CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

Is it harder to TA a course, or to take it?

Logistics

- Quiz and exercise 8 released
- Quiz due Friday April 24 at 11:59pm
- Exercise due Tuesday April 28 11:59pm

Polynomial Time vs Exponential Time

- Polynomial Time: $P = \bigcup_{c \in \{1,2,3,\dots\}} n^c$
  - Shortest Path: linear in the size of the graph, if the graph is size n, then shortest path took $O(n^1)$
  - 2-SAT: if the formula is of size n, then 2-SAT takes time $O(n^3)$
  - P is a complexity class that means "any problem in $TIME\left(O(n^c)\right)$ for a constant c"
- Exponential Time: $EXP = \bigcup_{c \in \{1,2,3,\dots\}} 2^{n^c}$
  - EXP is a complexity class that means "any problem in $TIME\left(O\left(2^{n^c}\right)\right)$ for a constant c"
  - $P \subseteq EXP$
  - 3-SAT and longest path belong to EXP, but we don't know if they belong to P
- A strange pattern:
  - Most "natural" problems are either done in small-degree polynomial (e.g. n^2) or exponential time

Tractability

- Tractable:
  - Feasible to solve in the “real world”
- Intractable:
  - Infeasible to solve in the “real world”
- Whether a problem is considered “tractable” or “intractable” depends on the use case
  - For theory: Tractable = polynomial time, Intractable = Exponential time

P vs NP

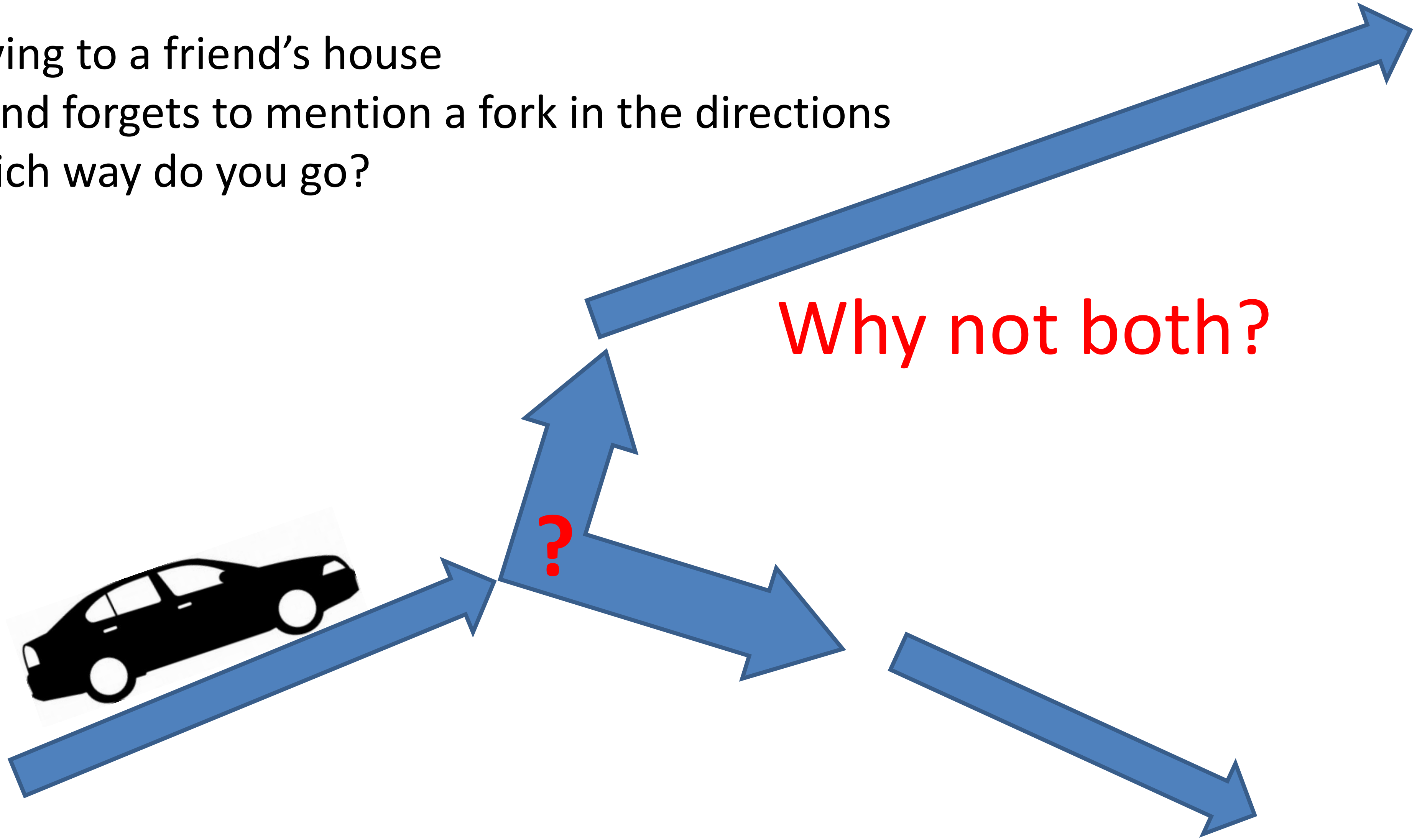
- The class P:
  - Problems that can be solved in polynomial time by a "standard" Turing machine:
    - $P = \bigcup_{c \in \{1,2,3,\dots\}} TIME(n^c)$
- The class NP:
  - Problems that can be solved in polynomial time by a non-deterministic Turing machine:
    - Correctness of a solution can be *verified* in polynomial time given a "witness"

Nondeterminism

Driving to a friend's house

Friend forgets to mention a fork in the directions

Which way do you go?



$P \subseteq NP$

- Why?
  - Deterministic machines are a "special case" of non-deterministic machines that don't use their "power"
  - Any Turing machine that is a polynomial-time deterministic machine is also a polynomial-time non-determinstic machine

Last class

- Shortest Path = "Linear time " $O(|V| + |E|)$
  - Belongs to P
  - Belongs to NP
- Longest Path = "exponential time" $O(|V|!)$
  - Belongs to NP
- 3-SAT = "exponential time" 2^n
  - Belong to NP
- 2-SAT = "polynomial time" n^3
  - Belongs to P
  - Belongs to NP

Longest Path $\in NP$?

- Longest Path: Given a graph G, start node s, end node t, and a number n, is there a simple path from s to t of length at least n?
- Solving with a non-deterministic Turing Machine:
  - At each node, non-deterministically go to each of its neighbors. If a path (explored in parallel using mon-determinism) reaches t and has length $\geq n$, return True.
  - Time: $|V|$

Longest Path $\in NP$?

- Longest Path: Given a graph G, start node s, end node t, and a number n, is there a simple path from s to t of length at least n?
- ^ Decision problem
  - Is there a path of length |V|?
- Verifying a witness:
  - Witness: an example path.
  - To verify: check that the path is a valid path from s to t, check that it is at least length n

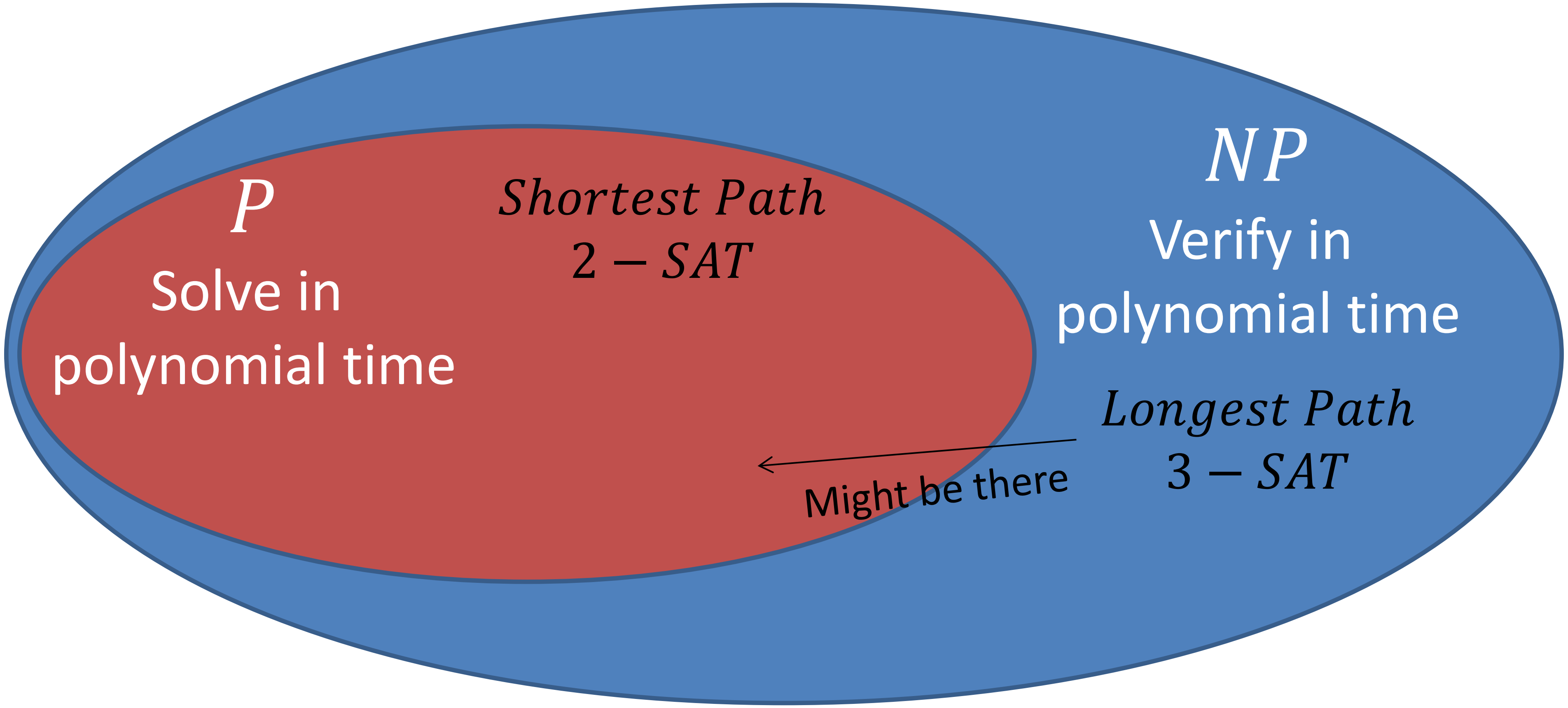
3-SAT $\in NP$?

- Given a 3-CNF formula (logical AND of clauses, each an OR of 3 variables), Is there an assignment of true/false to each variable to make the formula true?
- Solving with a non-deterministic Turing Machine:
  - Try all assignments in parallel using non-determinism. Evaluate the formula for each assignment
  - Time: $O(n)$ for n variables

3-SAT $\in NP$?

- Given a 3-CNF formula (logical AND of clauses, each an OR of 3 variables), Is there an assignment of true/false to each variable to make the formula true?
- Verifying a witness:
  - Witness: an example assignment
  - To verify: Check that all variables have a T/F value, check that the formula evaluates to True. $O(n)$ for n variables.

Overview


P
Solve in
polynomial time
$Shortest\ Path$
$2-SAT$
NP
Verify in
polynomial time
$Longest\ Path$
$3-SAT$
Might be there

Intuitive Restatement of P vs NP

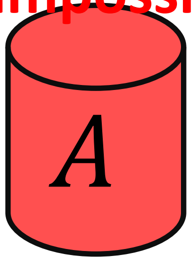
- Are the problems that are easy to verify also easy to solve?
  - Cure cancer
  - Manufacture antibodies for COVID-19
    - Protein folding is in NP
- Most people believe: No…
- Why do we care?

How do we show it?

- To show $P \subset NP$
  - Show that there is at least one NP problem that has no polynomial time standard Turing machine
  - Why is this hard?
- To show $P = NP$
  - Show that EVERY NP problem will also have a polynomial time standard Turing machine
  - Why is this hard?
- Solution: Reductions!

Reductions so far

1) If This is impossible



Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

2) And this is possible

Aim duct at door, insert keg

Put fire under the Keg

Reduction

B

Lighting a fire

If

3) Then this is impossible

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Reductions so far

3) Then this is possible

A

Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

2) And this is possible

Aim duct at door, insert keg

Put fire under the Keg

Reduction

B

Lighting a fire

1) If this is possible

If

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Polynomial Time Reductions

1) If this is slow

A

Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

2) And this is fast

Aim duct at door, insert keg

Put fire under the Keg

Reduction

B

Lighting a fire

3) Then this is slow

If

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Polynomial Time Reductions

3) Then this is fast

A

Opening a door

2) And this is fast

Aim duct at door, insert keg

Put fire under the Keg

Reduction

B

Lighting a fire

1) If this is fast

If

Solution for $\boldsymbol{B}$
Alcohol, wood, matches

Solution for $\boldsymbol{A}$
Keg cannon battering ram

How does this help?

- If we can show that problem A is slow, then anything A "efficiently" reduces to is also slow
- If we can show that problem B is fast, then anything that "efficiently" reduces to B is also fast
- If A "efficiently" reduces to B and B "efficiently" reduces to A, then they're either both fast or both slow
- Idea: Find a group a problems, all of which efficiently reduce to the others. If you can answer "is this efficiently solvable" for any of them, the answer is the same for all of them.

NP-Complete

- This class of "all are efficient or else none of them are" problems
- Problems that are within class NP
- Are also within class $NP - Hard$
  - $NP - Hard$: class of problems such that you can "efficiently" reduce ANY NP problem to them

P vs NP, Formally

- "Efficient" means "(deterministic)polynomial time"
- A $\geq_P B$ means "A polynomial-time reduces to B"
  - A polynomial time solver for B allows for a polynomial time solver for A
  - There is a polynomial time reduction from A to B (the pink box can be done in polynomial time)
- $NP - \mathrm{Hard}$ = The set of all problems B such that for every problem $A \in NP, A \geq_P B$
  - All problems that are "at least as hard as any NP problem"
- $NP - Complete$ = $NP \cap NP - Hard$
  - "The Hardest problems in NP"

NP-Hard

1) If this is $NP - Hard$

B

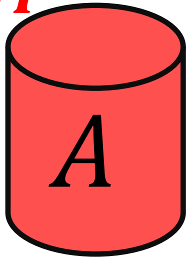
Lighting a fire

If

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

2) And this is anything in NP



Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

3) There's a way to do this in polynomial time

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Why NP-Hard is helpful

4) Anything in NP can be done in polynomial time

A

Opening a door

Solution for $\boldsymbol{A}$
Keg cannon battering ram

3) Then since this is polynomial time

Aim duct at door, insert keg

Put fire under the Keg

Reduction

1) If this is $NP-Hard$

B

Lighting a fire

2) And There's a way to do this in polynomial time

If

Solution for $\boldsymbol{B}$
Alcohol, wood, matches

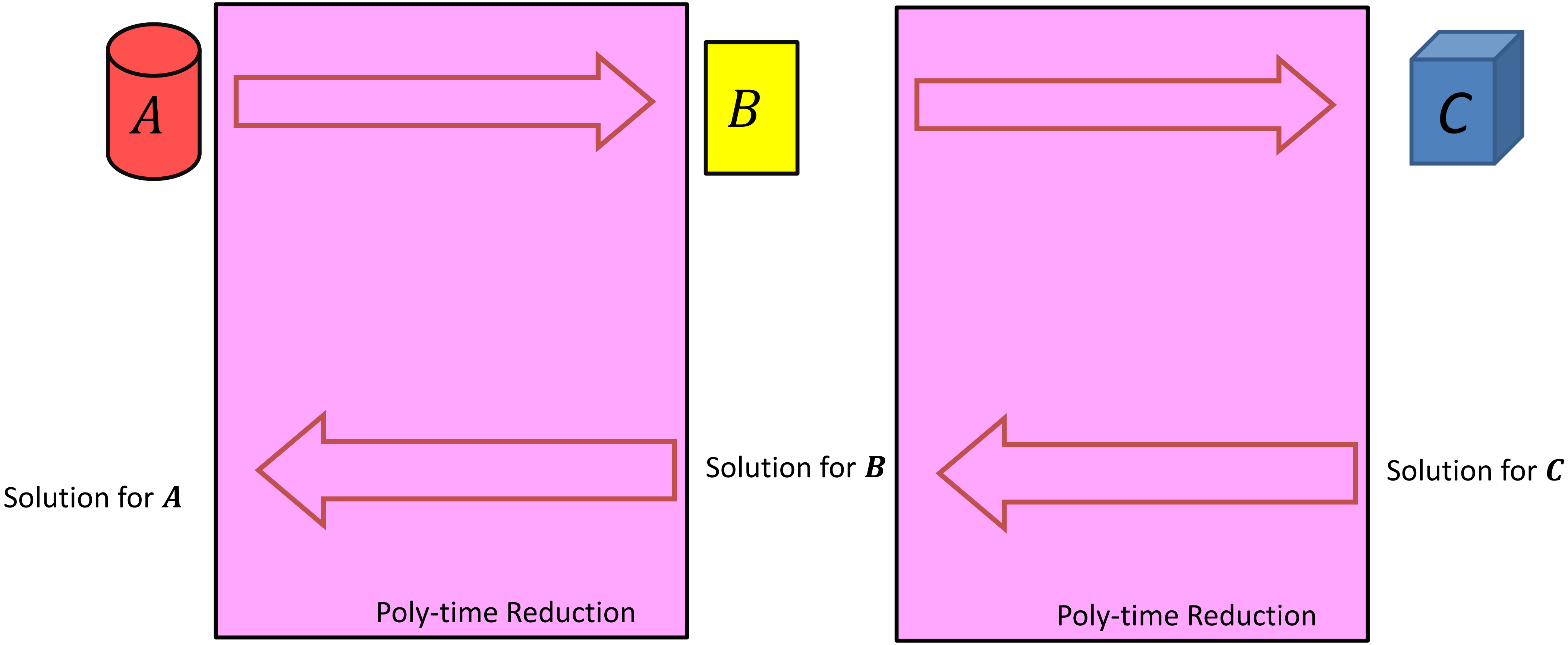
NP-Completeness

- $NP - Complete$ = $NP \cap NP - Hard$
  - "The Hardest problems in NP"
- If some $NP - Complete$ problem belongs to P:
  - Since it's $NP - Hard$, it is the only missing piece in a polynomial time solution to every NP problem
  - $P = NP$
- If some $NP - Complete$ problem does not belong to P
  - Since it's NP, we have an example of an NP problem not in P
  - $P \neq NP$ (and none of the $NP - Complete$ problems have polynomial time solutions)

Showing $NP - Completeness$

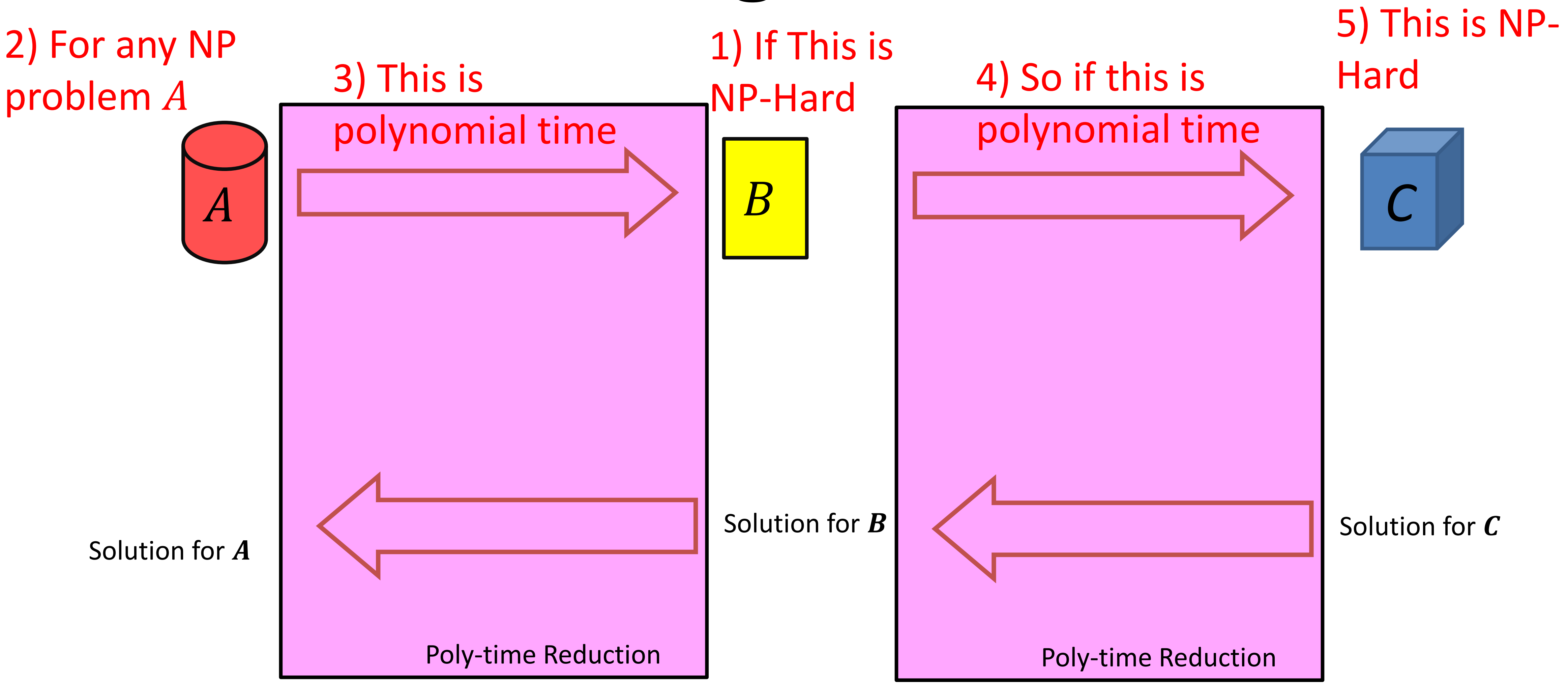
- Show that the problem belongs to NP
  - Give a polynomial time verifier
- Show that the problem belongs to $NP - Hard$
  - Show that EVERY NP problem polynomial-time reduces to it
  - Show that some known $NP - Hard$ problem reduces to it (why?)

"Chaining" Reductions



$A \geq_P B$ and $B \geq_P C$ means $A \geq_P C$

Showing NP-Hard



$A \geq_P B$ and $B \geq_P C$ means $A \geq_P C$

Showing NP-Hardness

- To show C is NP-Hard, reduce a known NP-Hard problem to it.
- The one thing missing?
- An already-known NP-Hard problem

3-SAT is NP-Hard

- Cook-Levin Theorem:
  - Any non-deterministic polynomial time Turing machine, input pair can be converted to a 3-CNF formula such that the formula is satisfiable if and only if the Turing machine accepts the input
  - You can use a 3-CNF formula to simulate a non-deterministic Turing machine in polynomial time

Another NP-Complete Problem

4-SAT: given a 4-CNF formula, is it satisfiable?

Show 4-SAT belongs to NP.

2-SAT: Is this 2-CNF formula satisfiable?
3-SAT: Is this 3-CNF formula satisfiable?
4-SAT: is this 4-CNF forumla satisfiable?
Give a boolean formula in CNF with exactly 4 variables per clause, is that formula satisfiable?

Verifying a solution in polynomial time:
If I have some example assignment of T/F to each variable, determine whether that was a satisfying assignment.
How: Plug in T/F for each variable, evaluate the formula, check if it’s True.

Another NP-Complete Problem

4-SAT: given a 4-CNF formula, is it satisfiable?

Show 4-SAT belongs to NP-Hard.

How do we show this? Show that some NP-Hard problem reduces to it (in polynomial time).

Reduce 3-SAT to 4-SAT in polynomial time.
We want to show that solving 4-SAT allows us to solve 3-SAT, use a 4-SAT solver to solve 3-SAT

Add a variable that can't be true to each clause

$$(f \vee x \vee y \vee z) \wedge (f \vee x \vee \bar{y} \vee y) \wedge (f \vee u \vee y \vee \bar{z}) \wedge (f \vee z \vee \bar{x} \vee u) \wedge (f \vee \bar{x} \vee \bar{y} \vee \bar{z}) \wedge (\bar{f} \vee \bar{f} \vee \bar{f} \vee \bar{f})$$

CAUTION

Tempting (but incorrect) argument: We know 2-SAT is P, 3-SAT is NP-Hard. More variables per clause makes the problem more difficult (since we need to solve 3-SAT in order to solve 4-SAT), so 4-SAT must also be NP-Hard

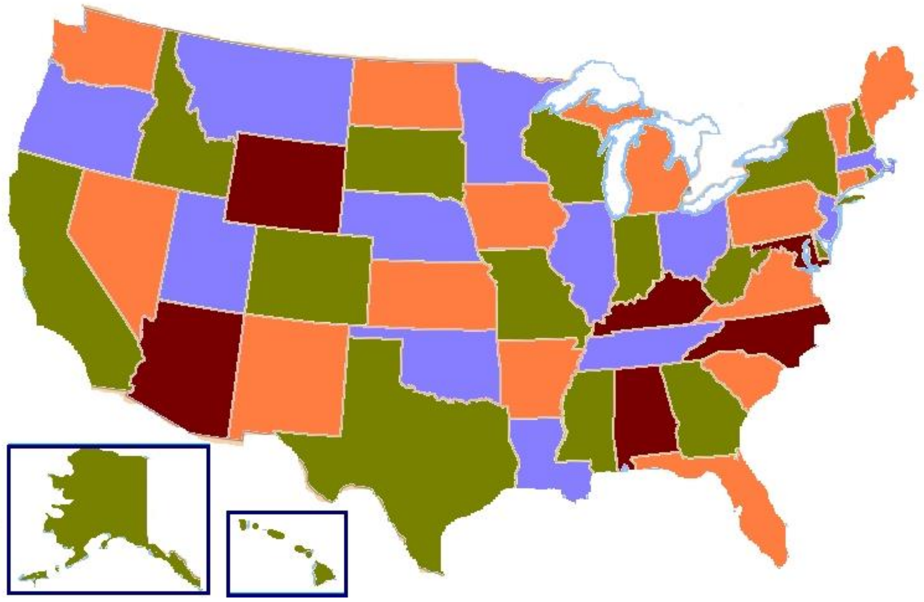
Problem? “we need to solve 3-SAT in order to solve 4-SAT” is hard to defend.

Instead use a reduction. “In the time it takes to solve 4-SAT, we could have solved 3-SAT”

Why "A is necessary for B" is dangerous

Map Coloring: Given a geographical map of states/countries, can I give each region a color so that no bordering regions share their colors?

1-colorable? **Trivial (yes iff no bordering regions)**
2-colorable? **Easy (BFS works)**
3-colorable? **NP-Hard**
4-colorable? **Trivial (answer is always yes)**



Procedure for showing $A \leq_p B$

1. Start with an instance x_a of problem A
2. Find an algorithm R to convert x_a into x_b and instance of B
3. Show that R takes polynomial time
4. Show that if $B(x_b) = 1$ then $A(x_a) = 1$
5. Show that if $B(x_b) = 0$ then $A(x_a) = 0$
    1. More often: if $A(x_a) = 1$ then $B(x_b) = 1$ (i.e. contrapositive)

3-SAT $\leq_P$ Longest Path

3-SAT

$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y)$
$\wedge (u \vee y \vee \bar{z})$
$\wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y}$
$\vee \bar{z})$

Solution for 3-SAT

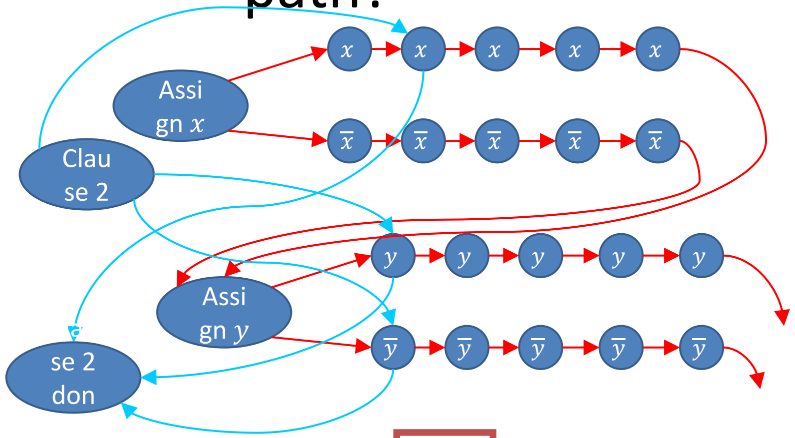
Solution exists if and only if there is a “long enough” path

Create a graph

Is that length long enough?

Reduction

How long is the longest simple path?



How?

Get the length of the longest path

Procedure for showing 3-SAT $\leq_p$ Longest Path

1. Start with a 3-CNF formula F
2. Find an algorithm R to convert F into a graph, start node, end node, G, s, t
3. Show that R takes polynomial time
4. Show that if 3SAT$(F) = 1$ then LP$(G, s, t) \geq n \cdot m + n + 3 \cdot m$
5. Show that if LP$(G, s, t) \geq n \cdot m + n + 3 \cdot m$ then 3SAT$(F) = 1$

Converting 3-SAT to longest path

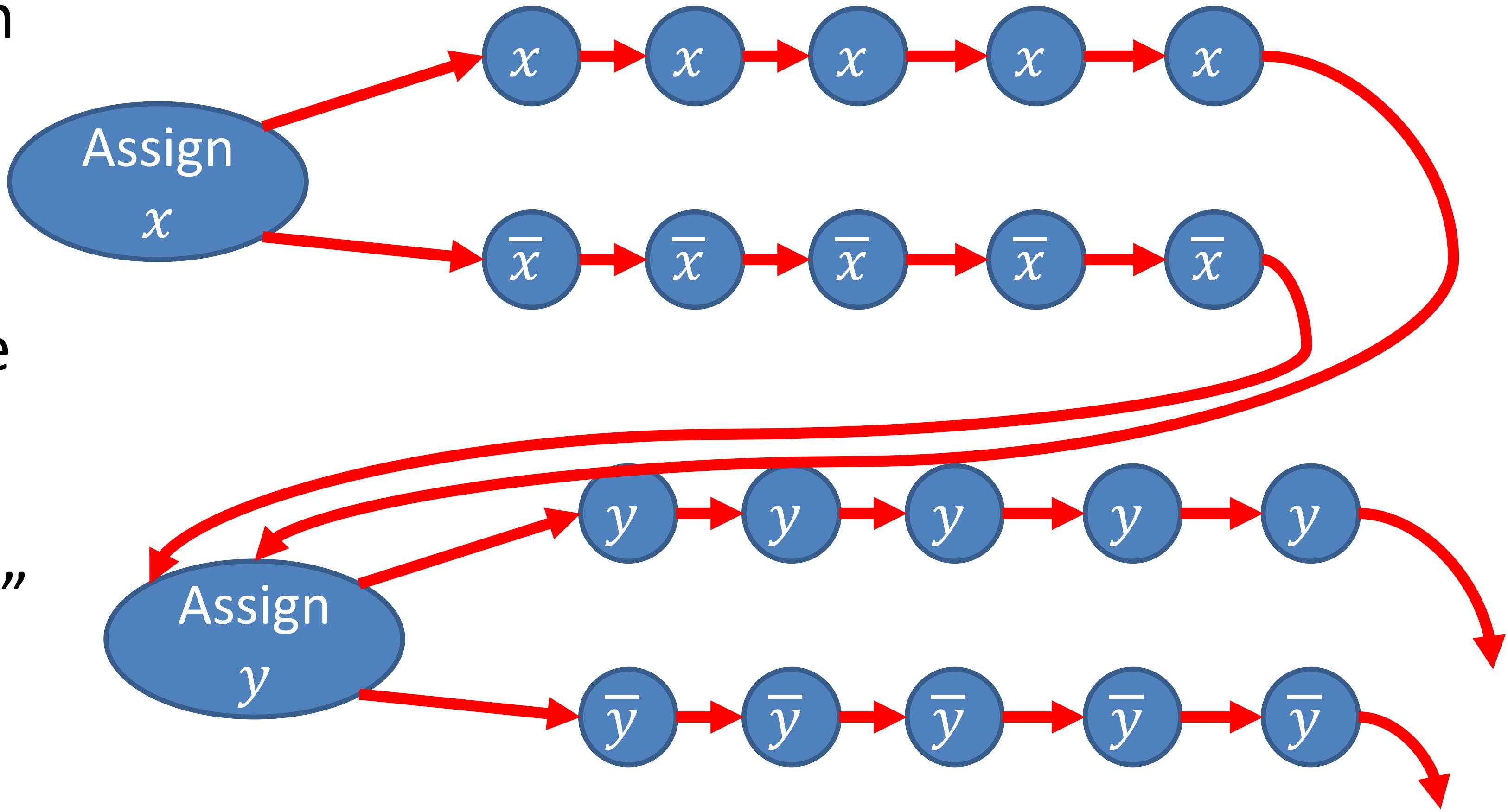
$$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Make one "red chain" for each variable and negation, chain length is number of clauses

Idea: To assign $x = True$, take the low path

Now we've visited each "False" node once

We can't visit those again and have a simple path

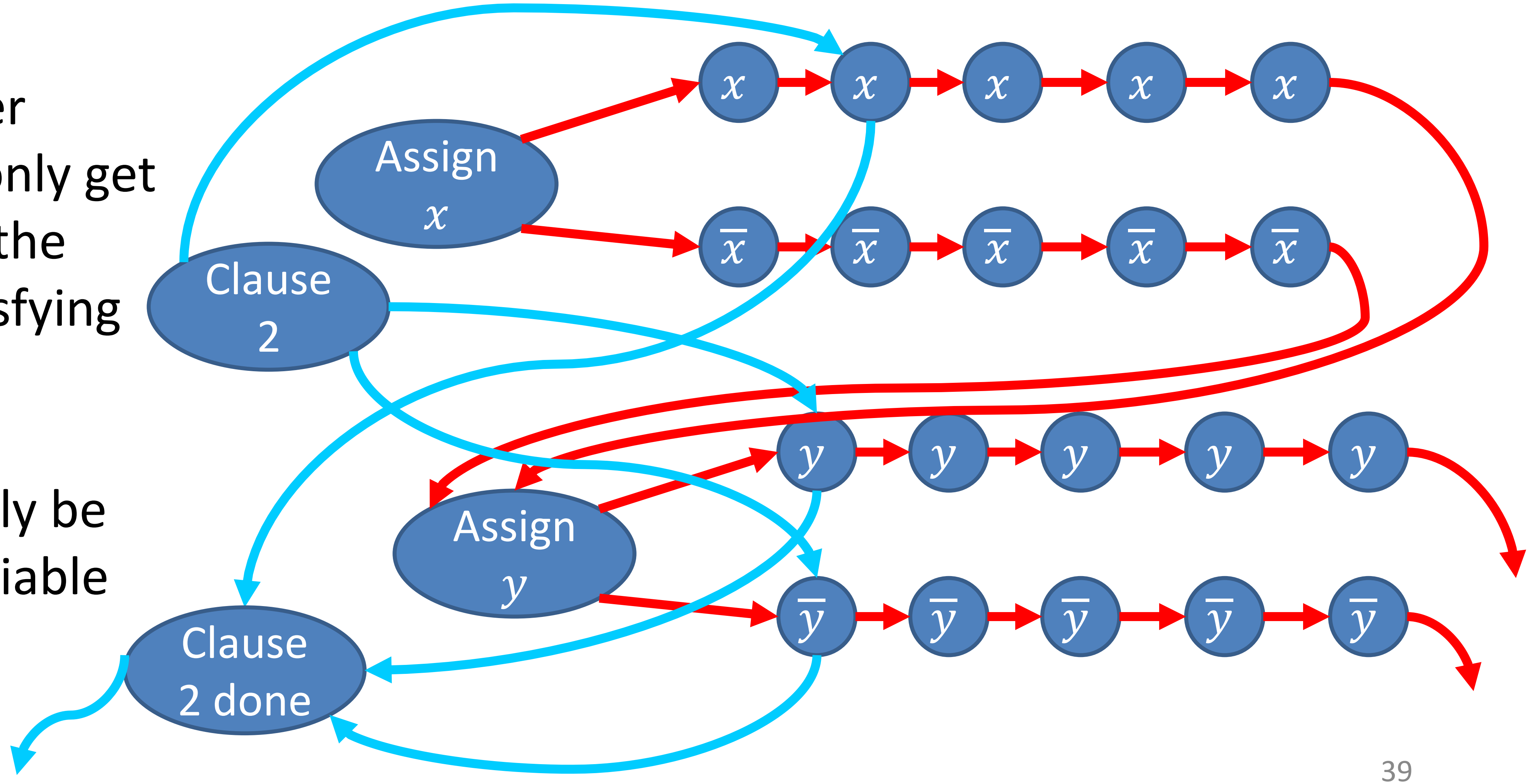


Converting 3-SAT to longest path

$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$

Make 2 nodes per clause. You can only get from the first to the second via a satisfying variable's node

That path will only be simple if that variable was True



n = #variables
m = #clauses

How Long Is the Longest Path?

- Start from the "assign x" node, end with "clause m done"
- If the formula is satisfiable:
  - Pick the chains to assign true/false to each variable in accordance with a satisfying assignment
    - m variables per chain, n chains, plus one more node per variable
    - $n \cdot m + n$ nodes total
  - Traverse through each clause, picking an unvisited node (meaning that variable was true)
    - m clauses, each with 3 nodes
    - $3 \cdot m$ nodes total

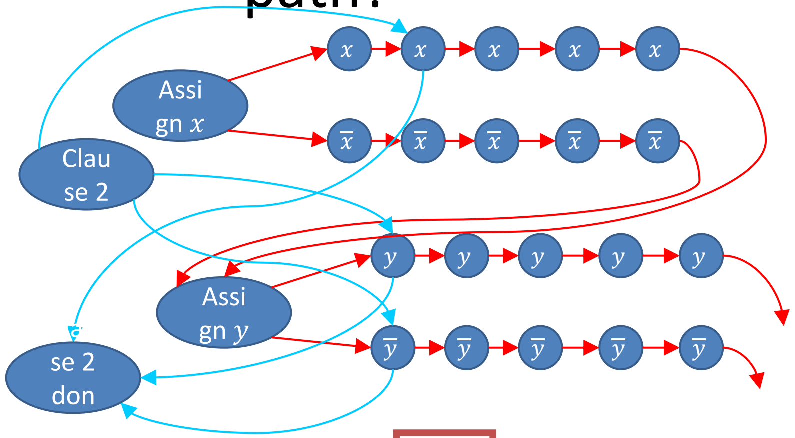
3-SAT $\leq_P$ Longest Path

3-SAT

$(x \vee y \vee z) \wedge (x \vee \bar{y} \vee y)$
$\wedge (u \vee y \vee \bar{z})$
$\wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y}$
$\vee \bar{z})$

Create a graph

How long is the longest simple path?



How?

Get the length of the longest path

Is that length $n \cdot m + n + 3 \cdot m$?

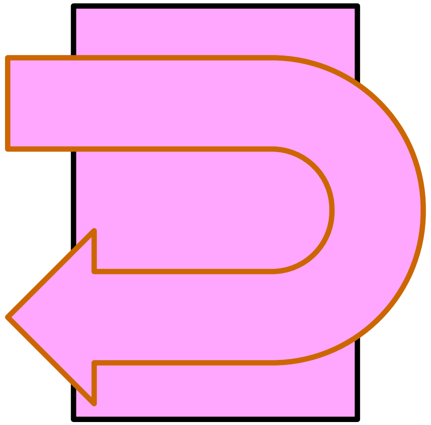
Solution for 3-SAT

If it is, the formula was satisfiable

Reduction

What can we conclude?

- If we found a polynomial time solution for Longest Path:
  - This procedure is a polynomial time solution for 3SAT



- If we somehow knew that it was impossible to find a polynomial time solution for 3SAT
  - We could never find a polynomial time solution for Longest Path