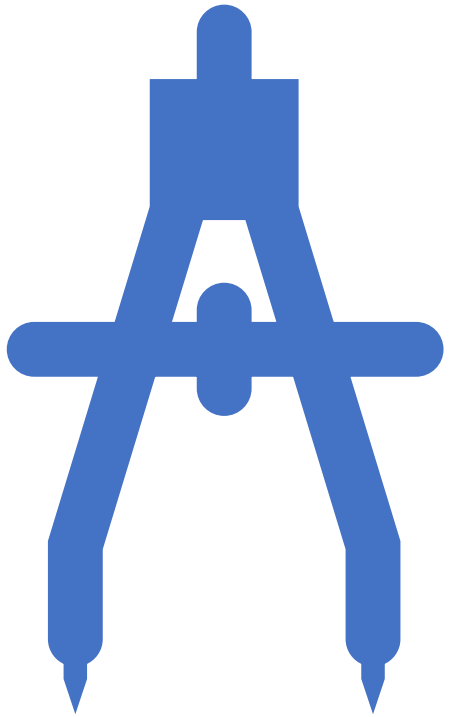
PSPACE



Space Complexity

How much space is required to solve 3-SAT?

- Linear, since you need to remember the current truth assignment.

How much space is required to verify a 3-SAT solution?

- Seems like it would be about the same.

PSPACE is the class of problems that can be solved with a polynomial amount of space on a deterministic TM.

NPSPACE is the class of problems that can be verified with a polynomial amount of space.

- The obvious question is, does PSPACE = NPSPACE?

Savitch's Theorem

Savitch's Theorem states that if a problem can be verified with f(n) space, then it can be solved with $f^2(n)$ space.

- This is (and was) a shocking result that asserts PSPACE = NPSPACE!

What we know

$P \subseteq NP$

$NP \subseteq PSPACE$

- Because $NP \subseteq NPSPACE$

$PSPACE = NPSPACE$

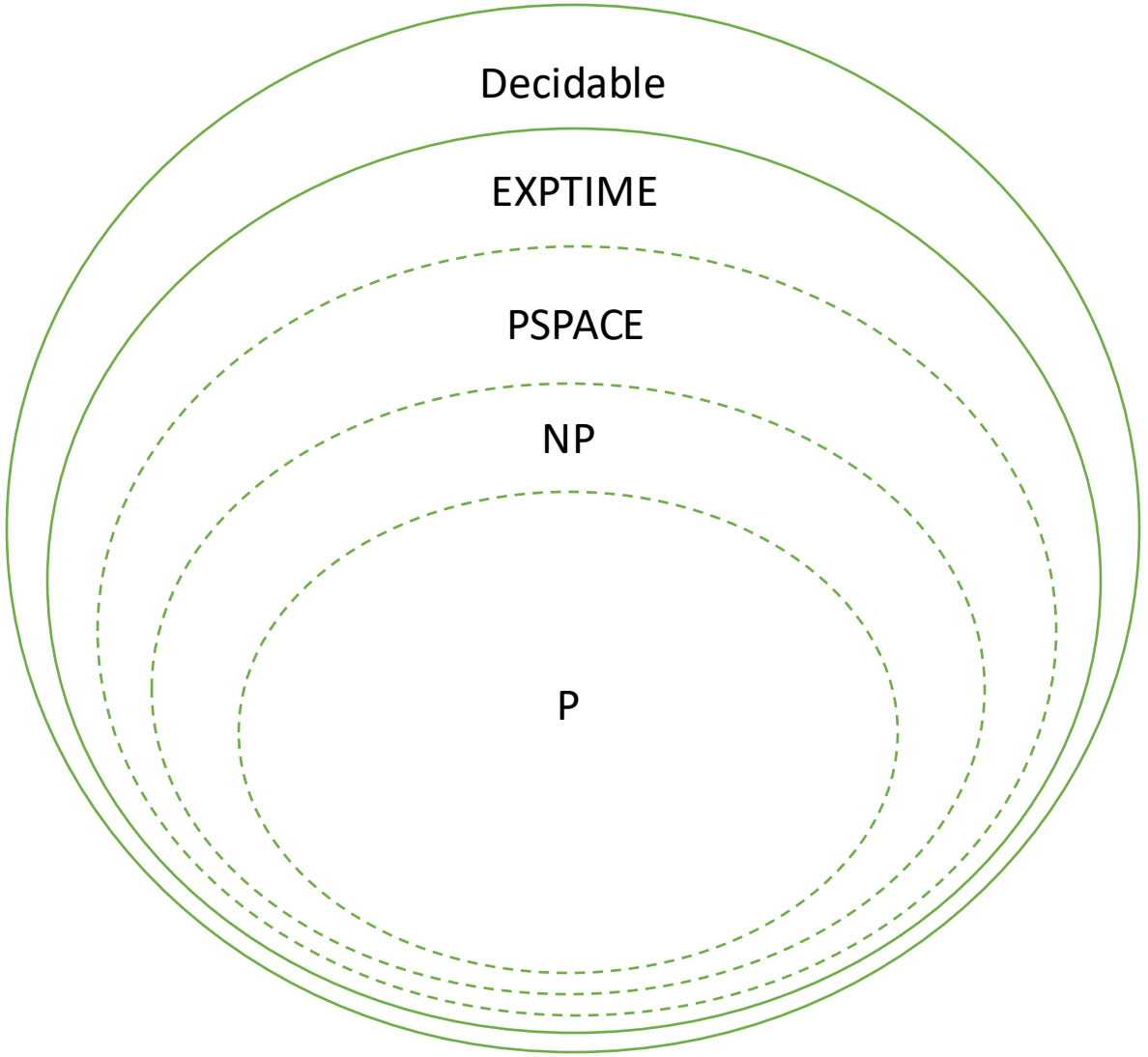
$PSPACE \subseteq EXPTIME$

- There are an exponential number of configurations when restrained by polynomial space.

$P \neq EXPTIME$

- Most researchers think $P \subsetneq NP \subsetneq PSPACE \subsetneq EXPTIME$

Complexity Classes



PSPACE-Complete

A language B is **PSPACE-Complete** if it satisfies both conditions:

- $B \in PSPACE$
- $\forall x \in PSPACE, x \leq_P B$

We still use poly-time reductions. One could imagine constructing a more powerful type of reduction and using that instead.

- This is unnecessarily complicated. Poly-time reductions are intuitive.
- If P=NP (which has not been disproven), there wouldn't even be a difference between these two types of reductions.
- Such a reduction would not help us prove that $P \neq PSPACE$, which is an easier result than showing $NP \neq PSPACE$

A PSPACE-Complete Problem

A **fully-quantified Boolean formula** is a Boolean formula (such as
$(x \vee y) \wedge (\neg x \vee \neg y)$) preceded by quantifiers for all variables.

- E.g., $\forall x \exists y((x \vee y) \wedge (\neg x \vee \neg y))$. Is this formula true?
- Yes: when x = true, y = false. When x = false, y = true.

The **True Quantified Boolean Formula** problem (TQBF) is defined:

- TQBF = $\{\langle \phi \rangle \mid \phi$ is a true fully-quantified Boolean formula$\}$

TQBF is our first PSPACE-Complete problem. We will not prove it, but the proof looks pretty similar to the Cook-Levin Theorem (take CSCI 475 for details).

TQBF $\in$ PSPACE

Do a recursive DFS through all possible variable assignments.

- If your current quantifier is $\forall$, return true if both T and F work, otherwise return false (to the previous level of recursion).
- If your current quantifier is $\exists$, return false if either T or F work, otherwise return true (to the previous level of recursion).
- This requires a linear amount of stack space.

$\forall x \exists y((x \vee y) \wedge (\neg x \vee \neg y))$

- For x = true, recursively call your function. If your recursive call returns false, you return false. Otherwise, for x = false, recursively call your function, and you return whatever the recursive call returns.
- In the recursive call, for y = true, evaluate the formula. If it evaluates to true, you return true. Otherwise, for y = false, you return whatever the formula evaluates to.

The Formula Game

You are given a fully-quantified Boolean formula ϕ.

- There are two players: Player A and Player E.
- Play order proceeds via the order of the quantifiers in ϕ.
- Every time you see $\forall x$, player A gets to choose the value of x.
- Every time you see $\exists y$, player E gets to choose the value of y.
- After all moves have been made, remove the quantifiers, and apply the chosen values to each variable.
- A wins if the formula evaluates to false.
- E wins if the formula evaluates to true.

An Example Game

$\forall x \exists y \exists z [(x \vee y) \wedge (y \vee z) \wedge (\neg y \vee \neg z)]$

- Player A sets x = False
- Player E sets y = True
- Player E sets z = False
- The formula evaluates to True, so player E wins.
- If Player A had set x = True, the same moves by Player E would lead to Player E winning.
- Player E has a winning strategy on this formula.

An Example Game

$\exists x \, \forall y \, \exists z \, [(x \vee y) \wedge (y \vee z) \wedge (y \vee \neg z)]$

Who has the winning strategy?

- Player A does.
- When y = True, player E can't make both the 2nd and 3rd clause true.

What kind of formula lead to player A having a winning strategy?

- Any formula that is false. Since there formula is false, there must be choices of the $\forall$ variables that evaluate to false, regardless of the choices of the $\exists$ variables.
- Similarly, player E has a winning strategy whenever the formula is true. No matter what choice player A makes, player E should be able to find values for the $\exists$ variables that satisfy the formula.

Therefore, this game is PSPACE-Complete, as it is literally the TQBF problem.

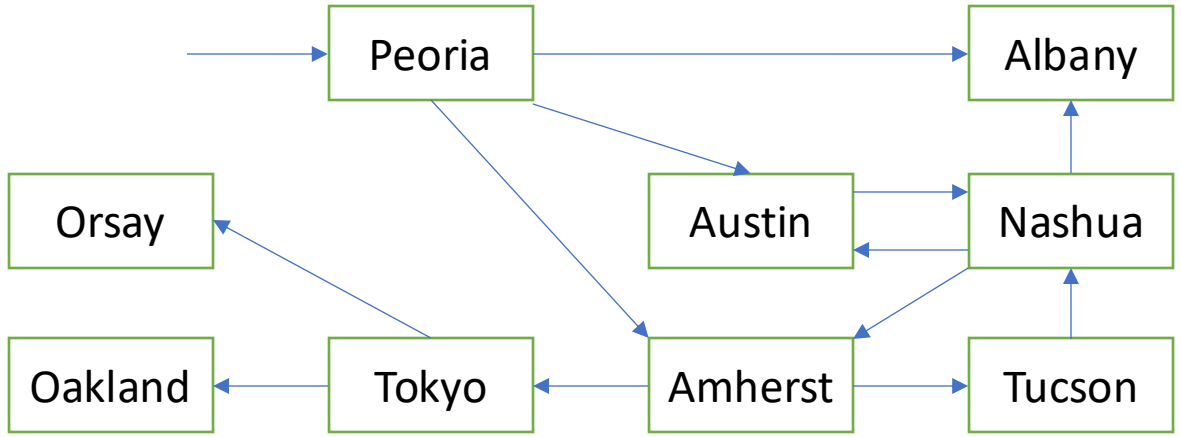
- Most PSPACE-Complete problems are "games".

The Game of Geography

In the game of **Geography**, players take turns naming cities from anywhere in the world.

- Each city's first letter must be the same as the last letter of the previous city named.
- Once a city has been named, it can never be used again.

You can treat this as a graph problem.



Generalized Geography

In the **Generalized Geography** game (GG), you are given a directed graph with a designated start node.

- Players alternate moving a token along an edge to form a path (you can't repeat nodes).
- If a player cannot make a move, that player loses.

GG will be our next PSPACE-Complete problem!

- Most PSPACE-Complete problems are games where you try and figure out who has the winning strategy.
- Chess, Checkers, and Go are on constant-sized boards, but if you generalize them to $n \times n$ boards, they are all PSPACE-Complete, or harder.

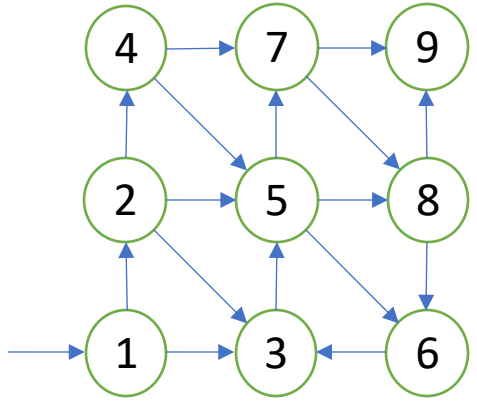
An Example

Who has a winning strategy?

- Player 1: move 3, then 6.

What if we reverse edge (3,6)?

- Player 2 wins.
  - If Player 1 goes to 3, Player 2 goes to 6. Player 2 wins.
  - If Player 1 goes to 2, Player 2 goes to 4.
    - If Player 1 goes to 5, Player 2 goes to 6. Player 2 wins.
    - If Player 1 goes to 7, Player 2 goes to 9. Player 2 wins.



GG ∈ PSPACE

Given a graph G and start node b:

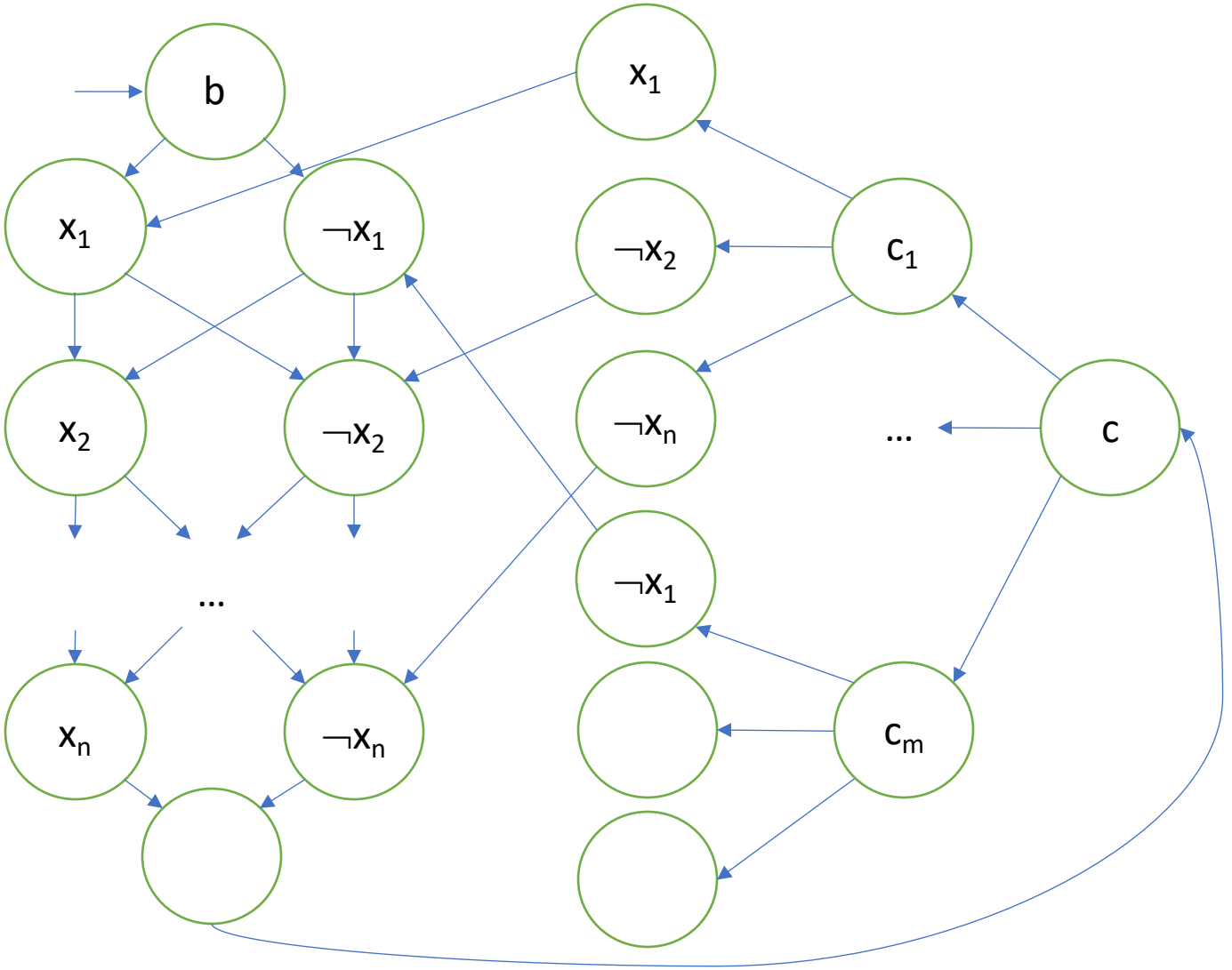
- Our algorithm will accept if Player 1 wins and reject if Player 2 wins.
- If b has out-degree 0, reject.
- Remove b, and all of it's outgoing edges.
- For each node b_i that had an edge from b:
  - Recursively call our algorithm, with start node b_i.
  - We have swapped players with this recursive call, so we will invert the output.
  - If our recursive call rejects (there is a good outcome for Player 1), then accept.
- Reject (all paths lead to a bad outcome for Player 1).

GG is PSPACE-Complete

Reduce from the Formula Game.

- We will assume our input formula starts and ends with $\exists$, and alternates between $\exists$ and $\forall$.
  - We can make our formula conform to this by adding extra quantifiers that bind unused “dummy” variables.
- We will assume the formula is in Conjunctive Normal Form.
  - Any Boolean formula can be transformed to CNF.
- Player E will be Player 1, and Player A will be Player 2.

GG is PSPACE-Complete, Part 2



- Player 2 chooses the clause that is false.
- Player 1 chooses the variable that actually makes that clause true.
- Player 1 (Player E) wins if the formula is true.