RECAP

- Encodings: $\pi_A \overset{J \to e(J)}{\Longrightarrow} \pi_C : \{0,1\}^* \to \{0,1\}$
  - reasonable / concise encodings
- Formal languages

$$\pi_C \Longrightarrow L_{\pi_C} = \{x \in \{0,1\}^* : \pi_C(x) = 1\}$$

solution ⟷ decision

- Poly-time solution/decision: **P**
- Verification: algorithm $V(x,y)$

$$L_V = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* : V(x,y) = 1\}$$

- Poly-time verification: **NP**
  - poly-size certificate $[|y| = O(|x|^{k_1})]$
  - poly-time algorithm $[T_V(x,y) = O((|x|+|y|)^{k_2})]$

ogni bit di certificato simile a guess nondet. -> numero di guess deve avere upper bound polinomiale -> connessione con nondet. TM

Theorem: $P \subseteq NP$ $L \in P \Rightarrow L \in NP$

Proof: Let $L \in P$. We'll prove $L \in NP$ by constructing a poly-time verifier:

da hp, abbiamo alg. di decisione polinomiale

Recall that $L \in P \Rightarrow \exists A_L$ which decides L in time $T_{A_L}(|x|) = O(|x|^k)$

Verifier $V_L(x,y)$ uses A_L as a subroutine:

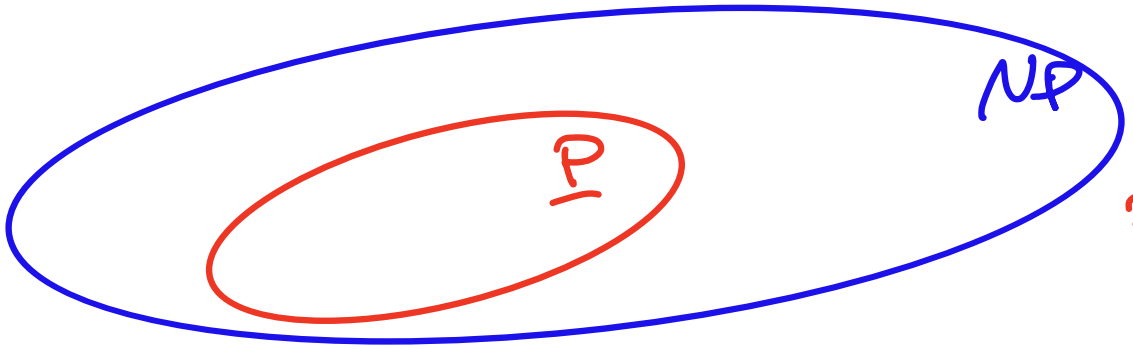
$V_L(x, y)$
return $A_L(x)$

The verifier disregards y and calls $A_L(x)$. Clearly we have

1. $L_V = L$ $\quad (V_L(x,y) = 1 \Leftrightarrow x \in L)$

2. $y = \varepsilon$ is a certificate $\forall x \in L$
$\quad (|\varepsilon| = O(|x|^0) : k_1 = 0)$

3. $T_V(|x|, |y|) = T_{A_L}(|x|) = O(|x|^k)$
$= O((|x| + |y|)^k) \quad (k_2 = k)$



$P \subseteq NP$?

What can we say about $NP - P$?
Could be empty! $(P = NP)$
Could be nonempty! $(P \neq NP)$

The question $P = NP$? is still open (1 million-dollar question)

Reasonable guess: $P \neq NP$

Evidence:

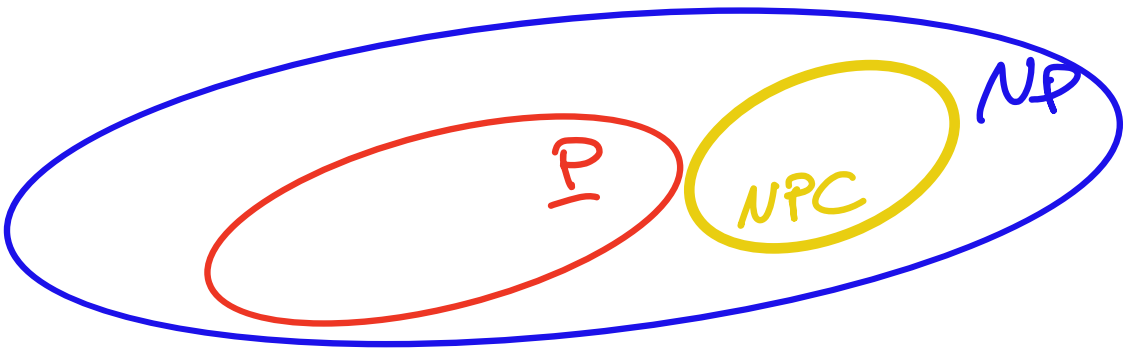
1. Verifying seems intrinsically easier than solving

relazione che definisce problemi più difficili di altri

2. We'll define a partial order on NP (based on solution difficulty). For the most difficult problems in NP no poly-time algorithms are known.

3. The most difficult problems in NP form an equivalence class: NPC (NP-complete problems)

se P!=NP, non esiste alg polinomiale per problemi NP-completi
si può anche dimostrare che esistono problemi in NP \ (P U NPC) (e.g. embedding di grafi in grafi) -> structural complexity theory



If $P \neq NP \Rightarrow NPC \in NP - P$ (LIKELY)

If $P = NP \Rightarrow NPC \subset P$ (UNLIKELY)

We will see that:

- NPC contains many decision problems from different fields of huge practical interest.

- Problems in NPC are "equivalent" in the following sense:

> A poly-time solution to any one problem in NPC immediately yields a poly-time solution to all problems in NPC!

REDUCIBILITY BETWEEN PROBLEMS

Intuitively, Π_1 can be reduced to Π_2 if any instance i_1 of Π_1 can be "easily" transformed into an instance i_2 of Π_2 so that a solution to i_2 provides a solution to i_1

For decision problems:

$i_1 \Longrightarrow f(i_1) = i_2 \qquad \Pi_1(i_1) = \Pi_2(i_2)$

Map f must transform positive instances into positive instances and negative instances into negative instances

SIMPLE EXAMPLE

Π_1: $\begin{cases} I: \langle a, b, x \rangle, \; a, b, x \in \mathbb{C} \\ Q: ax + b = 0 \; ? \end{cases}$

(decision version of first degree equations)

Π_2: $\begin{cases} I: \langle c, e, f, x \rangle, \; c, e, f, x \in \mathbb{C} \\ Q: cx^2 + ex + f = 0 \; ? \end{cases}$

(decision version of second degree equations)

We can reduce Π_1 to Π_2 as follows

$i_1 = \langle a, b, x \rangle \xrightarrow{f} \langle \overset{c}{0}, \overset{e}{a}, \overset{f}{b}, x \rangle = i_2$

Clearly, $\Pi_1(i_1) = \Pi_2(f(i_1)) = \Pi_2(i_2)$

$0 \cdot x^2 + ax + b = ax + b = 0$?

NOTE: An "efficient" reduction from Π_1 to Π_2 implies that Π_1 is "less difficult" than Π_2

(If I can solve Π_2 efficiently, then I can also solve Π_1 efficiently)

"Efficient" reducibility creates an order between problems.

DEF: A function $f:\{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable (ptc) if $\exists$ algorithm $A_f : \{0,1\}^* \to \{0,1\}^*$:

$\forall x \in \{0,1\}^*$: $A_f(x) = f(x)$ and $T_{A_f}(|x|) = O(|x|^k)$ for some constant $k \geq 0$

DEF: L_1 is poly-time reducible to L_2 (notation: $L_1 \leq_p L_2$) if:

$\exists f:\{0,1\}^* \to \{0,1\}^*$ ptc:

$$x \in L_1 \iff f(x) \in L_2$$

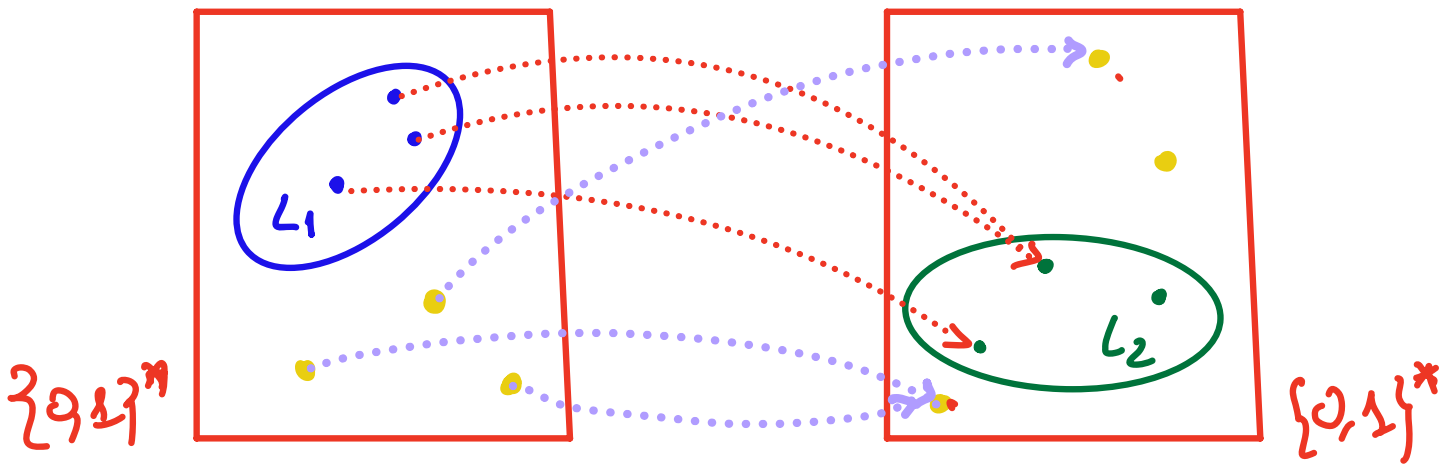
(w.r.t. problems: $\Pi_{L_1}(x) = \Pi_{L_2}(f(x))$)

$\iff$ means that:

WE NEED TO PROVE TWO IMPLICATIONS

- strings in L_1 are mapped to strings in L_2 $\Longrightarrow$
- strings not in L_1 are mapped to strings not in L_2 $\Longleftarrow \equiv \not\Longrightarrow$ COUNTERPOSITIVE

Remark f need not be neither injective or surjective



Poly-time reducibility captures the concept of "lesser difficulty" of L_1 with respect to L_2 in the following sense:

$$L_1 <_P L_2$$

THEOREM: Given $L_1, L_2 \subseteq \{0,1\}^*$:

$$(L_1 <_P L_2) \wedge (L_2 \in P) \Rightarrow L_1 \in P$$

(if L_2 can be decided in poly-time, so can L_1 !)

PROOF: We have two hypotheses:

HP1: $L_1 \leq_P L_2$: $\exists f$ ptc : $x \in L_1 \iff f(x) \in L_2$
Let $T_{A_f}(|x|) = O(|x|^{k_1})$ $k_1 \geq 0$

HP2: $L_2 \in P$: $\exists A_{L_2}$ which decides L_2 in poly time. Let $T_{A_{L_2}}(|x|) = O(|x|^{k_2})$

I can write the following algorithm

```
A_{L_1}(x)
  y ← A_f(x)
  return A_{L_2}(y)
```

Running time:
$$T_{A_{L_1}}(|x|) = O(|x|^{k_1} + |y|^{k_2})$$

How large can y be? A_f may use all of its steps to create bits of y, thus
$$|y| = O(|x|^{k_1})$$

Hence
$$T_{A_{L_1}} = O(|x|^{k_1} + |x|^{k_1 k_2}) = O(|x|^{k_3})$$

with $k_3 = \max\{k_1, k_1 \cdot k_2\}$ STILL POLYNOMIAL!

It remains to show that $L_1 \subseteq L_{A_{L_1}}$:

$x \in L_1 \Rightarrow y = f(x) \in L_2 \Rightarrow A_{L_1}(x) = A_{L_2}(y) = 1$

$\Rightarrow x \in L_{A_{L_1}}$ $\quad L_1 \subseteq L_{A_{L_1}}$

$x \in L_{A_{L_1}} \Rightarrow A_{L_1}(x) = A_{L_2}(y) = A_{L_2}(f(x)) = 1$

$\Rightarrow f(x) \in L_2 \Rightarrow x \in L_1$ $\quad L_{A_{L_1}} \subseteq L_1$

We proved $L_1 \subseteq L_{A_{L_1}} \Rightarrow L_1 = L_{A_{L_1}}$

A_{L_1} is a poly-time algorithm deciding L_1, therefore $L_1 \in P$

REMARK The proof uses both implications $x \in L_1 \Rightarrow f(x) \in L_2$ and $x \in L_1 \Leftarrow f(x) \in L_2$

Poly-time reducibility $\leq_P$ is a relation between languages.

Homework: Prove that:

$L \leq_P L$ reflexivity

$(L_1 \leq_P L_2) \wedge (L_2 \leq_P L_3) \Rightarrow (L_1 \leq_P L_3)$ transitivity

DEF: $L \subseteq \{0,1\}^*$ is NP-hard if

$$\forall L' \in NP : L' \leq_p L$$

$$NPH = \{L \in \{0,1\}^* : L \text{ NP-hard}\}$$

An NP-hard language/problem L is "more difficult" than any problem in NP, in the sense that a poly-time algorithm for L can be transformed into a poly-time algorithm for any problem in NP

REMARK An NP-hard language L need not be in NP itself!

DEF: $L \subseteq \{0,1\}^*$ is NP-Complete if

1. $L \in NP$
2. $L \in NPH$

$$NPC = \{L \subseteq \{0,1\}^* : L \text{ NP-complete}\}$$

REMARK: Clearly, $NPC \subseteq NP$

NP-complete problems are the "hardest" problems in NP! In fact, they are all somewhat equivalent, since

$$\forall L_1, L_2 \in NPC : (L_1 <_P L_2) \wedge (L_2 <_P L_1)$$

$L_1 \in NP$, $L_2 \in NPC$, $L_2 \in NP$, $L_1 \in NPC$

COROLLARY:

If $\exists L \in (NPC \cap P) \Rightarrow P = NP$

PROOF

$\forall L \in NP :$

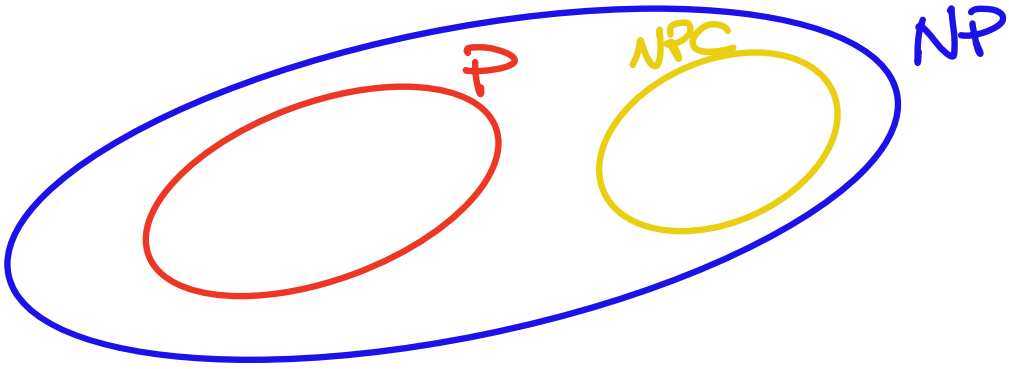
$(L <_P L) \wedge (L \in P) \Rightarrow L \in P$

Thus, $NP \subseteq P$, but also $P \subseteq NP$

$\Rightarrow P = NP$

No poly-time algorithms are known for NP-complete problems. Evidence points towards $P \cap NPC = \emptyset$.

If this is the case:



We will often prove results under the hypothesis that $P \neq NP$ (or, equivalently, $P \cap NPC = \emptyset$)

NP-COMPLETENESS PROOFS

We have defined NPC but how do we place problems in NPC?

Given a candidate language L:

1. Prove $L \in NP$
2. Prove that $\forall L \in NP : L \leq_P L$

1. is usually easy (come up with a verification algorithm)
   (e.g. HAMILTON)

2. more difficult because of $\forall$! Infinite reductions!

Fortunately, $\leq_P$ is transitive!

If I know that a given language $\hat{L} \in NPC$, it suffices to prove that

$$\hat{L} \leq_P \bar{L}$$

($\hat{L}$: known to be in NPC; $\bar{L}$: candidate to be in NPC)

If $\hat{L} \leq_P \bar{L}$, since $\forall L \in NP,\ L \leq_P \hat{L}$

we have: $\forall L \in NP: (L \leq_P \hat{L}) \wedge (\hat{L} \leq_P \bar{L})$

$$\Rightarrow \forall L \in NP: L \leq_P \bar{L}$$

A single reduction suffices!

Unfortunately, the transitivity trick cannot be used for the first language/problem to be placed in NPC!

The first problem requires a "universal" reduction from a generic problem in NP!