Hard Problems

When can we not expect efficient algorithms?

Outline

- NP-hardness
  - Definition
  - Why care about NP-hardness?
  - Proving NP-hardness
- NP-completeness
  - Definition
  - Proving NP-completeness
- Expressivity and Hardness
  - Hilbert’s tenth problem

Outline

- NP-hardness
  - Definition
  - Why care about NP-hardness?
  - Proving NP-hardness
- NP-completeness
  - Definition
  - Proving NP-completeness
- Expressivity and Hardness
  - Hilbert’s tenth problem

Learning objectives:
You are able to
- state the definitions of NP-hardness and NP-completeness
- describe a polynomial time reduction
- name three NP-hard problems
- state Hilbert’s tenth problem

NP-hardness

We will finally learn how to get our boss off our backs!

We say a problem is **NP-hard** if it is at least as hard than every single problem in NP.

NP-hardness

We will finally learn how to get our boss off our backs!

We say a problem is **NP-hard** if it is at least as hard than every single problem in NP.

Formally: problem A is NP-hard if for all problems B in NP, B has a polynomial-time reduction to A

NP-hardness

$$\text{Problem } B \xrightarrow{\text{Poly time}} \text{Problem } A$$

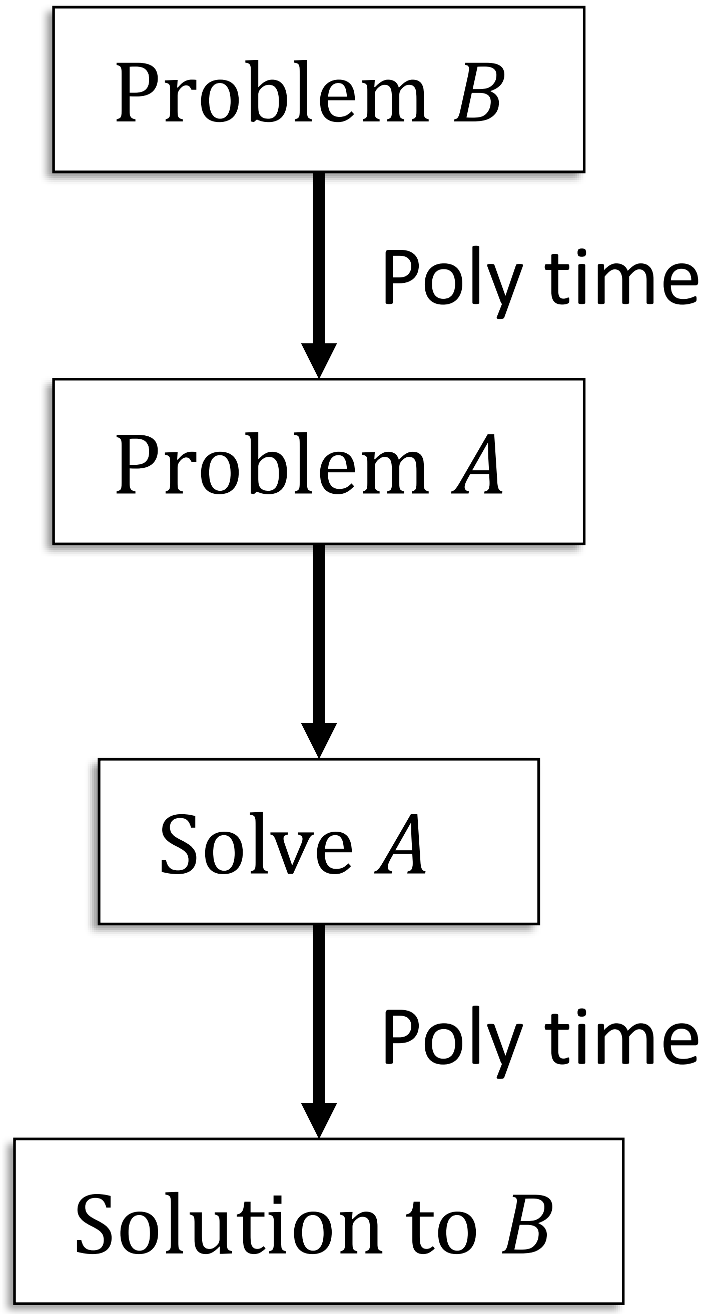
Assume a solver S for problem A

Consider an input instance I_B for problem B

1. Create an input instance I_A for problem A
2. Use S to obtain a solution $S(I_A)$ to I_A

Poly time

NP-hardness



Assume a solver S for problem A

Consider an input instance I_B for problem B

1. Create an input instance I_A for problem A
2. Use S to obtain a solution $S(I_A)$ to I_A
3. Turn $S(I_A)$ into a solution to I_B

Poly time

Why care about NP-hardness?

Assume problem A is NP-hard.

Our boss asks us to solve problem A in polynomial time.

Why use NP-hardness?

Assume problem A is NP-Hard. Imagine we have a polynomial-time solver for A.

Why use NP-hardness?

Assume problem A is NP-Hard. Imagine we have a polynomial-time solver for A.

Because A in NP-hard all problems in NP polynomially reduce to A.

Our imaginary solver can hence solve all problems in NP in polynomial time.

Because all problems in NP have a polytime algorithm (the imaginary one), all problems in NP are in P. This implies our algorithm proves **P = NP.**

Why use NP-hardness?

Assume problem A is NP-Hard. Imagine we have a polynomial-time solver for A.

Because A in NP-hard all problems in NP polynomially reduce to A.

Our imaginary solver can hence solve all problems in NP in polynomial time.

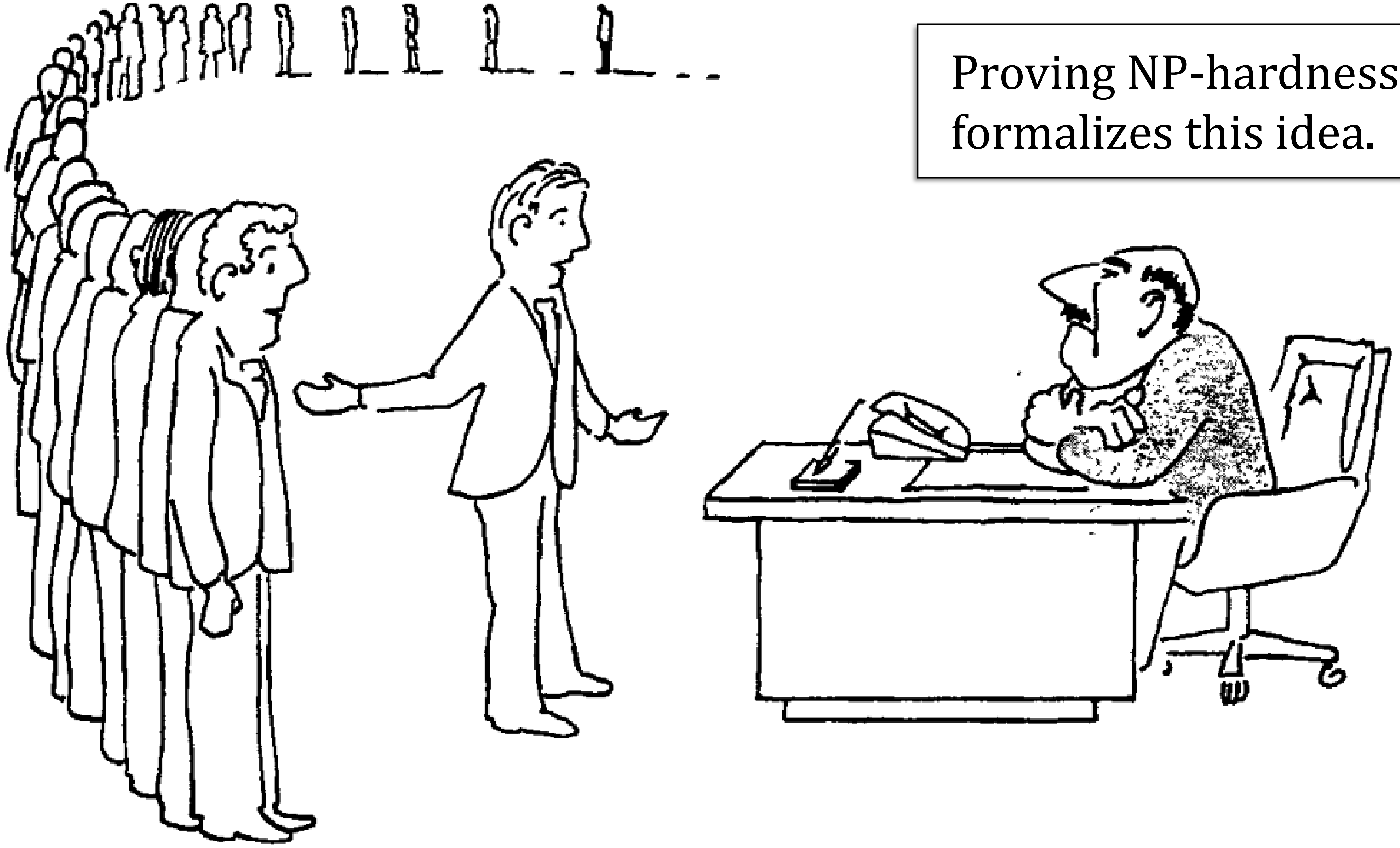
Because all problems in NP have a polytime algorithm (the imaginary one), all problems in NP are in P. This implies our algorithm proves **P = NP.**

This is not impossible.

But it is not reasonable to ask us to do what a century of algorithm experts could not.

Why use NP-hardness?

Proving NP-hardness formalizes this idea.



"I can't find an efficient algorithm, but neither can all these famous people."

Source: Garey, Michael R.; Johnson, D. S. *(1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman*.

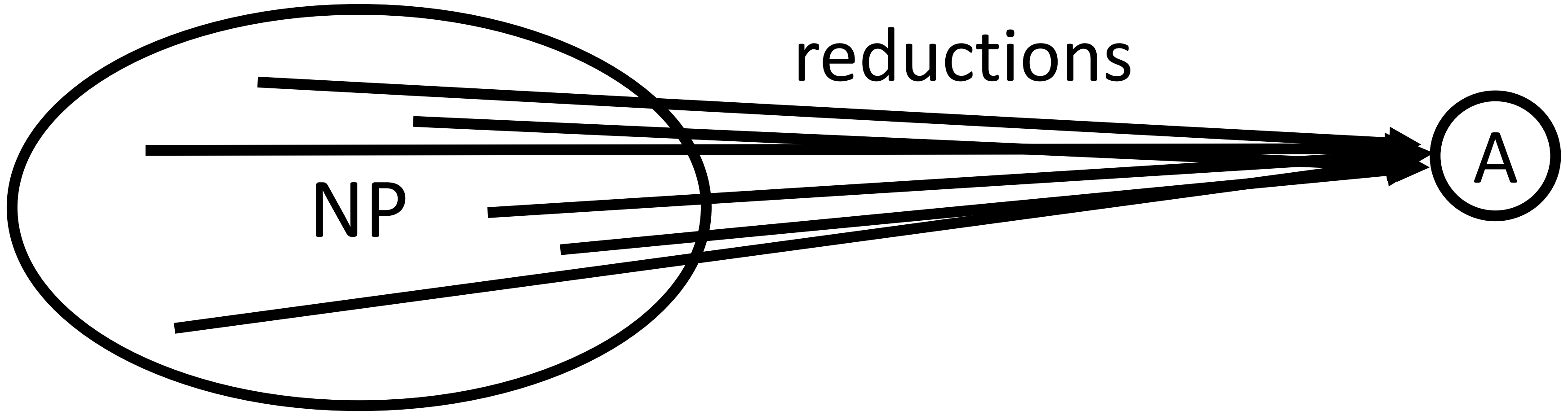
Why use NP-hardness?

Knowing if a problem is NP-hard sets expectations.

It will help others (your boss) to accept slow or inexact algorithms.

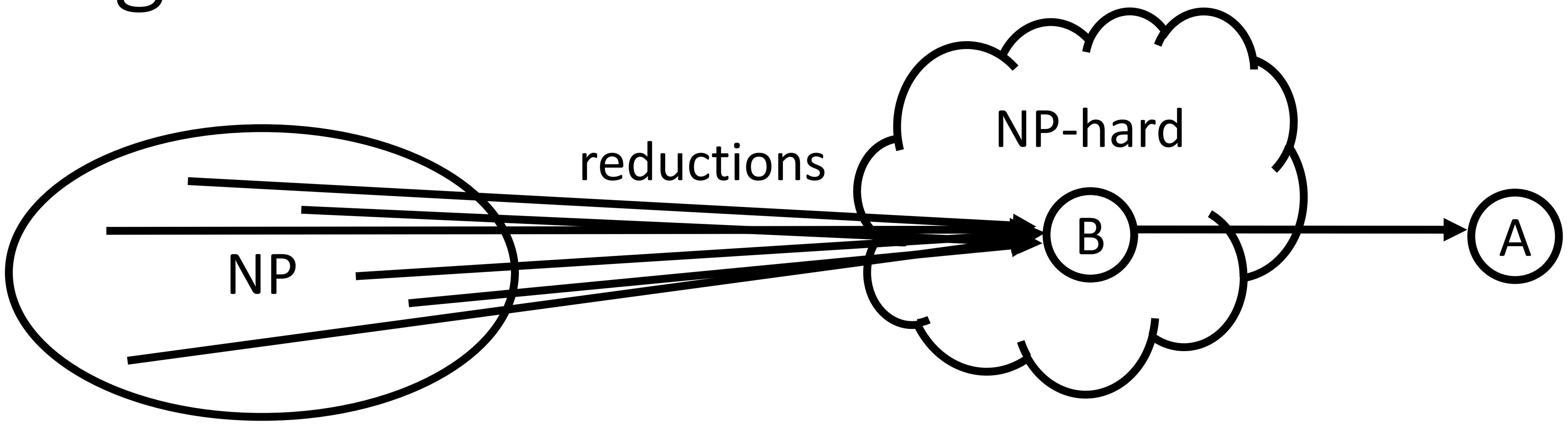
Proving NP-hardness

How do we prove a problem is NP-hard?



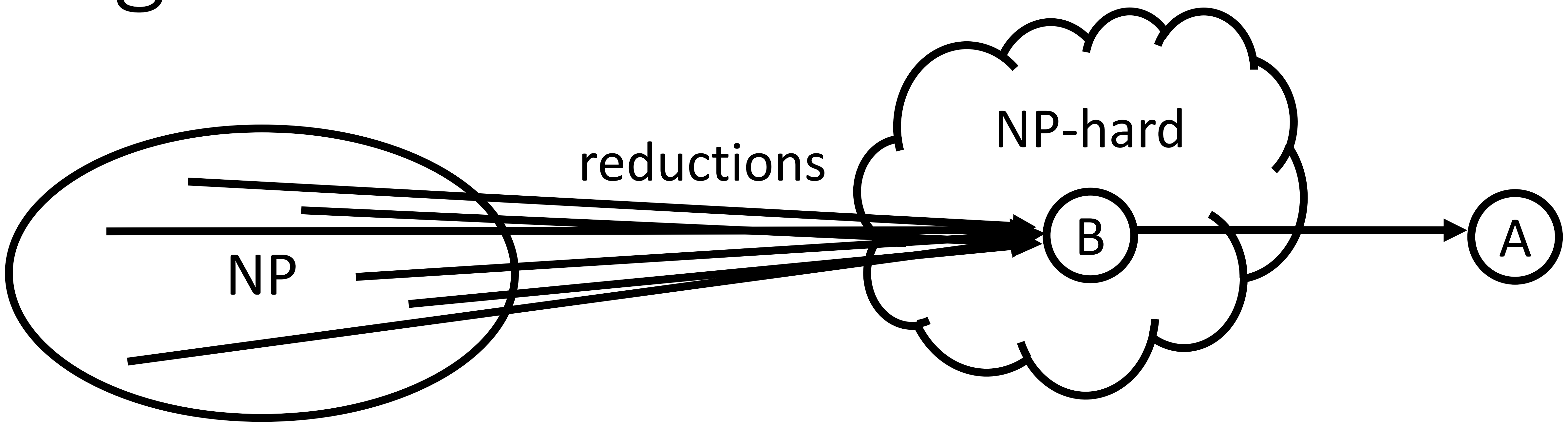
It is a lot of work to prove that all **problems in NP** reduce to A.

Proving NP-hardness



It is sufficient to prove **one NP-hard problem** B reduces to A.

Proving NP-hardness



In practice we just need to one NP-hard problem and reduce to our problem A.

It is sufficient to prove **one NP-hard problem** B reduces to A.

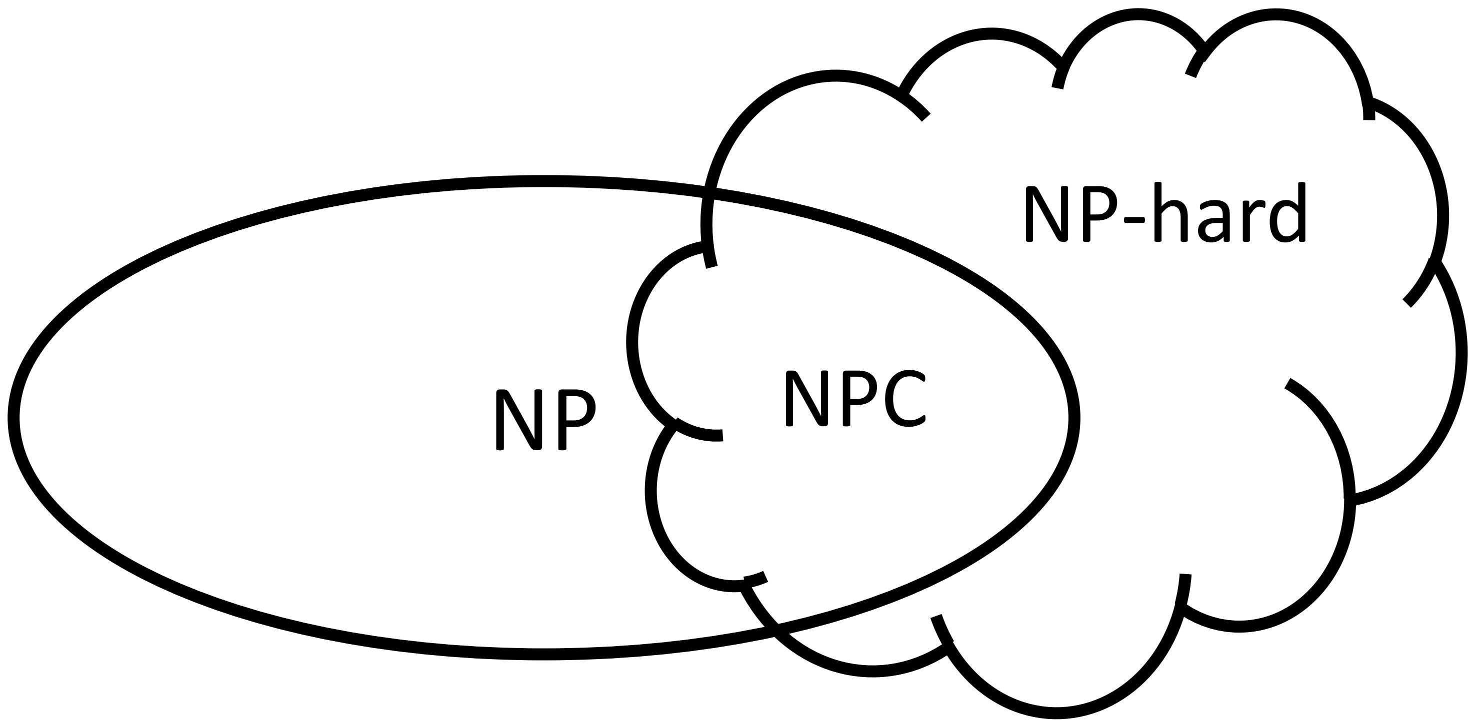
Polynomial time reductions are transitive:

B is NP-hard so all problems in NP reduce to B.
IF B reduces to A **THEN** all problems in NP reduce to A.

Outline

- NP-hardness
  - Definition
  - Why care about NP-hardness?
  - Proving NP-hardness
- NP-completeness
  - Definition
  - Proving NP-completeness
- Expressivity and Hardness

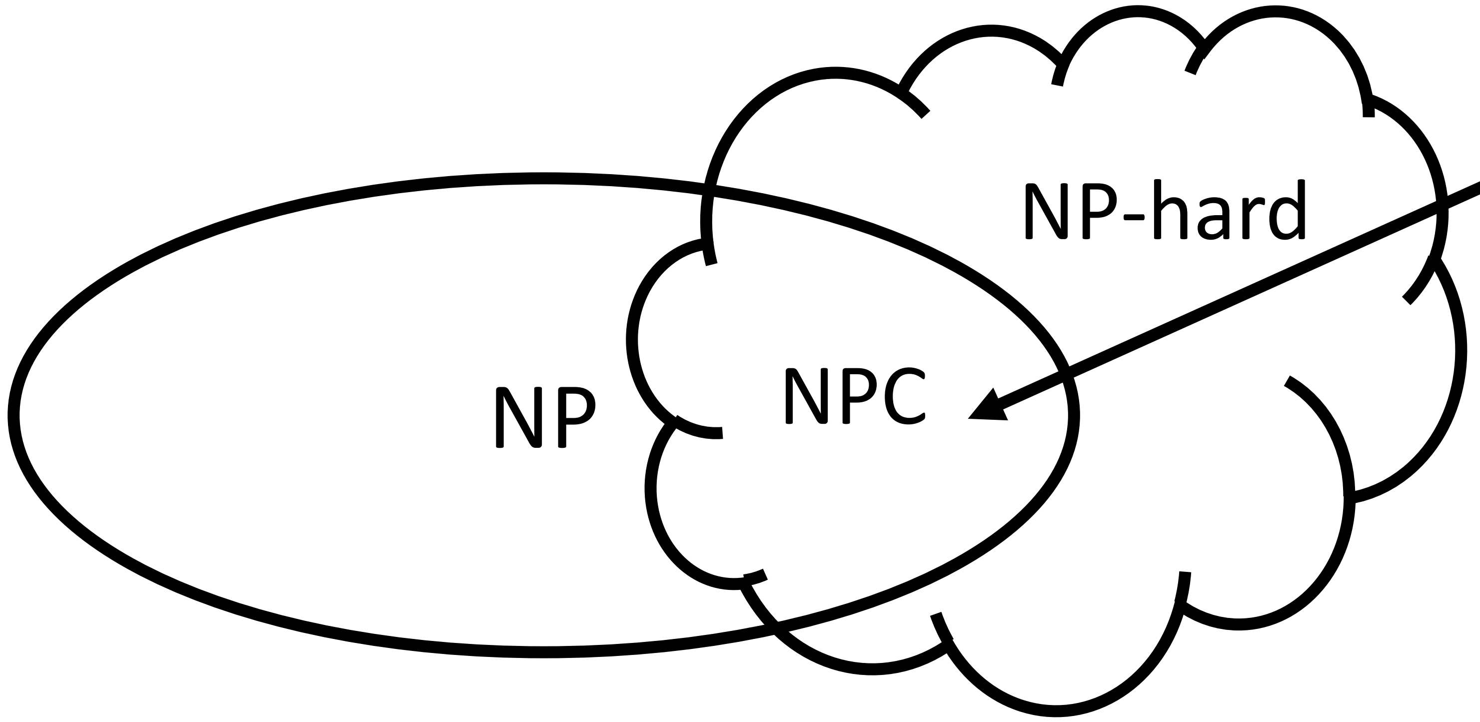
NP-completeness



NP-hard problems are at least as difficult than all problems in NP

NP-complete problems are problems in NP that are also NP-hard.

NP-complete (NPC) problems



NP-complete problems are the hardest problems in NP.

NP-hard problems are at least as difficult than all problems in NP

NP-complete problems are problems in NP that are also NP-hard.

Proving NP-completeness

See the tutorial exercise

Prove that the problem is NP-hard.
Prove that the problem is in NP.

See 1st lecture.

Outline

- NP-hardness
  - Definition
  - Why care about NP-hardness?
  - Proving NP-hardness
- NP-completeness
  - Definition
  - Proving NP-completeness
- Expressivity and Hardness

Hardness is expressivity

Fact: 3SAT is NP-hard (proven in Jeffs' book)

Claim: Vertex Cover is NP-hard (proven in the tutorial exercise)

Claim: Integer linear programming is NP-hard
No formal proof – but we saw how to write VC as an IP.

This means we (probably) cannot solve IPs efficiently.

Hardness is expressivity

More expressive problems are harder.

Maximize x $\quad \boldsymbol{f(x)}$

Subject to

$$\boldsymbol{g_1(x) = b_1}$$
$$\boldsymbol{g_2(x) = b_2}$$
$$\boldsymbol{\ldots}$$
$$\boldsymbol{g_m(x) = b_m}$$

If $\boldsymbol{f(x)}$ and $\boldsymbol{g_j(x)}$ are all **linear** the problem is NP-hard.

But we can solve it in **exponential time**.

Hardness is expressivity

More expressive problems are harder.

Maximize
x

$$\boldsymbol{f(x)}$$

Subject to

$$\begin{aligned} &\boldsymbol{g_1(x) = b_1} \\ &\boldsymbol{g_2(x) = b_2} \\ &\dots \\ &\boldsymbol{g_m(x) = b_m} \end{aligned}$$

If $\boldsymbol{f(x)}$ and $\boldsymbol{g_j(x)}$ are all **linear** the problem is NP-hard.

But we can solve it in **exponential time**.

Point: The expressivity of IPs comes at a cost.

The tools from last lecture will not solve all our problems.

Hardness is expressivity

What if we make the constraints a little bit more expressive?
Suppose $\boldsymbol{g_j}(\boldsymbol{x})$ is (any) **polynomial**. For instance:

$$\boldsymbol{g_{j(x)}} = a_1 \cdot x_1^2 + a_2 \cdot x_2^3 + a_3 \cdot x_3^5$$

Maximize x

$$\boldsymbol{f(x)}$$

Subject to

$$\begin{aligned} \boldsymbol{g_1(x) = b_1} \\ \boldsymbol{g_2(x) = b_2} \\ \dots \\ \boldsymbol{g_m(x) = b_m} \end{aligned}$$

Hardness is expressivity

Maximize
x

$\boldsymbol{f(x)}$

Subject to

$\boldsymbol{g_1(x) = b_1}$
$\boldsymbol{g_2(x) = b_2}$
…
$\boldsymbol{g_m(x) = b_m}$

What if we make the constraints a little bit more expressive?
Suppose $\boldsymbol{g_j(x)}$ is (any) **polynomial.** For instance:

$$\boldsymbol{g_{j(x)}} = a_1 \cdot x_1^2 \ + a_2 \cdot x_2^3 \ + a_3 \cdot \ x_3^5$$

Hilbert's tenth problem:
Consider a polynomial equation with integer coefficients and finite number of variables.
Does the equation have a solution?

Hardness is expressivity

Maximize x

$$\boldsymbol{f}(\boldsymbol{x})$$

Subject to

$$\begin{aligned} \boldsymbol{g_1}(\boldsymbol{x}) &= \boldsymbol{b_1} \\ \boldsymbol{g_2}(\boldsymbol{x}) &= \boldsymbol{b_2} \\ &\dots \\ \boldsymbol{g_m}(\boldsymbol{x}) &= \boldsymbol{b_m} \end{aligned}$$

What if we make the constraints a little bit more expressive?
Suppose $\boldsymbol{g_j}(\boldsymbol{x})$ is (any) **polynomial.** For instance:

$$\boldsymbol{g_{j(x)}} = a_1 \cdot x_1^2 + a_2 \cdot x_2^3 + a_3 \cdot x_3^5$$

Hilbert's tenth problem:
Consider a polynomial equation with integer coefficients and finite number of variables.
Does the equation have a solution?

MRDP theorem:
There is no algorithm that can answer this question

Hardness is expressivity

What if we make the constraints a little bit more expressive?
Suppose $g_i(x)$ is (any) **polynomial**. For instance:

$a_3 \cdot x_3^5$

More expressivity leads to more difficulty!

Maximize x $\quad f(x)$

Subject to

$$g_1(x) = b_1$$
$$g_2(x) = b_2$$
$$\ldots$$
$$g_m(x) = b_m$$

Hilbert's tenth problem:
Consider a polynomial equation with integer coefficients and finite number of variables.
Does the equation have a solution?

MRDP theorem:
There is no algorithm that can answer this question

Hardness is expressivity

Maximize x $f(x)$

Subject to

$g_1(x) = b_1$
$g_2(x) = b_2$
…
$g_m(x) = b_m$

What if we make the constraints a little bit more expressive?
Suppose $g_i(x)$ is (any) polynomial. For instance:

$a_3 \cdot x_3^5$

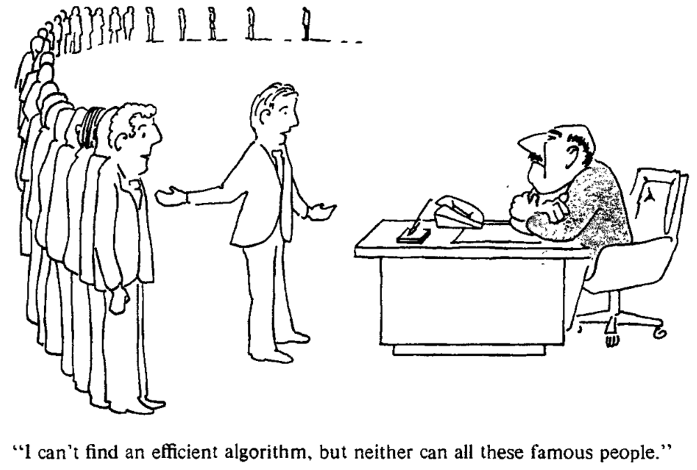
More expressivity leads to more difficulty!

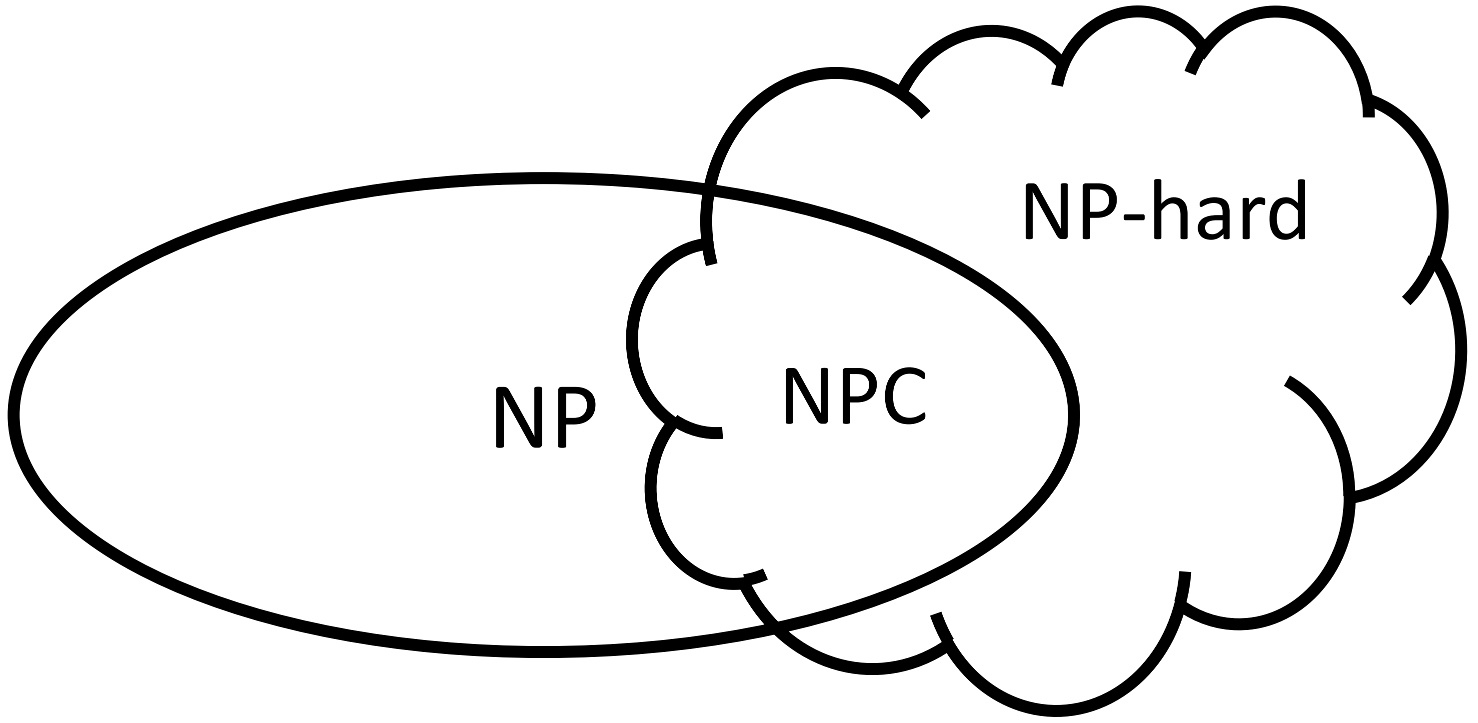
Hilbert's tenth problem:

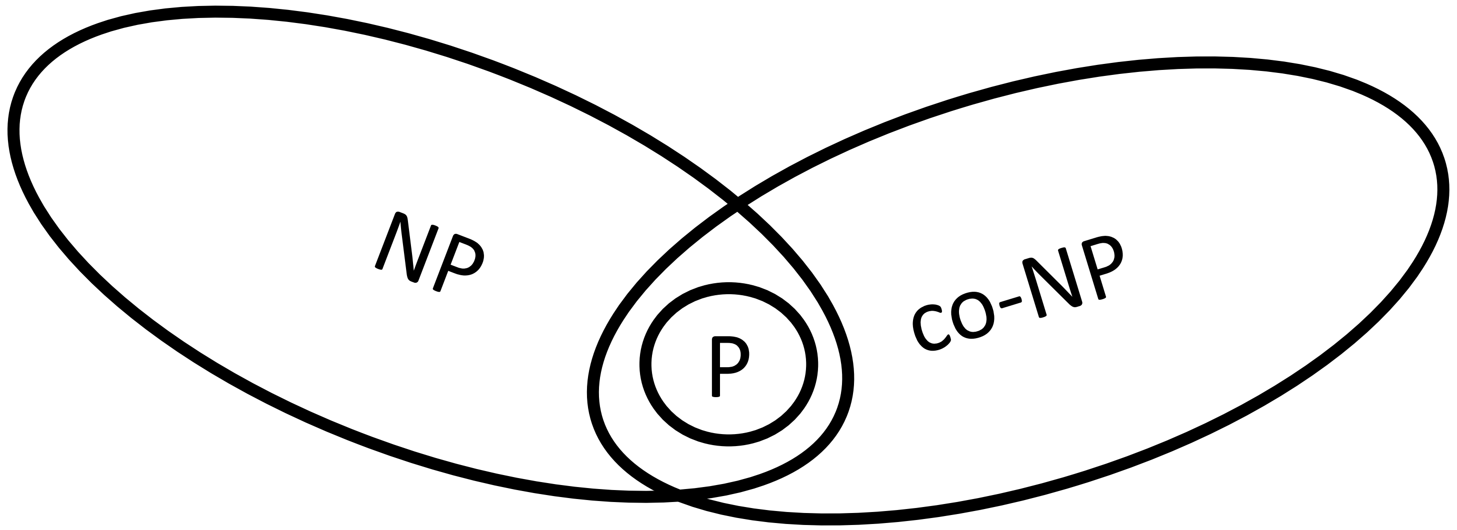
Point:
Keep an eye out for expressivity!
It can be the death of efficient algorithms, or algorithms altogether.

MRDP theorem:
There is no algorithm that can answer this question

Wrap-up







Solvable at all?