from $s$ to $t$, and $y$ doesn't, so Alice and Bob output the edge $(s,t)$. The protocol's depth is $\leq (\log n \text{ iterations}) \cdot (\log n + 1 \text{ bits per iteration}) = O(\log^2 n)$.

**Exercise 18.11:** This is like the proof for Directed Reachability (Theorem 18.19).

Define Undirected Reachability$_{n(n-1)/2}$: $\{0,1\}^{n(n-1)/2} \to \{0,1\}$ as a monotone function: The input is an $n$-node undirected graph $z$ where for all $1 \le i < j \le n$, we have $z_{i,j} = 1$ iff there's an edge between nodes $i$ and $j$. The output is 1 iff the graph has a walk between node $s = 1$ and node $t = n$.

By balancing (Theorem 6.24), it suffices to prove $f =$ Undirected Reachability$_{n(n-1)/2}$ has monotone formula depth complexity $\Omega(\log^2 n)$. By Lemma 18.18, it suffices to prove $f'$ has protocol depth complexity $\Omega(\log^2 n)$.

Undirected Fork$_{k,m}$ is the search problem where Alice gets $x \in [k]^m$, Bob gets $y \in [k]^m$, and they want to output an *undirected fork*, which is an index $i \in \{0,1,\dots,m\}$ such that either $x_i = y_i$ and $x_{i+1} \neq y_{i+1}$ or $x_i \neq y_i$ and $x_{i+1} = y_{i+1}$, assuming implicitly that $x_0 = y_0$ and $x_{m+1} \neq y_{m+1}$, say $x_0 = y_0 = 1$ and $x_{m+1} = k$ and $y_{m+1} = k-1$.

We reduce Undirected Fork$_{k,m}$ to $f'$ where $n = k(m+2)$ and $k = m \approx \sqrt{n}$. Then by Exercise 17.22, both problems have protocol depth complexity $\Omega\big((\log k)(\log m)\big) = \Omega(\log^2 n)$.

The inputs to $f'$ will be layered graphs with nodes $[k] \times \{0,1,\dots,m+1\}$: Layer $i$ is the nodes $[k] \times \{i\}$, and each edge goes between two adjacent layers. The start and destination nodes are $s = (1,0)$ and $t = (k, m+1)$.

- Map $x \in [k]^m$ to a graph $x'$ having just the path

$$(x_0,0) - (x_1,1) - (x_2,2) - \cdots - (x_m,m) - (x_{m+1},m+1)$$

  between $(1,0) = s$ and $(k,m+1) = t$. Thus $f(x') = 1$ ($t$ is reachable from $s$).

- Map $y \in [k]^m$ to a graph $y'$ having the path

$$(y_0,0) - (y_1,1) - (y_2,2) - \cdots - (y_m,m) - (y_{m+1},m+1)$$

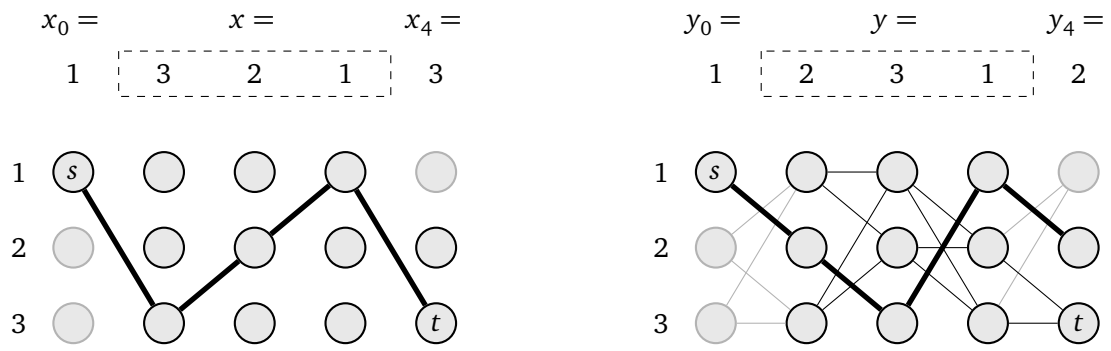
  between $(1,0) = s$ and $(k-1,m+1) \neq t$, and $y'$ also has an edge between each pair of nodes that are both not on this path and are in adjacent layers. The latter edges don't create any new paths from $s$, so $f(y') = 0$ ($t$ isn't reachable from $s$).

Another way to describe the layered graphs $x'$ and $y'$ is:

- $(v,i) - (w,i+1)$ is an edge of $x'$ iff $x_i = v$ and $x_{i+1} = w$.
- $(v,i) - (w,i+1)$ is a non-edge of $y'$ iff either $y_i = v$ and $y_{i+1} \neq w$ or $y_i \neq v$ and $y_{i+1} = w$.

A protocol for Undirected Fork$_{k,m}$ on input $(x,y)$ could run a protocol for $f'$ on input $(x',y')$ to find an acceptable solution $(v,i) - (w,i+1)$ (an edge of $x'$ that's a non-edge of $y'$) and then output $i$, which is an undirected fork for $(x,y)$ since either $x_i = v = y_i$ and $x_{i+1} = w \neq y_{i+1}$ or $x_i = v \neq y_i$ and $x_{i+1} = w = y_{i+1}$. Hence the protocol depth complexity of Undirected Fork$_{k,m}$ is no greater than that of $f'$.

For example, with $k = m = 3$ and $x = (3,2,1)$ and $y = (2,3,1)$, with implicit "bookends" $x_0 = y_0 = 1$ and $x_4 = 3$ and $y_4 = 2$:




The undirected forks are the indices 0, 2, and 3.

**Exercise 18.12:** This is like Lemma 18.16.

$\Rightarrow$: Suppose $f$ has a shape-$G$ circuit. We design a shape-$G^*$ rectangle dag for $f^*$. For each node $v$, let $A_v \times B_v \subseteq X \times Y$ be the rectangle of all $(x, y)$ such that in the circuit, $v$ evaluates to 1 on $x$ and to 0 on $y$.

- Each sink $v$'s rectangle $A_v \times B_v$ is monochromatic with respect to $f^*$ since the corresponding source in the circuit is a literal $z_i$ or $\overline{z_i}$, which means either $x_i = 1$ for all $x \in A_v$ and $y_i = 0$ for all $y \in B_v$, or $x_i = 0$ for all $x \in A_v$ and $y_i = 1$ for all $y \in B_v$, and thus $i$ is an acceptable solution (that is, $x_i \neq y_i$) for all $(x, y) \in A_v \times B_v$.
- For each non-sink $v$ with outneighbors $v_0$ and $v_1$ in the rectangle dag:
  - If $v$ is an $\vee$ gate, then $A_v \subseteq A_{v_0} \cup A_{v_1}$ and $B_v \subseteq B_{v_0} \cap B_{v_1}$ by definition.
  - If $v$ is an $\wedge$ gate, then $A_v \subseteq A_{v_0} \cap A_{v_1}$ and $B_v \subseteq B_{v_0} \cup B_{v_1}$ by definition.
- For the unique source $v$, we have $A_v \times B_v = X \times Y$ because the circuit computes $f$.

$\Leftarrow$: Suppose $f^*$ has a shape-$G^*$ rectangle dag. We design a shape-$G$ circuit for $f$ such that each node $v$ evaluates to 1 on all $x \in A_v$ and to 0 on all $y \in B_v$.

- For each source $v$ in the circuit, where $A_v \times B_v$ is monochromatic with respect to $f^*$ and $i$ is an acceptable solution (that is, $x_i \neq y_i$) for all $(x, y) \in A_v \times B_v$, either $x_i = 1$ for all $x \in A_v$ and $y_i = 0$ for all $y \in B_v$, in which case $v$ becomes the positive literal $z_i$, or $x_i = 0$ for all $x \in A_v$ and $y_i = 1$ for all $y \in B_v$, in which case $v$ becomes the negative literal $\overline{z_i}$.
- For each non-source $v$ with inneighbors $v_0$ and $v_1$ in the circuit:
  - If $A_v \subseteq A_{v_0} \cup A_{v_1}$ and $B_v \subseteq B_{v_0} \cap B_{v_1}$, then $v$ becomes an $\vee$ gate.
  - If $A_v \subseteq A_{v_0} \cap A_{v_1}$ and $B_v \subseteq B_{v_0} \cup B_{v_1}$, then $v$ becomes an $\wedge$ gate.
- The unique sink $v$ outputs 1 on all $x \in A_v = X$ and 0 on all $y \in B_v = Y$, so the circuit computes $f$.

**Exercise 18.13:** The example in §18.7.1 (based on Lemma 18.11) shows for $s = 2^{N^{1/(d-1)}/128}$, there exists a $(2^{3N}, 1/2)$-natural property for depth-$d$ size-$s$ circuits. By Theorem 18.22 (which preserves depth), if there exists a $(t - 2^N, \varepsilon)$-natural property for depth-$d$ size-$s$ circuits, then there doesn't exist a $(t, \varepsilon)$-pseudorandom function computed by a depth-$d$ size-$s$ circuit with $n \geq N$. Combining these shows that there doesn't exist a $(2^{3N} + 2^N, 1/2)$-pseudorandom function computed by a depth-$d$ size-$s$ circuit with $n \geq N$. Letting $N = n^c$, there doesn't exist a $(2^{3n^c} + 2^{n^c}, 1/2)$-pseudorandom function computed by a depth-$d$ size-$2^{n^{c/(d-1)}/128}$ circuit.