Solving $\overset{2\text{x}2}{\vee}$ Go

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

1. Logistics & Recap
2. Nim, part 2½
3. 2x2 Go

Logistics

- **Practice questions #3** are available
  - ⚠️ **Corrected solutions released this morning** ⚠️
- **Quiz 3** is **this Friday** (Feb 27)
  - *Coverage:* up to and including **Feb 13** (Move Ordering & Nim)
  - Bring your student ID!
  - No calculators or other devices
- **TA Office hours:** every Thursday 1pm-2pm in **UCOMM-3-136**

Recap: XorSum

- **xorsum** operation: Cumulative bitwise XOR of **each binary digit** of the pile sizes
- Examples:
  - xorsum(1,2,3) = 00
    - $\begin{array}{l} 1 = 01 \\ 2 = 10 \\ 3 = 11 \end{array}$ and $\begin{array}{l} 0 \oplus 1 \oplus 1 = 0 \\ 1 \oplus 0 \oplus 1 = 0 \end{array}$
  - xorsum(3,5,3) = 101b = 5
    - $\begin{array}{l} 3 = 011 \\ 5 = 101 \\ 3 = 011 \end{array}$ and $\begin{array}{l} 0 \oplus 1 \oplus 0 = 1 \\ 1 \oplus 0 \oplus 1 = 0 \\ 1 \oplus 1 \oplus 1 = 1 \end{array}$

Recap: Nim Formula Theorem

Theorem: A Nim position with pile sizes $p_1, \ldots, p_k$ is **losing** iff

$$\text{xorsum}(p_1, \ldots, p_k) = 0.$$

Proof sketch:

1. If $\text{xorsum}(\text{P}) = 0$, then $\text{xorsum}(C) \neq 0$ for **every** child C of P
2. If $\text{xorsum}(P) \neq 0$, then $\text{xorsum}(C) = 0$ for **some** child C of P
3. $\text{xorsum}(0,0,\ldots,0) = 0$
4. Any position that has $(0,0,\ldots,0)$ as a child is a **winning** position
5. Complete the proof by induction

Nim Formula Theorem Proof Step 2

2. If $\mathrm{xorsum}(P) \neq 0$, then $\mathrm{xorsum}(C) = 0$ for **some** child C of P

(a) Let d be the digit number of the most significant 1 in the binary representation of $\mathrm{xorsum}(P)$

(b) There exists at least one j such that p_j has a 1 in digit d (**why?**)

(c) $\mathrm{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k)$ has a 0 in the d-th column (**why?**)

- and *also* in every more-significant column that p_j has a 0 in (**why?**)

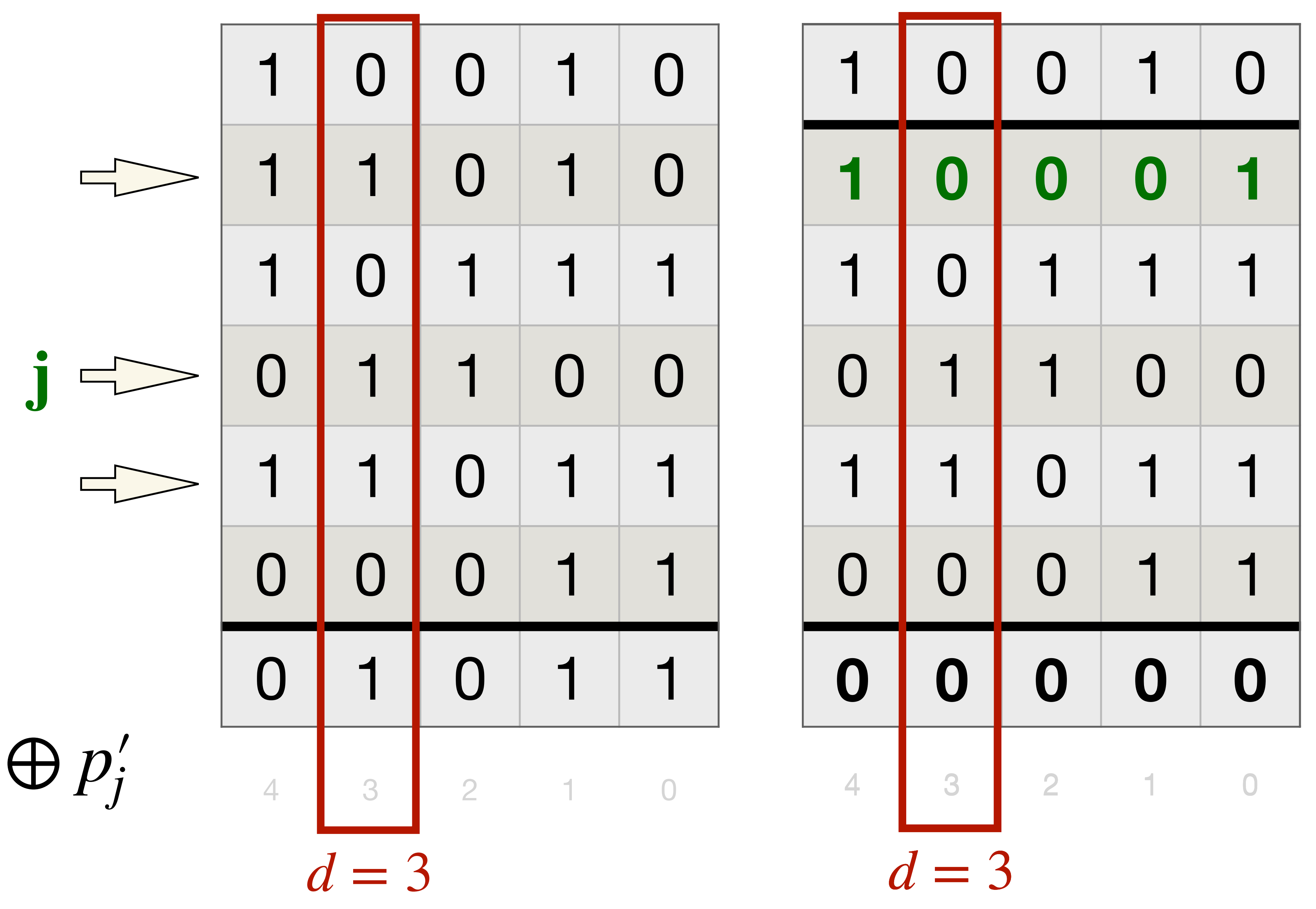
(d) So $\mathrm{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k) < p_j$ (**why?**)

(e) Set $p'_j = \mathrm{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k)$

(f)

$$\begin{aligned} \mathrm{xorsum}(p_1, \ldots, p_{j-1}, \mathbf{p'_j}, p_{j+1}, \ldots, p_k) &= \mathrm{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k) \oplus p'_j \\ &= p'_j \oplus p'_j \\ &= 0 \end{aligned}$$

■



1	0	0	1	0
1	1	0	1	0
1	0	1	1	1
0	1	1	0	0
1	1	0	1	1
0	0	0	1	1
0	1	0	1	1
4	3	2	1	0

1	0	0	1	0
1	0	0	0	1
1	0	1	1	1
0	1	1	0	0
1	1	0	1	1
0	0	0	1	1
0	**0**	**0**	**0**	**0**
4	3	2	1	0

Nim Formula Theorem Proof Step 5

1. If $\mathrm{xorsum(P)} = 0$, then $\mathrm{xorsum}(C) \neq 0$ for **every** child C of P
2. If $\mathrm{xorsum}(P) \neq 0$, then $\mathrm{xorsum}(C) = 0$ for **some** child C of P
3. $\mathrm{xorsum}(0,0,\ldots,0) = 0$
4. Any position that has $(0,0,\ldots,0)$ as a child is a **winning** position

Inductive hypothesis: suppose that for all positions P within k steps of $(0,\ldots,0)$,
$(\mathrm{xorsum}(P) \neq 0 \implies P$ is winning) and $(\mathrm{xorsum}(P) = 0 \implies P$ is losing)

Then for all P that are within $k+1$ steps of $(0,\ldots,0)$:

(a) $\mathrm{xorsum}(P) \neq 0$:

- by (2), $\mathrm{xorsum}(C) = 0$ for **some** child C
- C is within k of $(0,\ldots,0)$, so by IH C is **losing**
- So P has **at least one losing child**, and is therefore **winning**

(b) $\mathrm{xorsum}(P) = 0$:

- by (1), $\mathrm{xorsum}(C) \neq 0$ for **all** children C
- every C is within k of $(0,\ldots,0)$, so by IH every C is **winning**
- So **all of P's children are winning**, hence P is **losing**

Base case:

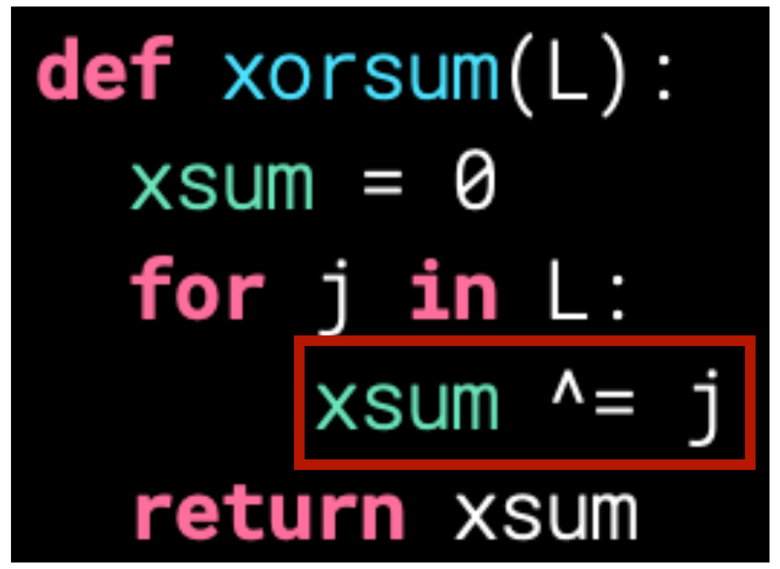
(a) By (3), every position P within 0 of $(0,\ldots,0)$ has $\mathrm{xorsum}(C) = 0$, and by definition it is a losing position

(b) By (1), every position P within 1 of $(0,\ldots,0)$ has $\mathrm{xorsum}(C) \neq 0$, and by (4) they are all winning ■

Recap: Using the Nim Formula

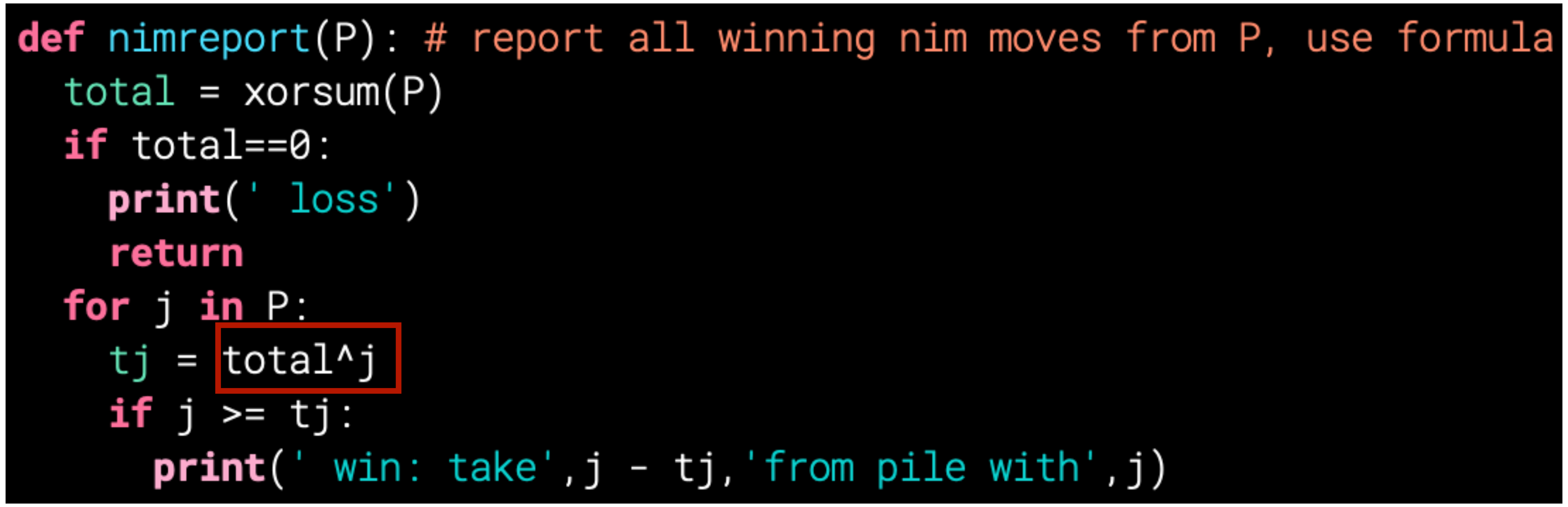
- **Checking** if a position P is **winning** or **losing**: Simply compare $\text{xorsum}(P)$ to 0
- Finding a **winning move**:
  1. Compute $\text{xorsum}(P)$
  2. Find d and a pile j with a 1 in digit d
     - Equivalently: a pile j with $p_j \geq \text{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k)$
  3. Set $p'_j = \text{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k)$
  4. Take $p_j - p'_j$ stones from pile j
     - i.e., move to position $(p_1, \ldots, p_{j-1}, \mathbf{p'_j}, p_{j+1}, \ldots, p_k)$

Nim Formula Implementation: **nim/nim.py**



```python
def xorsum(L):
  xsum = 0
  for j in L:
      xsum ^= j
  return xsum
```

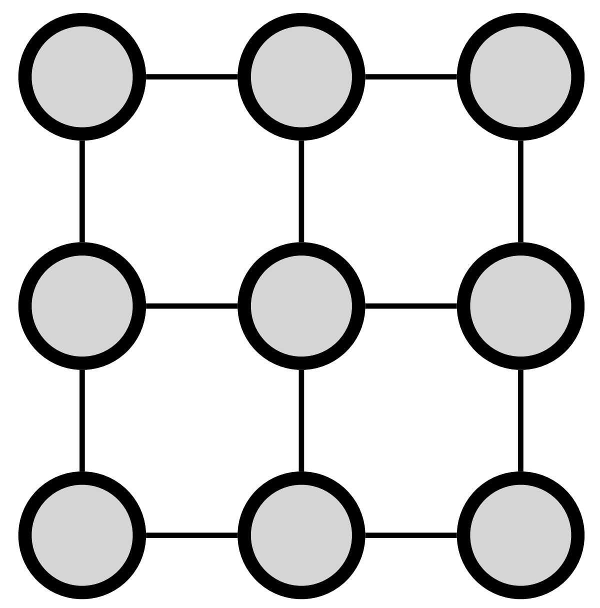
(x ^ y) is the Python syntax for $(x \oplus y)$



```python
def nimreport(P): # report all winning nim moves from P, use formula
  total = xorsum(P)
  if total==0:
    print(' loss')
    return
  for j in P:
    tj = total^j
    if j >= tj:
      print(' win: take',j - tj,'from pile with',j)
```

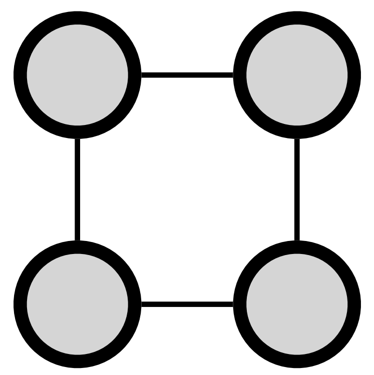
Note: $\text{xorsum}(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k) = \text{xorsum}(p_1, \ldots, p_k) \oplus p_j$ **(why?)**

Recap: Go!



1. Played on a grid of **points** between players Black and White
2. Each point is either Black, White, or Empty
3. A **move** consists of either Pass, or colouring one point (placing a stone)
4. An Empty point adjacent to a point is a **liberty** of that point
5. Connected **groups** of the same colour share liberties
6. After placing a stone, any group of the other colour with no liberties is **captured**
7. **Suicide:** After all captures, the placed stone must belong to a group with at least 1 liberty
8. **Positional superko:** After a move, the position must differ from all previous positions

2×2 Go: Search Graph



- **Question:** How many **completed games** of tic-tac-toe are there? (ignoring isomorphisms)
  - Brute-force minimax search on tic-tac-toe is workable even without optimizations
- What about for 2×2 Go?
  - 4 points, each can take one of 3 values
  - So at most $3^4 = 81$ **positions**
  - But a position is not enough to determine the state (**why?**)
  - A completed game is a legal sequence of states
- It turns out there are **386,356,909,593** legal games of 2×2 Go
  - For 3×3 Go, the number is approximately 10^{1100}

Minimax Value of 2×2 Go

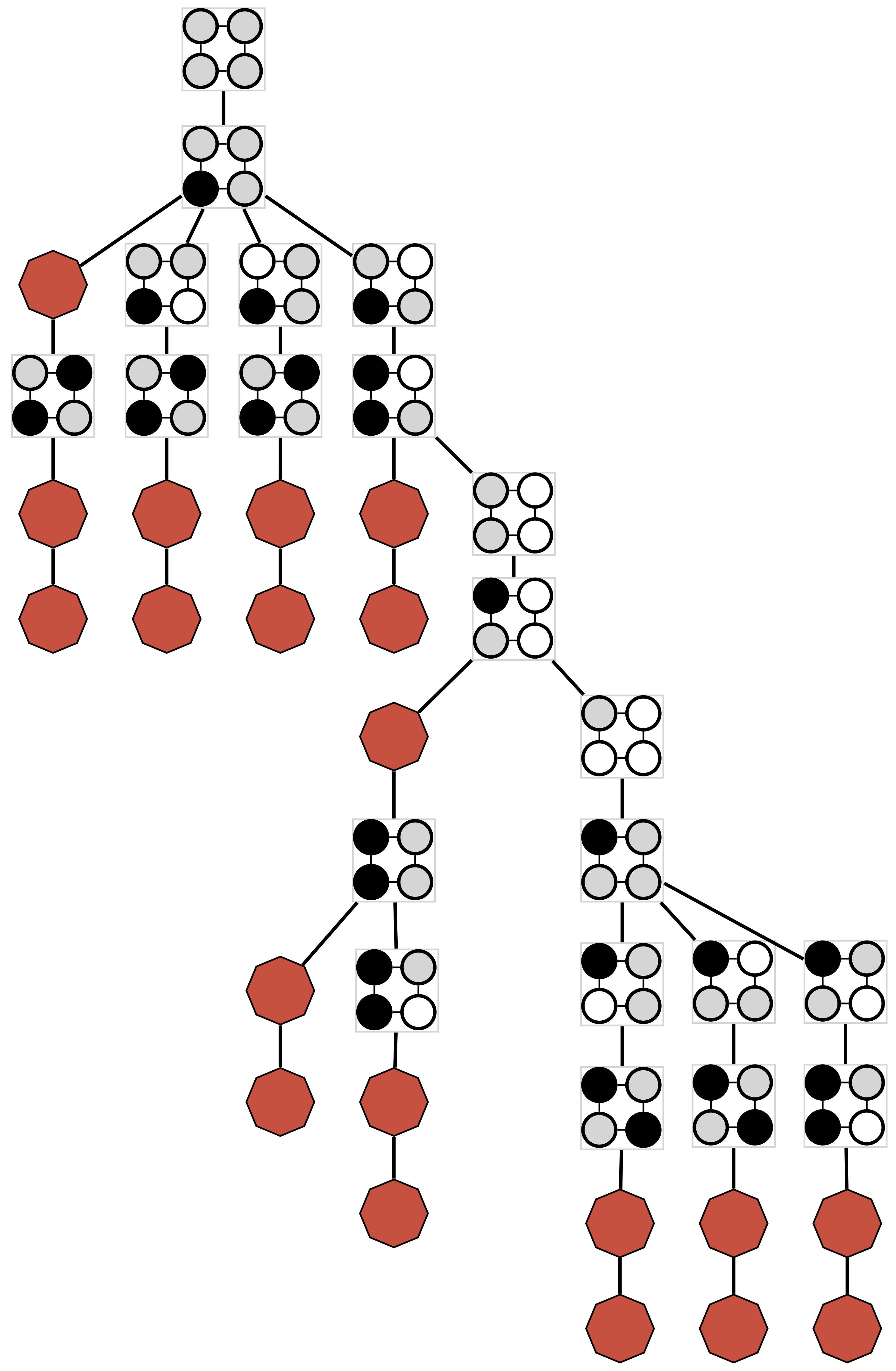
- John Tromp implemented three programs to compute the minimax value of 2x2 Go
- Used **positional superko** rule
  - **Question:** Would the **situational superko** rule be easier or harder to solve?
- The three programs used **only techniques from CMPUT 355:**
  1. Unoptimized minimax
  2. Alpha-beta pruning (example: **go/tromp2.py**)
  3. Alpha-beta pruning with move ordering (example: **go/tromp.py**)
- **Question:** Which move ordering will yield the most pruning?

Move Ordering for 2×2 Go

- Move ordering makes a really big difference:
  1. **Minimax:** Searches at least 10^{12} **nodes** (ran for weeks but then killed)
  2. **Alpha-beta** pruning with **stone moves first**, pass moves second:
     Searches $19{,}397{,}529$ **nodes** (with a maximum depth of **58**)
  3. **Alpha-beta** pruning with **pass moves first**:
     Searches $1{,}446$ **nodes** (with a maximum depth of **22**)
- **Pass-first** ordering yields dramatic improvements in pruning:
  - Gets to the closest possible terminal
  - Establishes alpha/beta values very quickly

Minimax Proof for 2×2 Go

- It turns out that the minimax value of 2×2 Go is **1**
- **Question: How many proof trees** do we need to prove that? (**why?**)
- **Question:** What does the proof tree at the right prove?



Source: https://webdocs.cs.ualberta.ca/~hayward/355/ssgo.pdf

Summary

- **Nim formula:**
    - Tells us whether a position is **winning** or **losing** **without search**
    - Allows us to compute a **winning move** from a winning position
        - Without search
        - Without even looking at **every move!**
- **2x2 Go:**
    - Can be solved using **just alpha-beta search**
        - Without alpha-beta: hopeless
    - **Pass-first** move ordering dramatically reduces the search space
- 3x3 Go is **significantly harder**
    - So 2x2 Go is right at the edge of feasibility for alpha-beta-based techniques