CS 161: Design and Analysis of Algorithms

NP-Complete I

- P, NP
- Polynomial time reductions
- NP-Hard, NP-Complete
- Sat/ 3-Sat

Decision Problem

- Suppose there is a function A that outputs True or False
- A **decision problem** is a problem of the form "is A(x) = True?"
- Example: A(G,s,t,len) = True if and only if G has a path from s to t of length at most len

P

- **P** is the class of all decision problems that are solvable in polynomial time ($O(n^c)$ for some c) in the size of the input
- Example: To compute A(G,s,t,len), compute the shortest path from s to t in G, and check if its length is at most len

P

- Example: A(LP,c) = True if and only if LP has a solution attaining a value of at least c
- The problem of determining if A(LP,c) = True is in P since we can always solve the linear program, and check that the value is at least c

Is Polynomial Time the Same as Efficient?

- If some problem was solvable in time $O(n^{1000})$, it would be extremely hard to solve, but still in P
- However, for large n, $O(n^c)$ is still much better that $O(d^n)$
- Good property: polynomials are closed under composition

Binary Relation

- A **binary relation** is a function R(x,y) that outputs True or False

Search Problem

- A binary relation R specifies a **search problem**
  - Given an input x, determine if there is a y such that R(x,y) = True
  - If there is, output such a y

NP

- **NP** = set of decision problems A such that there exists a search problem R_A where:
  - A(x) = True if and only if there is some y such that $R_A(x,y)$ = True
  - $R_A(x,y)$ is computable in polynomial time
- y is called a **witness** that f(x) is True

NP

- Example: A((G,c)) = True if and only if G has a tour T with total length at most c
  - R_A((G,c), T) = True if and only if T is a tour of G with total length at most c
  - While we don’t know how to actually compute such a T, we can easily check that T is a tour of length at most c

NP

- Example: Any problem in P is in NP
  - $R_A(x,-) = A(x)$

Decision vs Search

- NP is technically defined as a class of decision problems: “Does G have a minimum spanning tree with weight at most W?”
- Often, we abuse notation and say that the search problem is in NP: “Find a spanning tree of G with weight at most W”
- For many problems, possible to show that decision and search are essentially the same

P vs NP

- P is the set of problems solvable in polynomial time
- NP is the set of problems whose solutions can be checked in polynomial time
- Does P = NP?
  - Seems unlikely that every problem that can be checked in polynomial time can also be computed in polynomial time

Polynomial Time Reductions

- Recall that a reduction from problem A to problem B consists of two components:
  - A conversion from an instance of problem A into an instance of problem B
  - A conversion from a solution for the instance of problem B into a solution for the original instance

Polynomial Time Reductions

- We will be more precise now:
- A decision problem A is polynomial-time reducible to B if:
  - We can efficiently convert any instance x of A into an instance x' of B
  - A(x) = True if and only if B(x') = True
- We write $A \leq_P B$

Polynomial Time Reductions

- **Theorem**: if $A \leq_P B$ and B is in P, then A is in P
- Proof: Given an instance x of A, use the reduction to get an instance x' of B. Then solve B using a polynomial time algorithm

NP-Complete

- What if there was some problem B in NP such that $A \leq_P B$ for all A in NP?
- If B is in P, then all A are in P, so P = NP
- If B is not in P, then clearly P ≠ NP
- If such a B exists, we have reduced the problem of deciding if P = NP to deciding if B is in NP

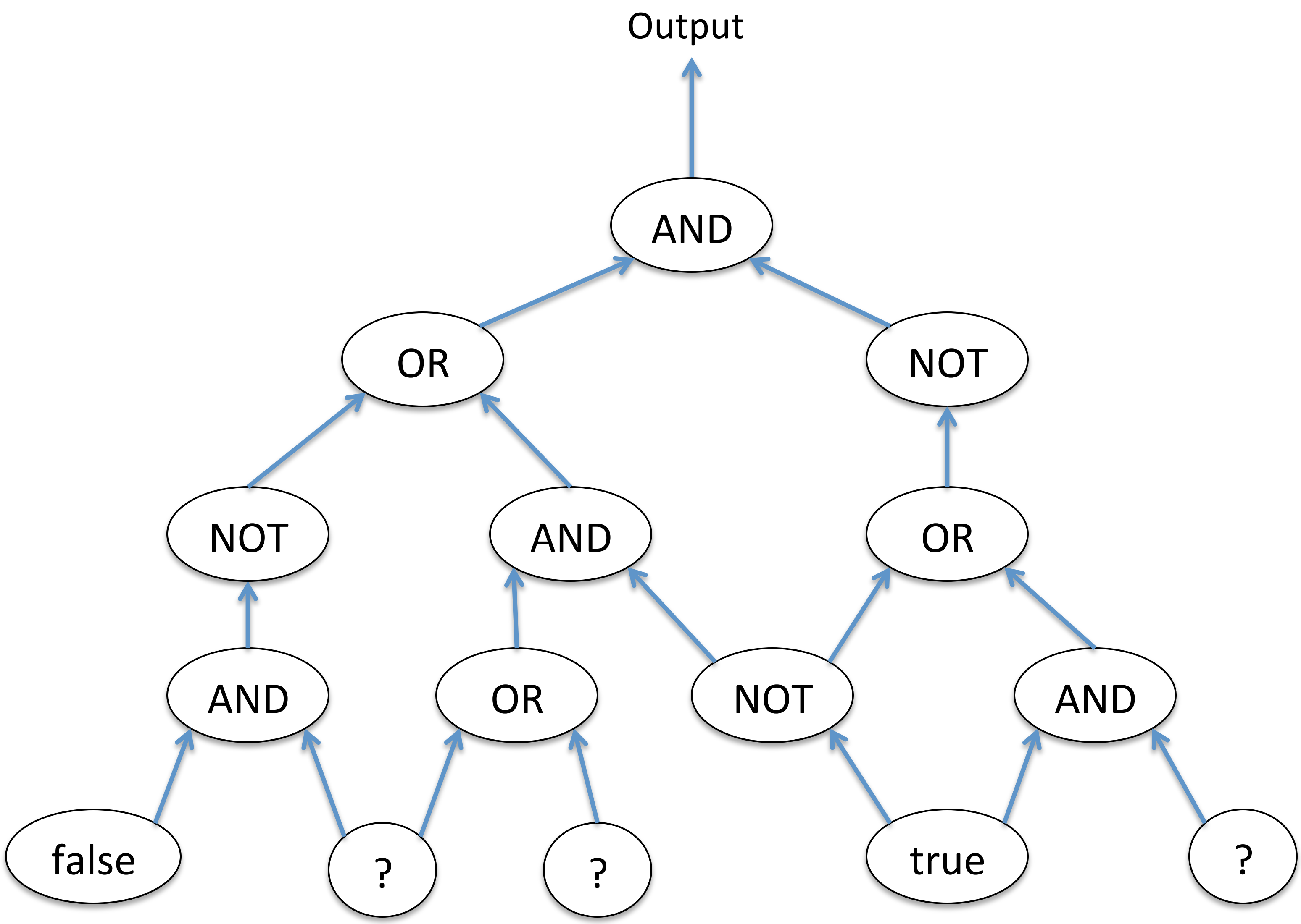
NP-Complete

- A decision problem B is **NP-Complete** if B is in NP and $A \leq_P B$ for all A in NP
  - Informally: B is as hard as the hardest problems in NP
- A problem C is **NP-Hard** if $A \leq_P B$ for all A in NP
  - In formally: C is at least as hard as the hardest problems in NP

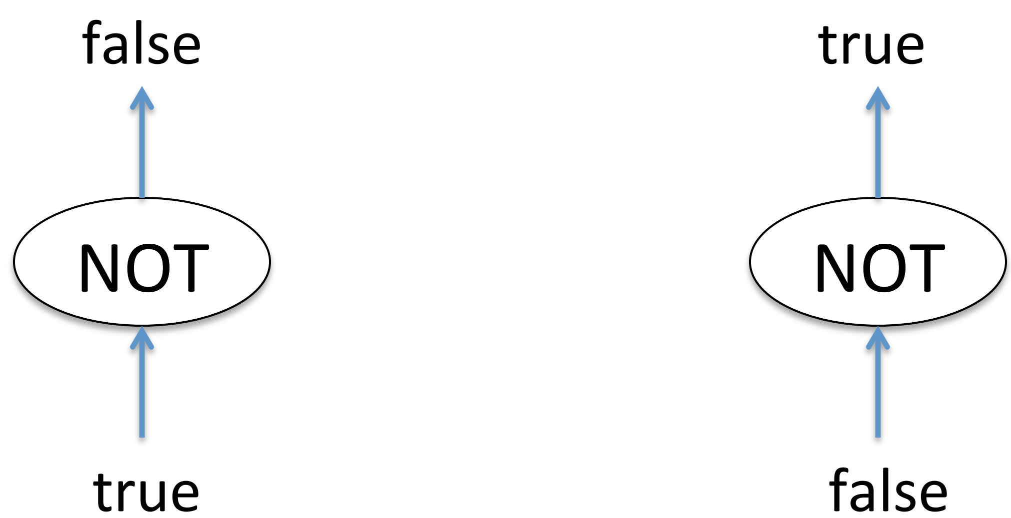
Do NP-Complete Problems Exist?

- At first glance, the existence of NP-Complete problems seems unlikely
- How can one problem be reducible from a the entire class of infinitely many problems?

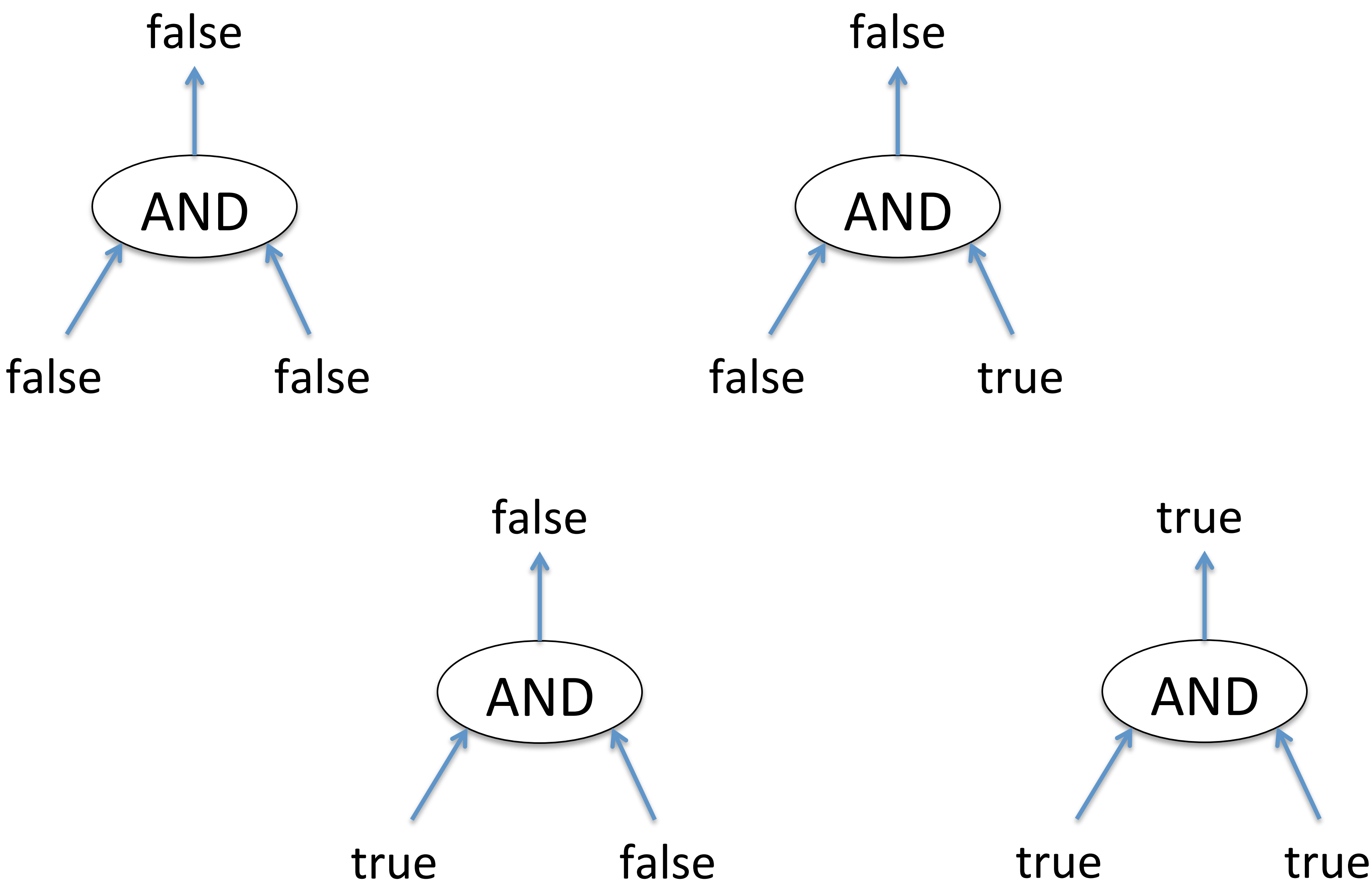
Boolean Circuit



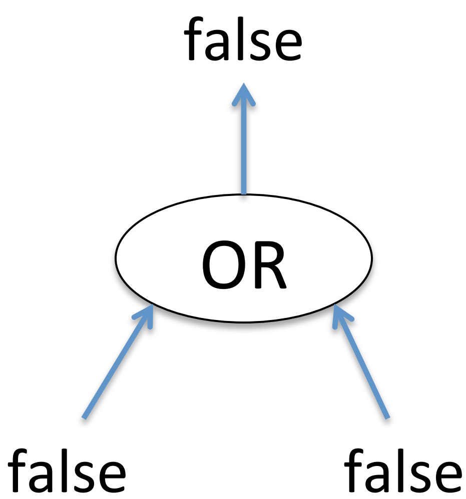
Boolean Circuit

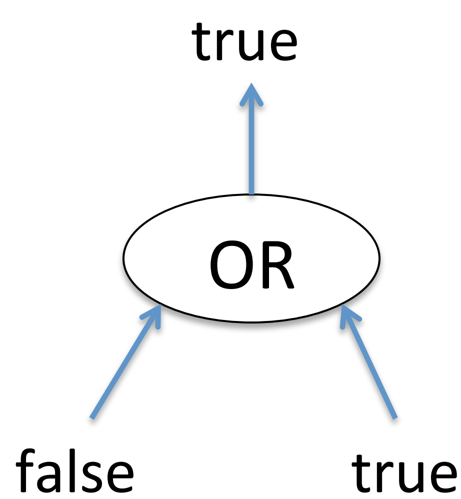


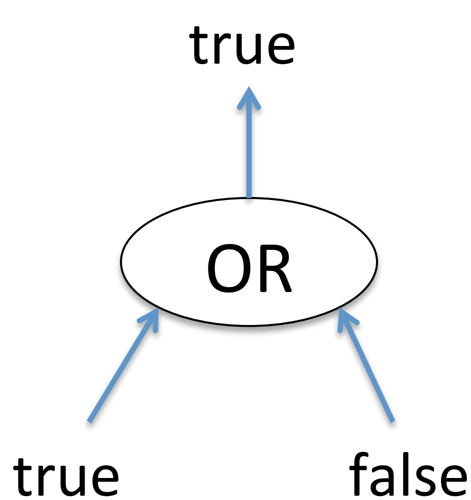
Boolean Circuit

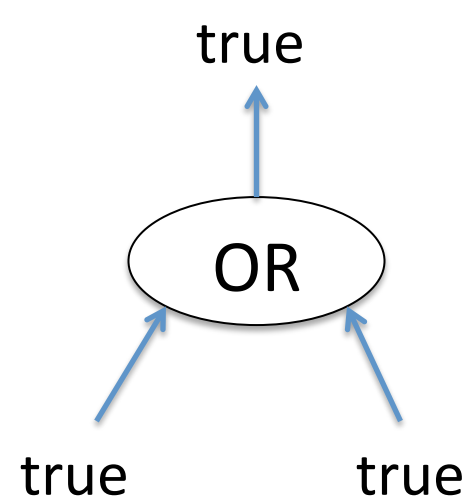


Boolean Circuit



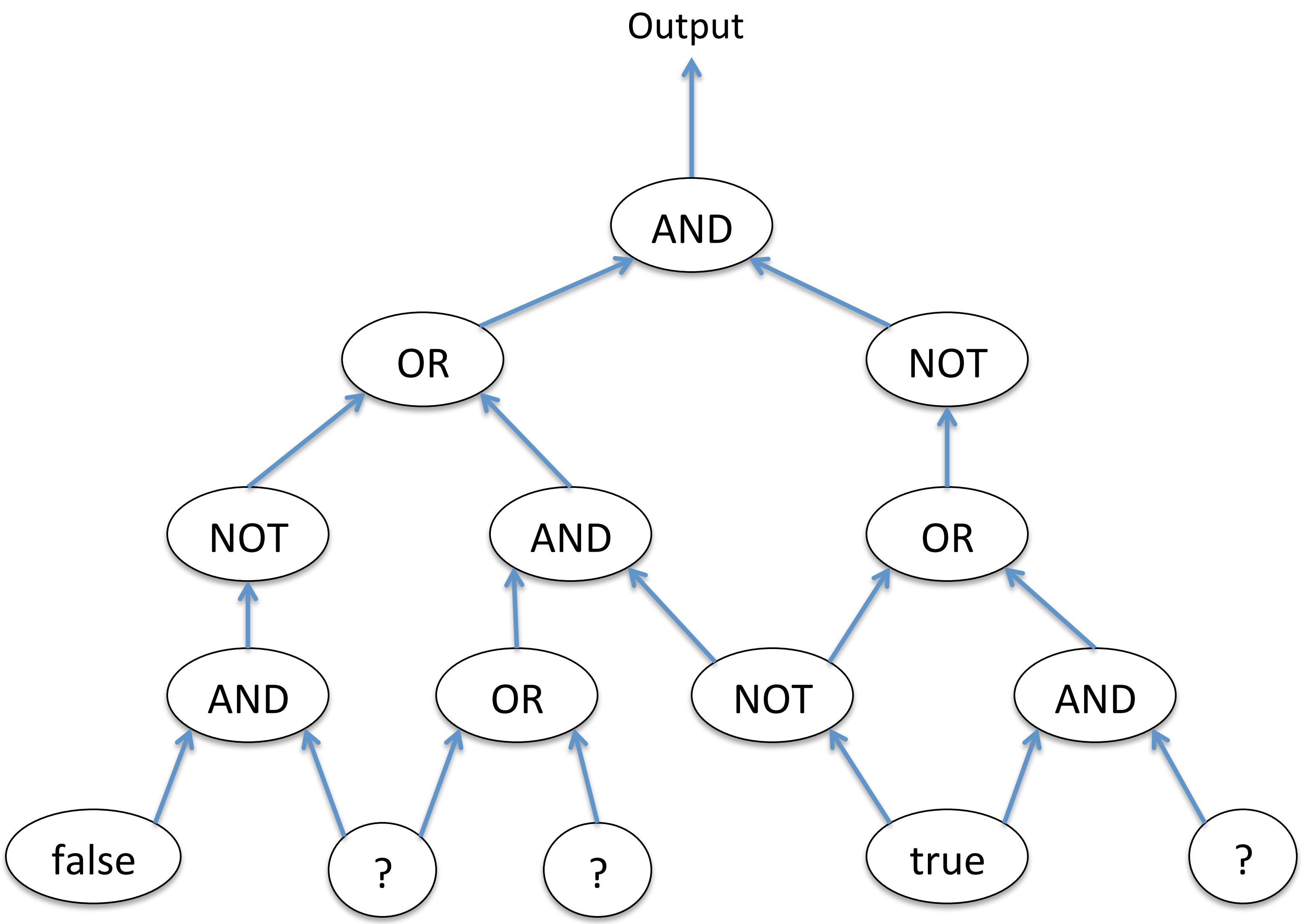






Boolean Circuit

Boolean Circuit



Circuit SAT

- Given a boolean circuit C, is there a setting of the unknown inputs that makes the circuit evaluate to "true"?
- Clearly, Circuit SAT is in NP: we can check whether a setting of the unknown inputs leads to a "true" by evaluating the circuit

Circuit SAT is NP-Complete

- Theorem: Given any NP problem A, we have that A $\leq_P$ Circuit SAT

Proof

- Our NP problem A has an efficiently computable binary relation R such that A(x) = True if and only if there is a y such that R(x,y) = True

Proof

- R is computable in polynomial time
- R can be represented as a boolean circuit!
  - The computer that runs R is a boolean circuit Circ on a chip
  - Since R runs in polynomial time, R can be rendered as a boolean circuit consisting of a polynomial number copies of Circ, one per unit of time
  - Values of gates in one copy used to compute values in next

Proof

- We have a boolean circuit C that computes R
- A(x) = True if and only if there is a y such that C(x,y) evaluates to true
- Let the circuit C_x be the circuit C, with the values for x hardwired
- Then C_x has a satisfying assignment if and only if there is a y that makes R(x,y) = True

Proof

- Therefore, for any NP problem A, we have the following reduction to Circuit SAT:
  - Construct the polynomial-sized circuit C that checks if R(x,y) = True
  - For instance x, hardwire the x, obtaining the circuit C_x
  - C_x is our instance of the Circuit SAT problem

Satisfiability

- A **boolean formula** is any of the following:
  - A variable: x
  - The negation of a boolean formula: $\overline{x}$
  - The **disjunction** (or) of boolean formulae:
    $$x_1 \lor \overline{x_2} \lor x_3$$
  - The **conjunction** (and) of boolean formulae:
    $$(x_1 \lor \overline{x_2}) \land x_2 \land (\overline{x_1 \land x_3})$$

SAT Problem

- The SAT problem is to, given a boolean formula, find a satisfying assignment, or report that none exists.
- Clearly, SAT is a special case of Circuit SAT

Disjunctive Normal Form

- A variable or its negation are called **literals**
- Any boolean formula can be massaged into the following form **disjunctive normal form (DNF)**: the disjunction of conjunctions of literals

$$(x_1 \wedge x_2 \wedge \overline{x_3}) \vee \overline{x_1} \vee (x_2 \wedge \overline{x_4} \wedge x_5)$$

- Satisfiability of DFS formulas is easy!

Conjunctive Normal Form

- **Conjunctive normal form (CNF)**: conjunction of disjunction of literals

$$(x_1 \vee x_2 \vee \overline{x_3} \vee x_4) \wedge \overline{x_1} \wedge (x_2 \vee \overline{x_4} \vee x_5)$$

- Define a **clause** to be one of the disjunctions

3 SAT

- 3SAT is the satisfiability problem on CNF formula where all clauses have at most 3 literals

$$(x_1 \vee \overline{x_3} \vee x_4) \wedge \overline{x_1} \wedge (x_2 \vee \overline{x_4} \vee x_5)$$

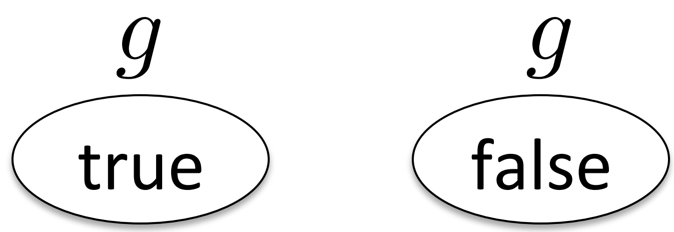
- 3SAT is NP-Complete

Proof

- We will reduce from Circuit SAT
- Given an instance C of circuit say, create a variable g for each gate, representing the output of that gate
- For each gate, we will create one or more clauses that force the variables to be set correctly

Proof

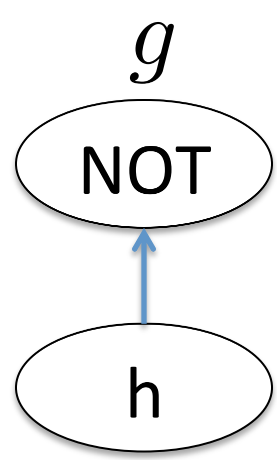
Gate g:



Clauses: (g) $(\overline{g})$

Proof

Gate g:



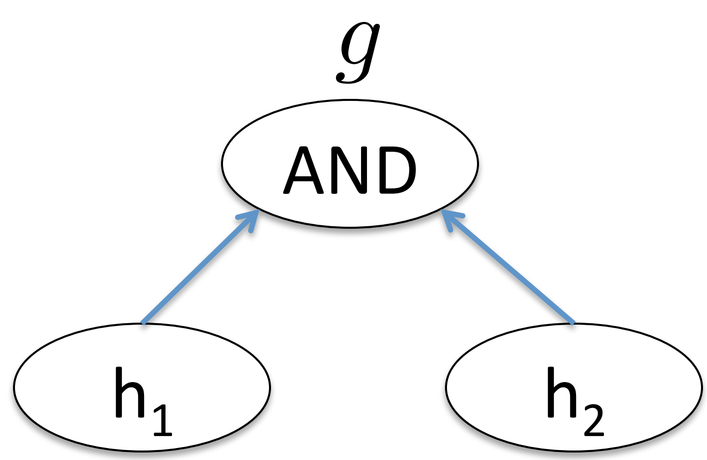
Clauses:

$$(g \vee h)$$

$$(\overline{g} \vee \overline{h})$$

Proof

Gate g:



Clauses:

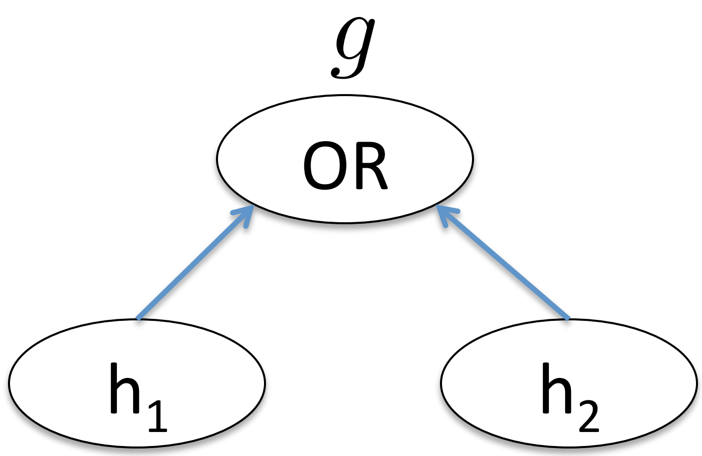
$$(\overline{g} \vee h_1)$$

$$(\overline{g} \vee h_2)$$

$$(g \vee \overline{h_1} \vee \overline{h_2})$$

Proof

Gate g:



Clauses:

$$(g \vee \overline{h_1})$$

$$(g \vee \overline{h_2})$$

$$(\overline{g} \vee h_1 \vee h_2)$$

Proof

- Given a Circuit SAT instance, construct a variable g for each gate
- Create up to three disjuntive clause for each gate that force the outputs of each gate to be correct
- Additionally, if g is the output gate, we add the clause (g), forcing the output of g to be True

Proof

- An assignment satisfies the 3SAT instance if and only if, when we assign the output of each gate the corresponding value:
  - All gates output the correct value
  - The output of the whole circuit is True
- Thus, the Circuit SAT instance has a satisfying assignment if and only if the 3SAT instance does

Proof

- We have exhibited a poly-time reduction form Circuit SAT to 3SAT
- Since Circuit SAT is NP-Complete, and 3SAT is in NP, 3SAT must also be NP-Complete

The Power of NP-Completeness

- We have shown that 3SAT is as hard as any problem in NP
  - If 3SAT has an efficient algorithm, P = NP
  - If not, P ≠ NP
- The general belief is that P ≠ NP
  - If so, *any* NP-Complete problem is hard to solve
  - If you can prove your problem is NP-Complete, you probably shouldn't bother trying to find an efficient algorithm for it

A Less Obvious Reduction

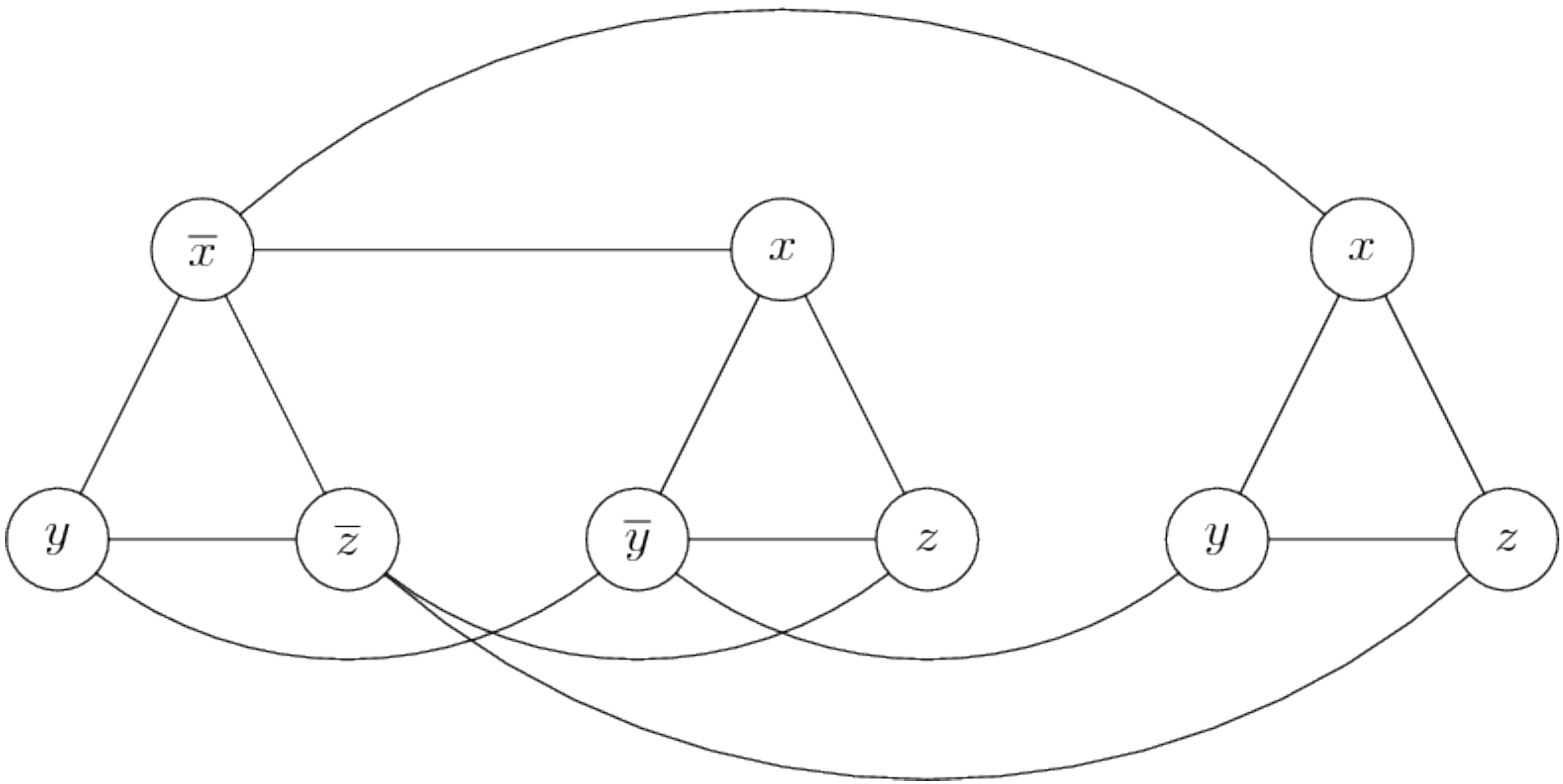
- Recall: an **independent set** of a graph G = (V,E) is a subset of nodes S such that no edge has both endpoints in S
- Independent Set Problem: Given G and a goal k, find an independent set of size k if one exists

A Less Obvious Reduction

- Given an instance of 3SAT (a collection of k clauses $(z_i \vee z_j \vee z_k)$
- Construct a graph as follows:
  - For each clause, create a triangle, where nodes are labeled by the literals in the clause
  - Connect each node to each of the nodes labeled with its negation

A Less Obvious Reduction

$$(\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee z)$$

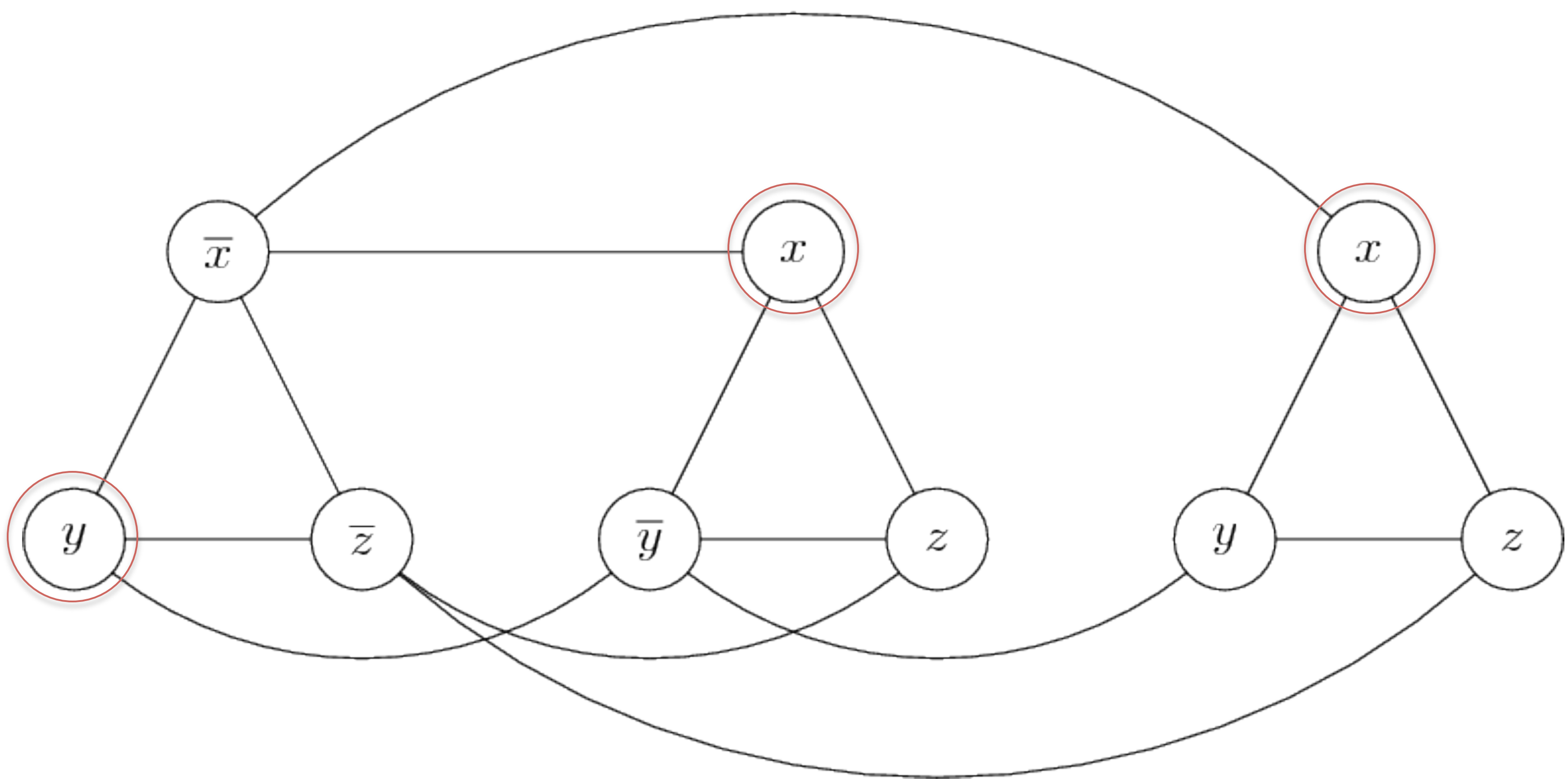


A Less Obvious Reduction

- Suppose the 3SAT instance has a satisfying assignment
- From each triangle, select a true literal
- Result must be independent set of size k

A Less Obvious Reduction

$$(\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee z)$$



A Less Obvious Reduction

- Suppose the graph has an independent set of size at least k
- Then at least one node from each triangle is in the set
  - There can be only one node in each triangle, so the size is at most k
- Set the corresponding literal to true

A Less Obvious Reduction

- Need to show that setting each literal in the independent set to true gives a satisfying assignment, and we never try to set a variable to be both true and false

A Less Obvious Reduction

- Since every literal has an edge to each of its negations, if a literal is in the independent set, none of its negations are
  - We will never try to set a variable to be both true and false
- Since every clause has a literal set to true, every clause is true, and so the 3SAT instance is satisfied

What do P and NP Stand For?

- P stands for polynomial time
- NP? Non-deterministic polynomial time

Non-determinism

- Informally, a non-deterministic algorithm is one that makes many arbitrary decisions
- A non-deterministic algorithm solves the decision problem A if
  - Provided that A(x) = True, there is some sequence of choices that makes the algorithm output True
  - If A(x) = False, no sequence of choices makes the algorithm output True.

Equivalence to Our Definition?

- If a poly-time non-deterministic algorithm solves A, let R(x,y) be the following relation:
  - Run A on input x, and whenever there is an arbitrary decision to make, look at the next chunk of y to make the decision
  - If there is a sequence of decisions that makes our algorithm output True, then there is a y making R(x,y) output True
  - If no such sequence of decisions exist, no such y exists

Equivalence to Our Definition

- If a problem A has a poly-time computable binary relation R(x,y), construct the following non-deterministic algorithm:
  – Run the algorithm for R on input x and an arbitrary choice for the input y

Reminders

- Final August 17th 2:15 – 3:15 in Skilling Auditorium
- Material: through Lecture 20 (Monday)
- SCPD students: welcome to take exam on campus, just let us know by the end of Monday