Complexité CM7

Antonio E. Porreca
aeporreca.org

Diapositives basées sur le livre Complexité algorithmique par Sylvain Perifel

Algorithmes non déterministes

Un exemple : clique

fonction clique(S, A, k)
    C := tableau(k)
    pour $i := 0$ **à** $k - 1$ **faire**
        $C[i]$:= devine(S)
    pour chaque $s \in S$ **faire**
        si C contient s plusieurs fois **alors**
            rejeter
    pour $i := 0$ **à** $k - 1$ **faire**
        pour $j := i + 1$ **à** $k - 1$ **faire**
            si $\{C[i], C[j]\} \notin A$ **alors**
                rejeter
    accepter
fin

Un exemple : clique

fonction clique(S, A, k)
    C := tableau(k)
    pour $i := 0$ **à** $k - 1$ **faire**
        $C[i]$:= devine(S)
    pour chaque $s \in S$ **faire**
        si C contient s plusieurs fois **alors**
            rejeter
    pour $i := 0$ **à** $k - 1$ **faire**
        pour $j := i + 1$ **à** $k - 1$ **faire**
            si $\{C[i], C[j]\} \notin A$ **alors**
                rejeter
    accepter
fin

C est-il un ensemble de k éléments différents ?

Un exemple : clique

fonction clique(S, A, k)
    $C :=$ tableau(k)
    pour $i := 0$ **à** $k-1$ **faire**
        $C[i] :=$ devine(S)
    pour chaque $s \in S$ **faire**
        si C contient s plusieurs fois **alors**
            rejeter
    pour $i := 0$ **à** $k-1$ **faire**
        pour $j := i+1$ **à** $k-1$ **faire**
            si $\{C[i], C[j]\} \notin A$ **alors**
                rejeter
    accepter
fin

C est-il un ensemble de k éléments différents ?

les sommets dans C sont-ils tous reliés par des arêtes ?

Un exemple : clique

```
fonction clique(S, A, k)
    C := tableau(k)
    pour i := 0 à k − 1 faire
        C[i] := devine(S)
    pour chaque s ∈ S faire
        si C contient s plusieurs fois alors
            rejeter
    pour i := 0 à k − 1 faire
        pour j := i + 1 à k − 1 faire
            si {C[i], C[j]} ∉ A alors
                rejeter
    accepter
fin
```

$O(k \log |S|)$ **bits devinés**

C **est-il un ensemble de** k **éléments différents ?**

les sommets dans C **sont-ils tous reliés par des arêtes ?**

Divinations 🔮 et vérifications 🔍, ou le non déterminisme dans le monde réel

Proposition 2-AO (p. 56)

Caractérisation existentielle de $\mathbf{NP}$

Les deux conditions suivantes sont équivalentes :

- Le langage L est reconnu en temps polynomiale par une machine de Turing non déterministe N

- Il existe une machine de Turing déterministe M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

$$x \in L \text{ ssi } \exists y \in \{0,1\}^{p(n)}\ M(x,y) \text{ accepte}$$

Le mot y qui justifient l'appartenance de x à L est appelé preuve ou certificat

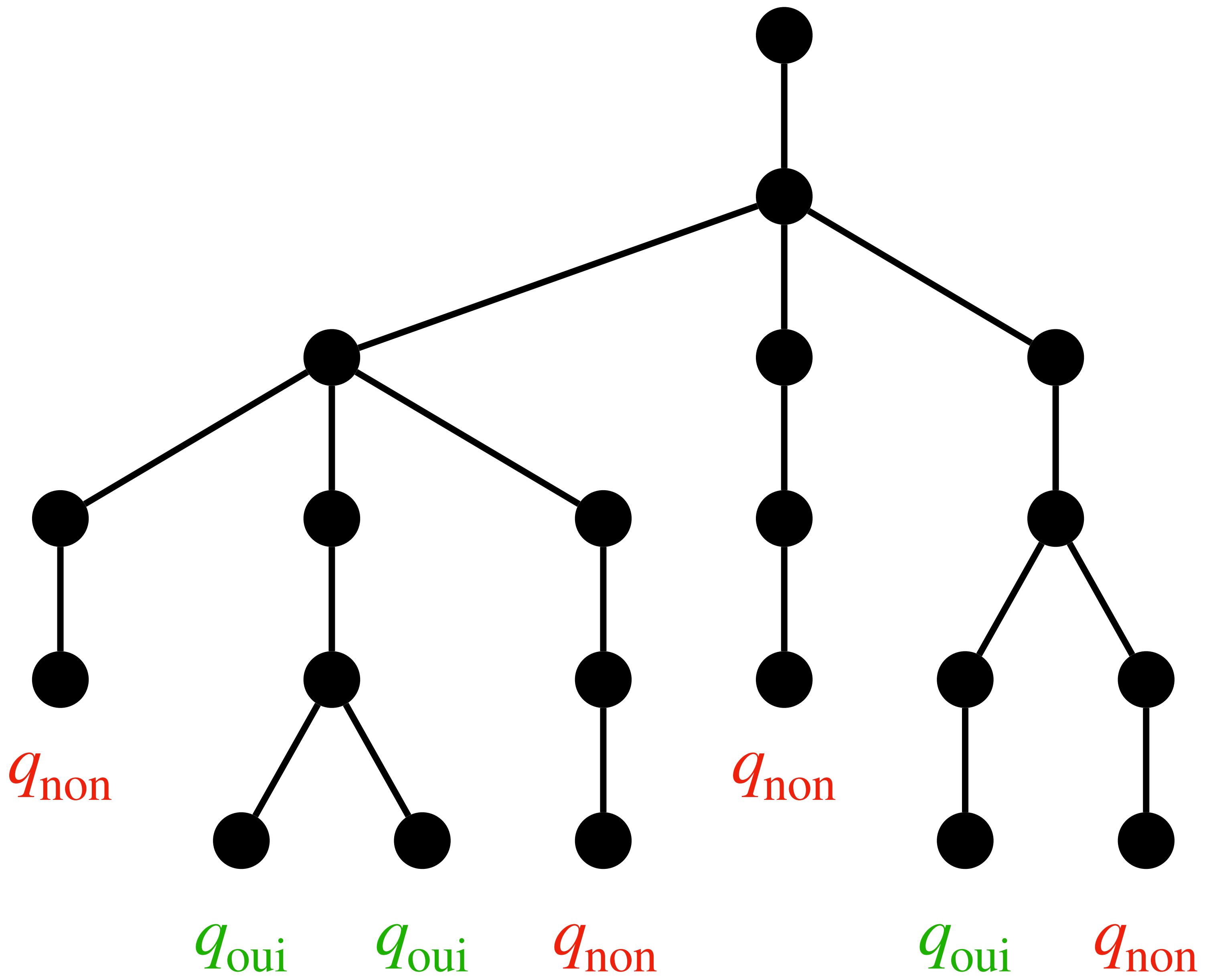
💡 Idée de la démonstration

Le certificat y correspond au chemin acceptant
pour l'entrée x dans la machine non déterministe N
qui reconnait le langage L

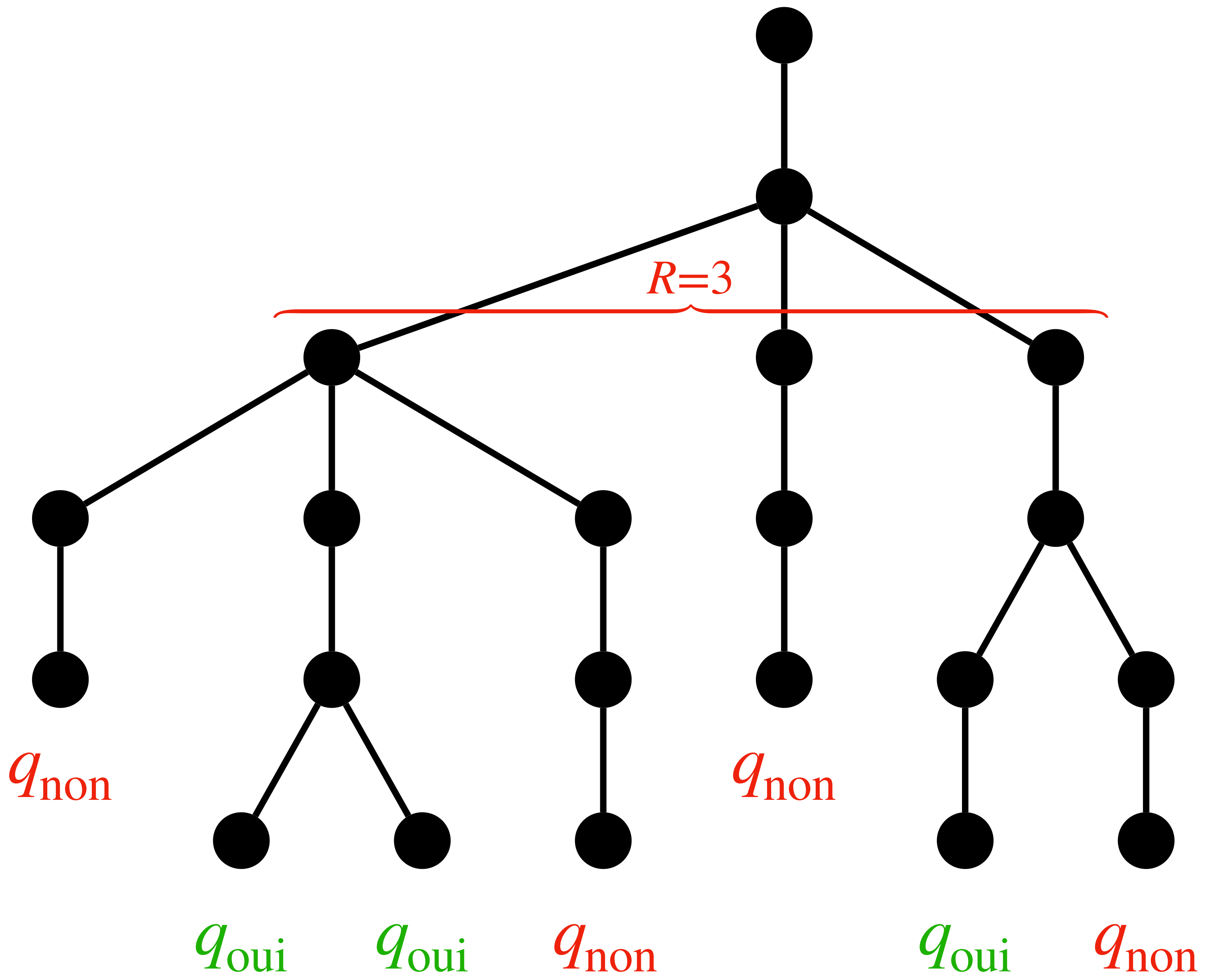
Demonstration

- Supposons que N reconnaît L en temps polynomial $q(n)$
- À chaque étape, N choisit entre un nombre $\leq R$ de transitions possibles (R est constant, il ne dépend pas de l'entrée x)
- Chaque choix peut être décrite avec $\lceil \log R \rceil$ bits

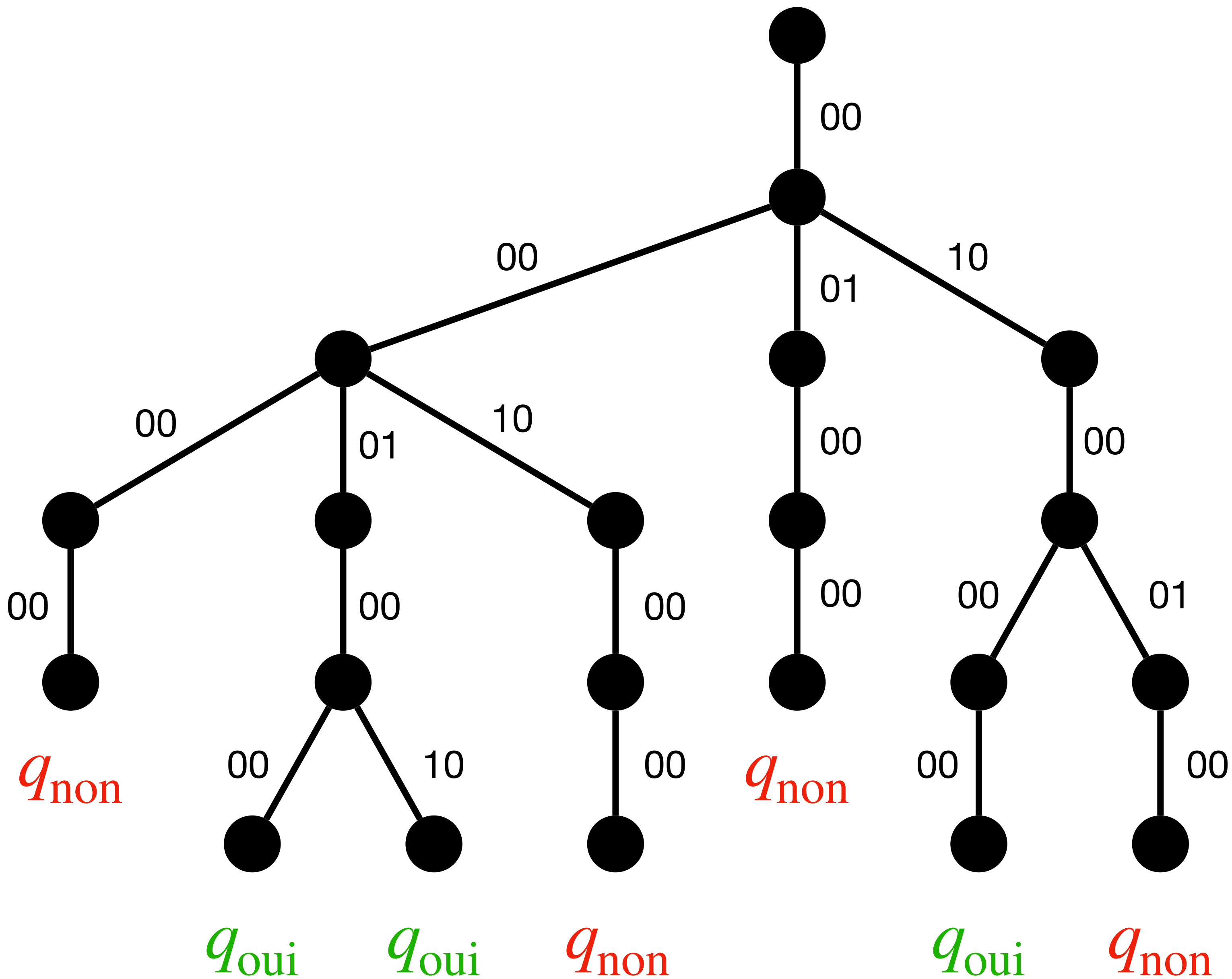
Arbre de calcul de N sur x



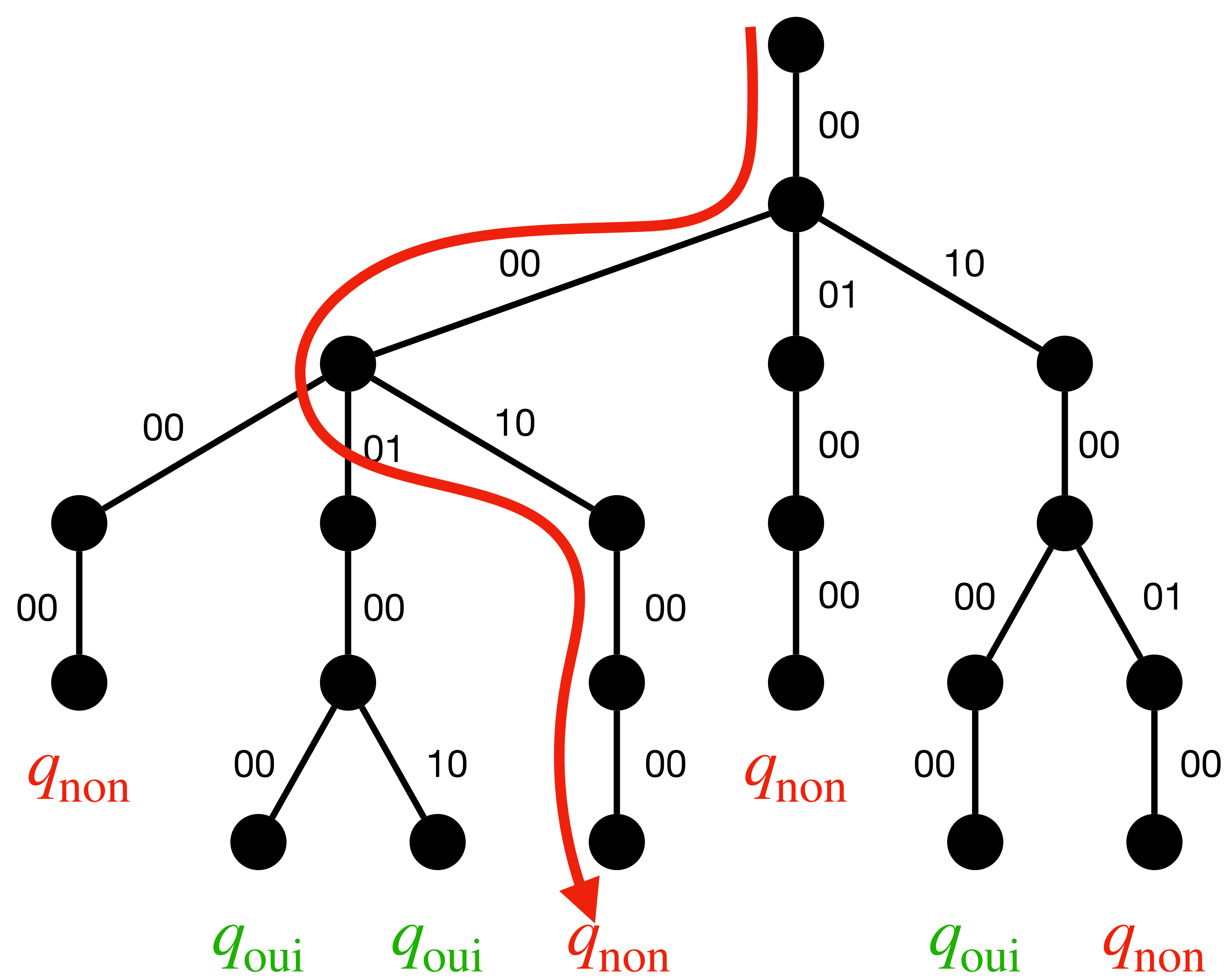
Arbre de calcul de N sur x



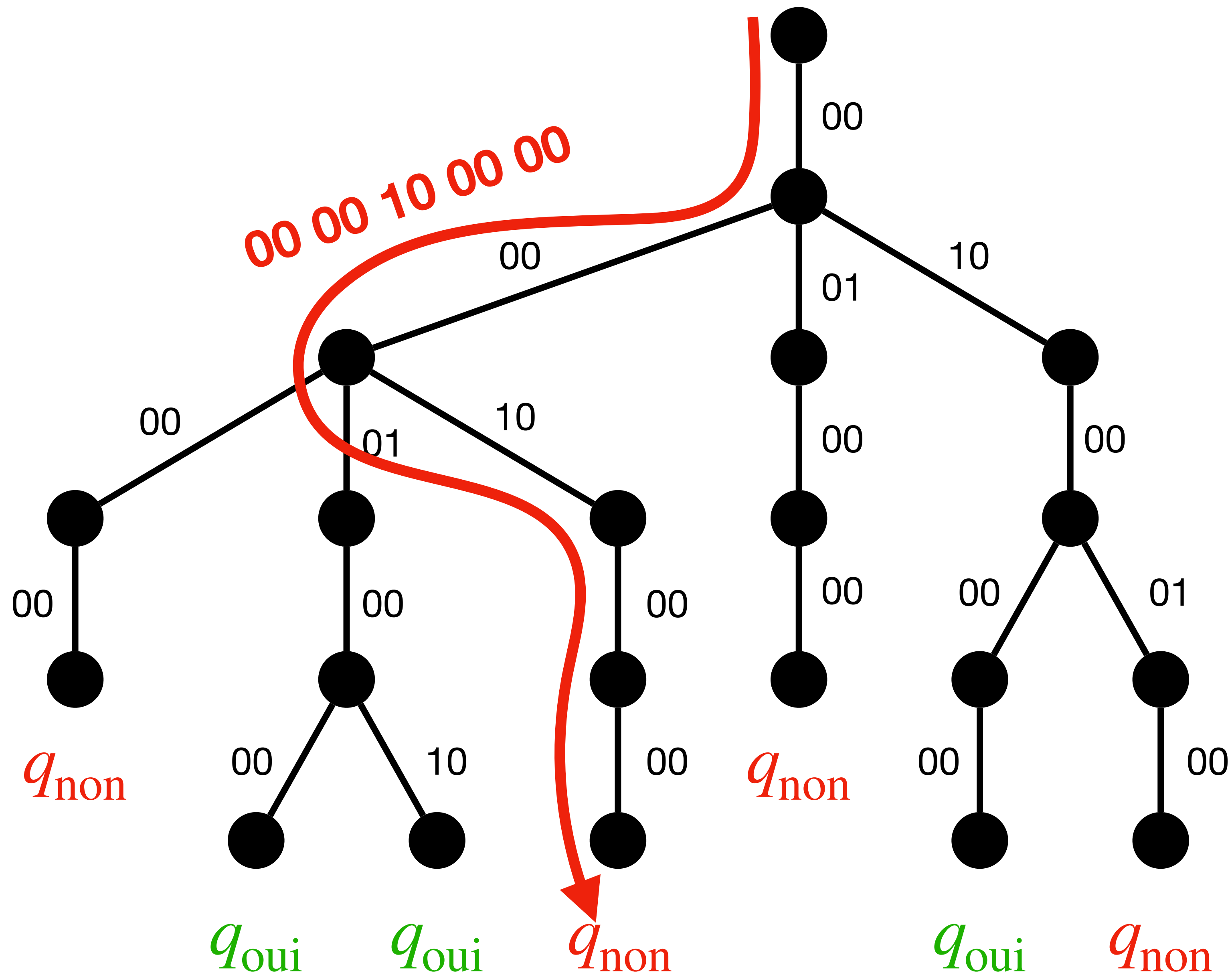
Arbre de calcul de N sur x



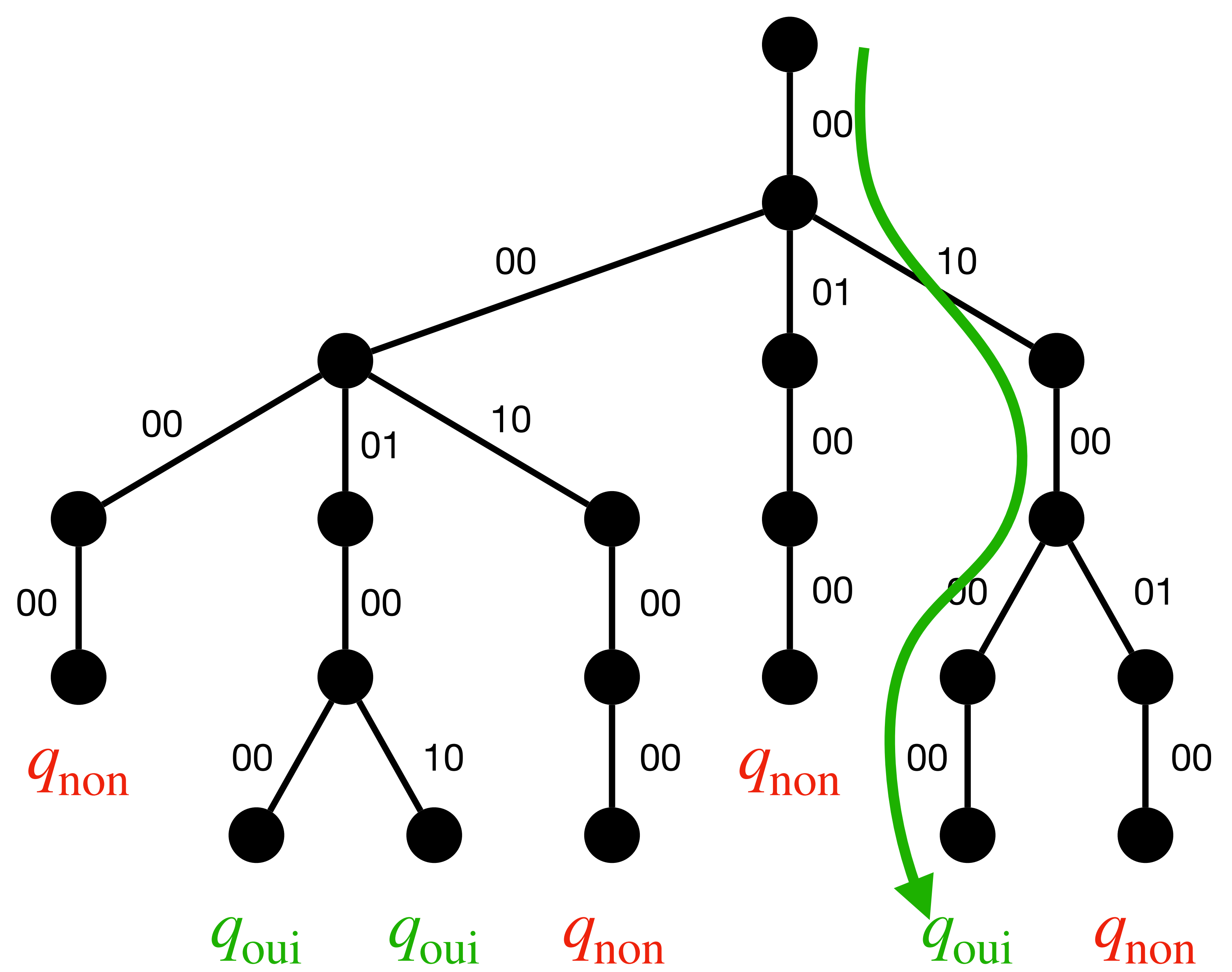
Arbre de calcul de N sur x



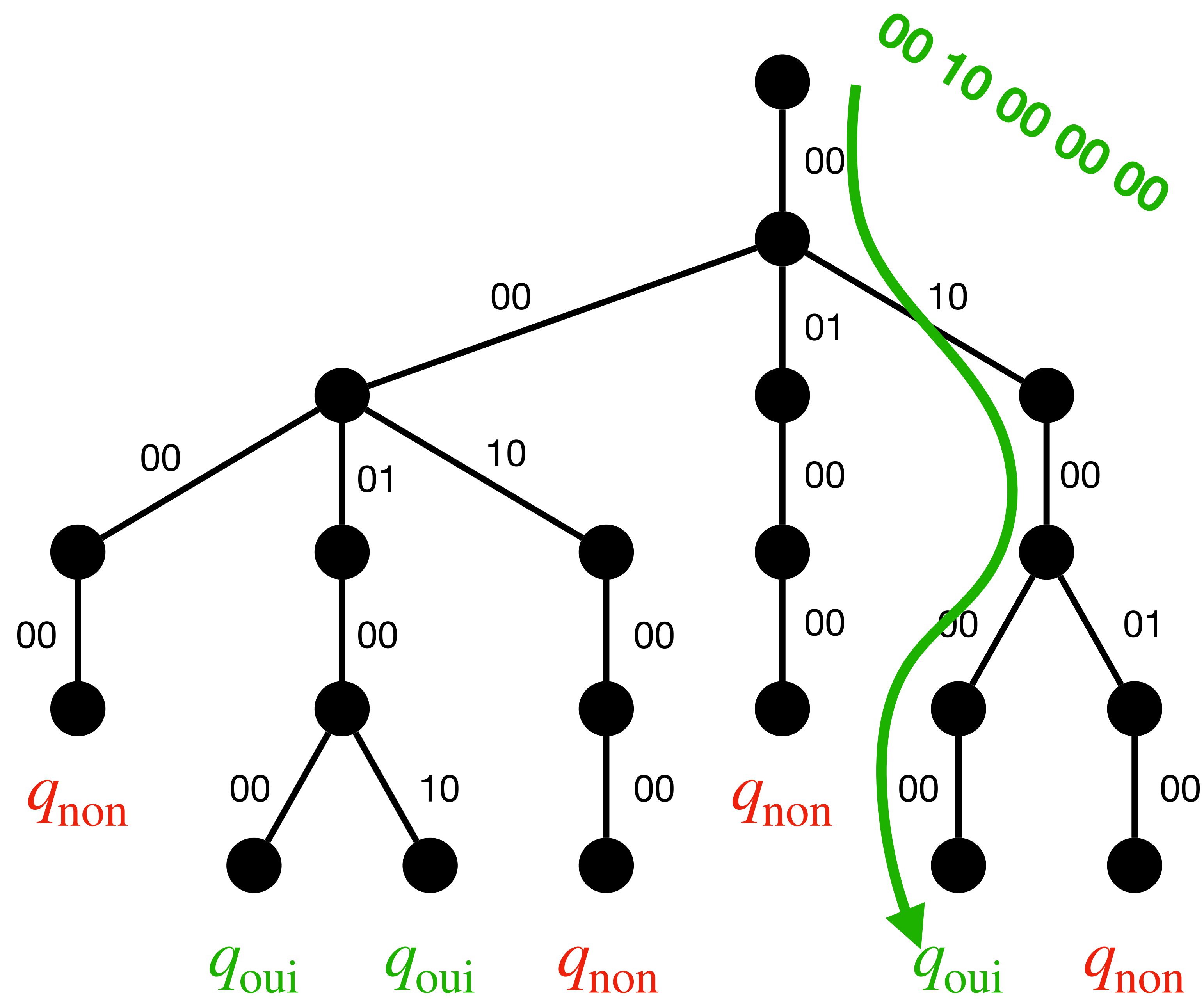
Arbre de calcul de N sur x



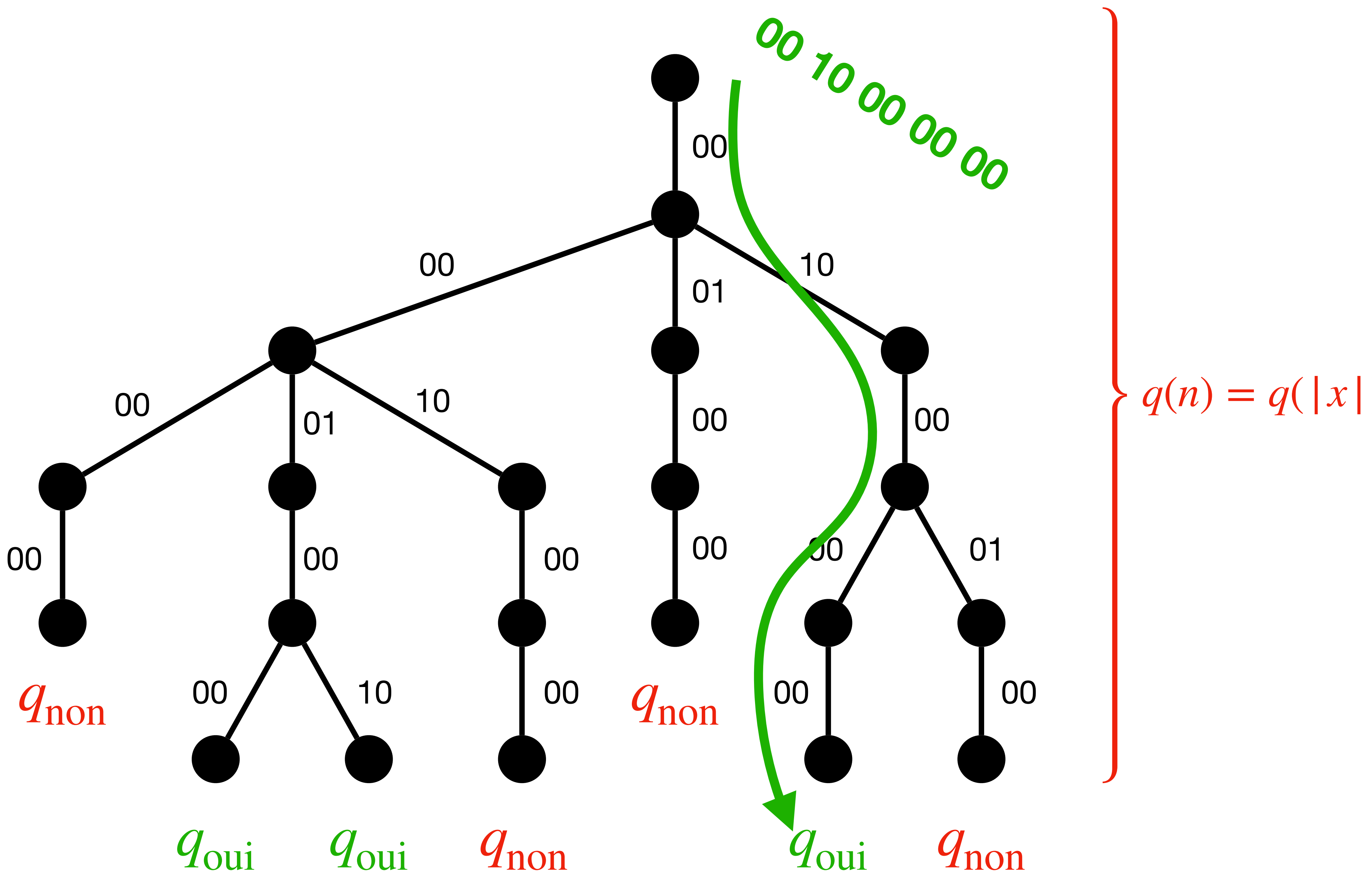
Arbre de calcul de N sur x



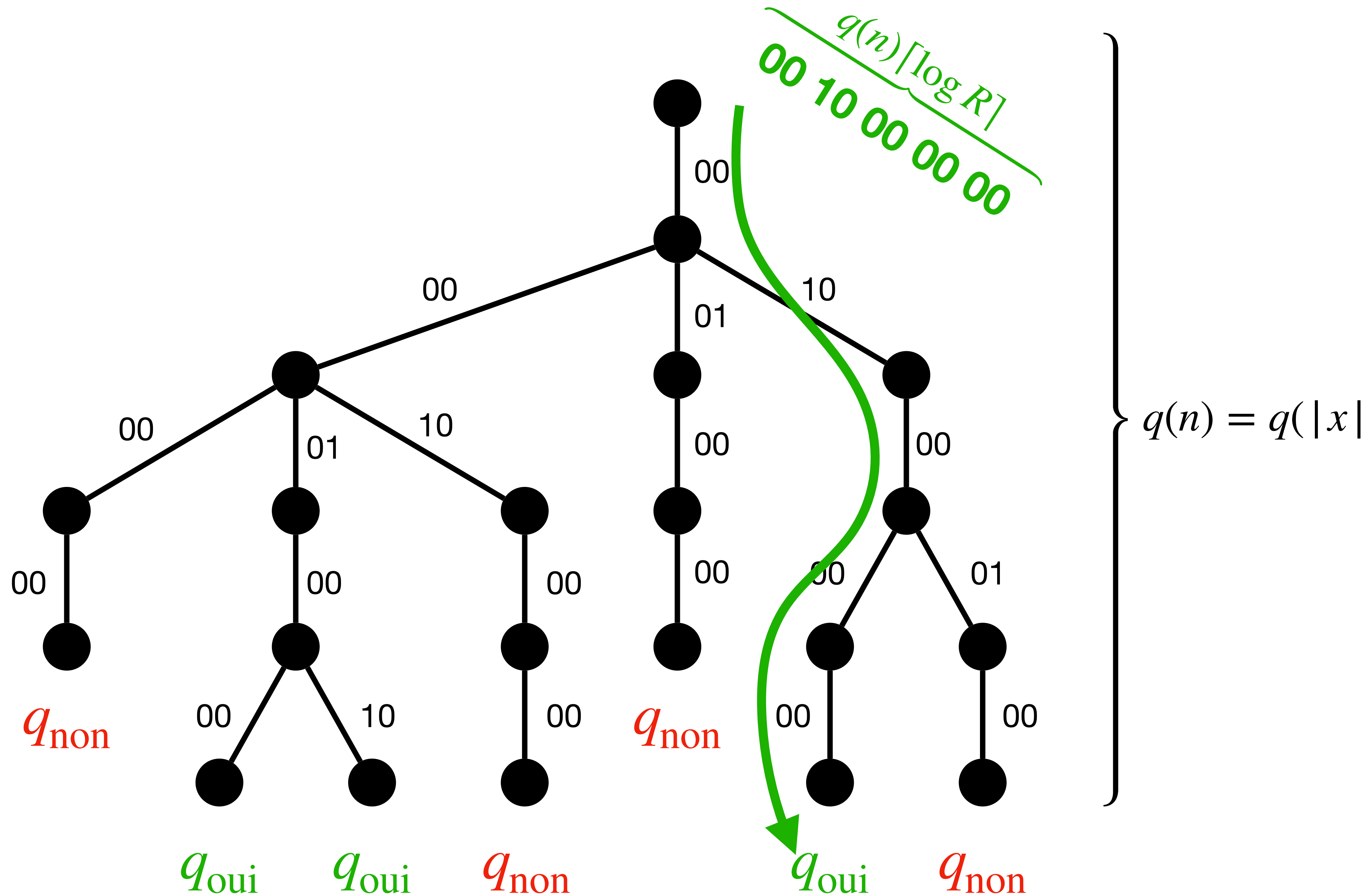
Arbre de calcul de N sur x



Arbre de calcul de N sur x

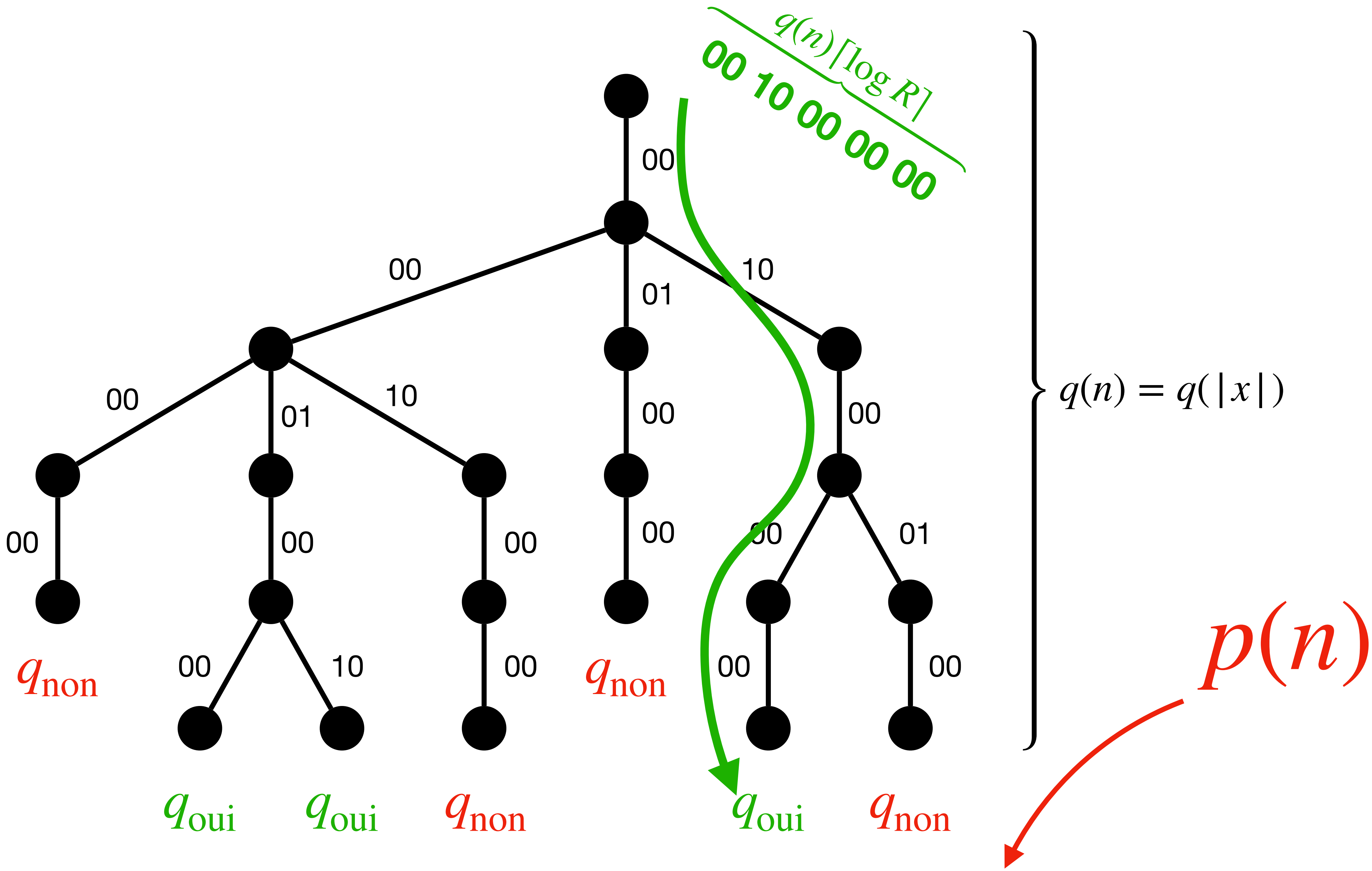


Arbre de calcul de N sur x



Chaque chemin y est décrit par $q(n)\lceil \log R \rceil$ bits

Arbre de calcul de N sur x



Chaque chemin y est décrit par $q(n)\lceil \log R \rceil$ bits

Machine déterministe $M(x, y)$

- Simuler la machine non déterministe N sur l'entrée x
- À chaque étape simulée, choisir la transition indiqué par y
- Comme on connaît le chemin à simuler, on peut faire ça en temps polynomial (déterministe)
- La machine M accepte ssi la machine N a un chemin acceptant

Première implication 👍

- Le langage L est reconnu en temps polynomiale par une machine de Turing non déterministe N

$$\Downarrow$$

- Il existe une machine de Turing déterministe M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

$$x \in L \text{ ssi } \exists y \in \{0,1\}^{p(n)}\ M(x,y) \text{ accepte}$$

Deuxième implication 🤔

- Le langage L est reconnu en temps polynomiale par une machine de Turing non déterministe N

$$\Uparrow$$

- Il existe une machine de Turing déterministe M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

$$x \in L \text{ ssi } \exists y \in \{0,1\}^{p(n)}\ M(x, y) \text{ accepte}$$

Machine non déterministe $N(x)$

- Deviner un certificat $y \in \{0,1\}^{p(n)}$
- Simuler $M(x, y)$ et accepter ssi cette machine accepte
- Si M fonctionne et temps $q(n)$, sa simulation prend du temps

$$p(n) \quad + \quad q(n + p(n))$$

divination

simulation de M sur (x, y)
vu que $|(x, y)| = n + p(n)$

Deuxième implication 👍

- Le langage L est reconnu en temps polynomiale par une machine de Turing non déterministe N

$$\Uparrow$$

- Il existe une machine de Turing déterministe M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

$$x \in L \text{ ssi } \exists y \in \{0,1\}^{p(n)}\ M(x, y) \text{ accepte}$$

Algorithmes non déterministes

```
fonction clique(S, A, k)
    C := tableau(k)
    pour i := 0 à k − 1 faire
        C[i] := devine(S)
    pour chaque s ∈ S faire
        si C contient s plusieurs fois alors
            rejeter
    pour i := 0 à k − 1 faire
        pour j := i + 1 à k − 1 faire
            si {C[i], C[j]} ∉ A alors
                rejeter
    accepter
fin
```

Vérificateurs déterministes

fonction clique(S, A, k, C)
pour chaque $s \in S$ **faire**
si C contient s plusieurs fois **alors**
rejeter
pour $i := 0$ **à** $k - 1$ **faire**
pour $j := i + 1$ **à** $k - 1$ **faire**
si $\{C[i], C[j]\} \notin A$ **alors**
rejeter
accepter
fin

$\mathbf{P}$ vs $\mathbf{NP}$

- $\mathbf{P}$ est la classe des problèmes faciles à résoudre
- $\mathbf{NP}$ est la classe des problèmes avec des solutions faciles à vérifier
- Facile à résoudre implique facile à vérifier, donc $\mathbf{P} \subseteq \mathbf{NP}$
- On pense que facile à vérifier n'implique nécessairement pas facile à résoudre, donc $\mathbf{P} \neq \mathbf{NP}$
- Mais ça reste un problème ouvert, d'ou le 1000000 $

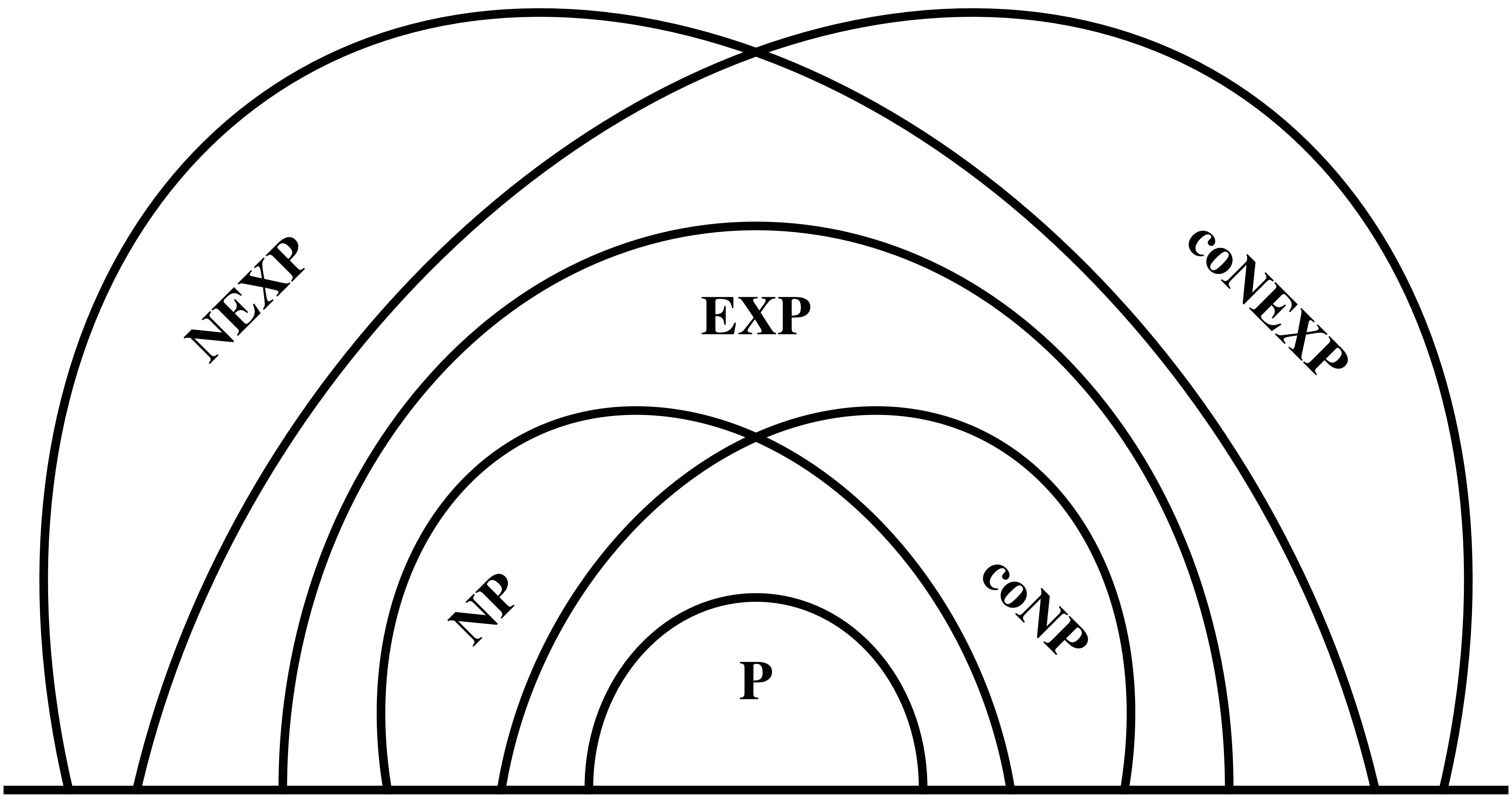
Proposition 2-BA (p. 62)
Caractérisation universelle de $\mathbf{coNP}$

Un langage L appartient à $\mathbf{coNP}$ ssi il existe une machine de Turing déterministe M qui fonctionne en temps polynomial et un polynôme $p(n)$ tels que

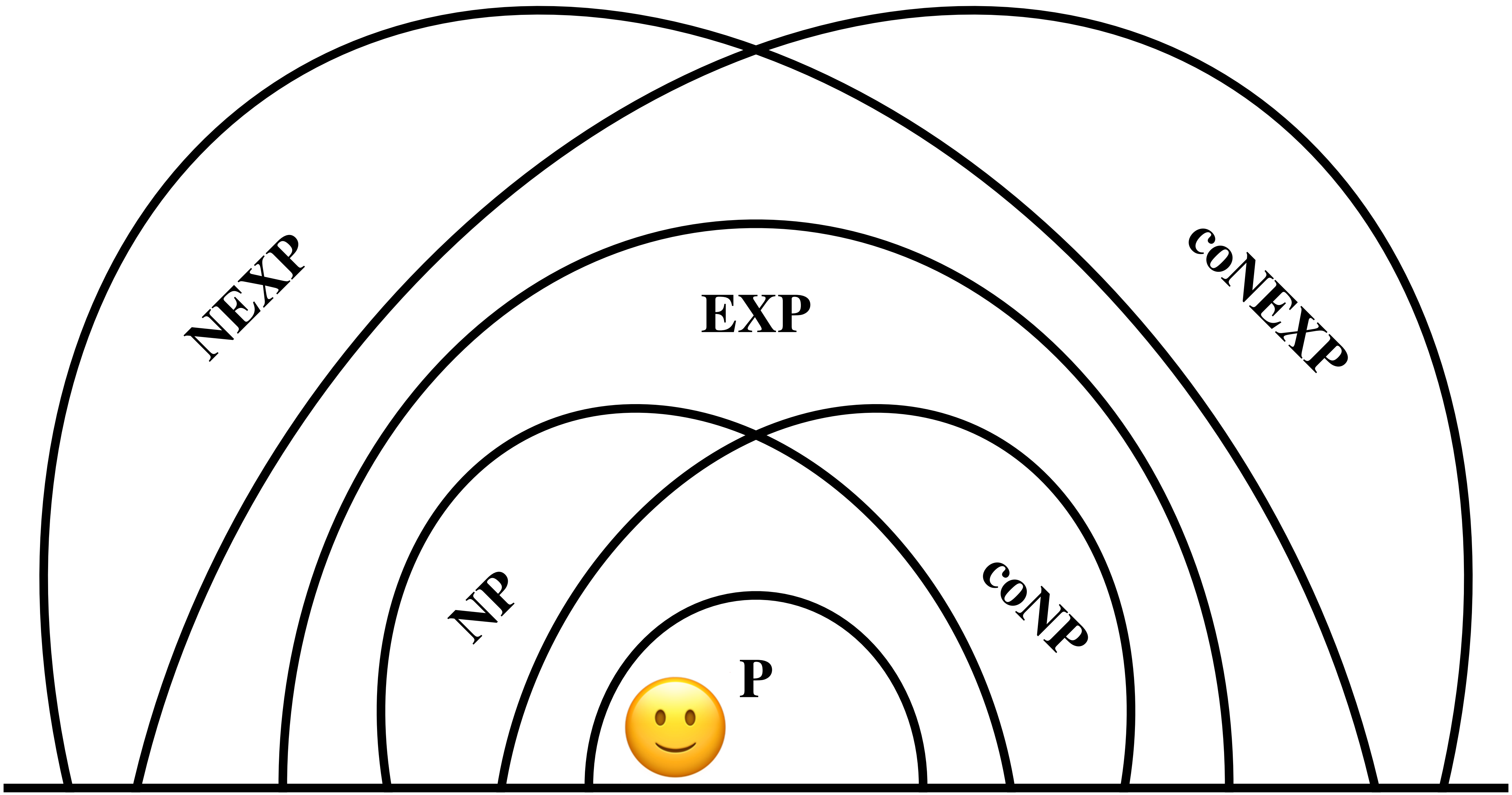
$$x \in L \text{ ssi } \forall y \in \{0,1\}^{p(n)}\ M(x, y) \text{ accepte}$$

Réductions,
ou comment classer les problèmes par difficulté

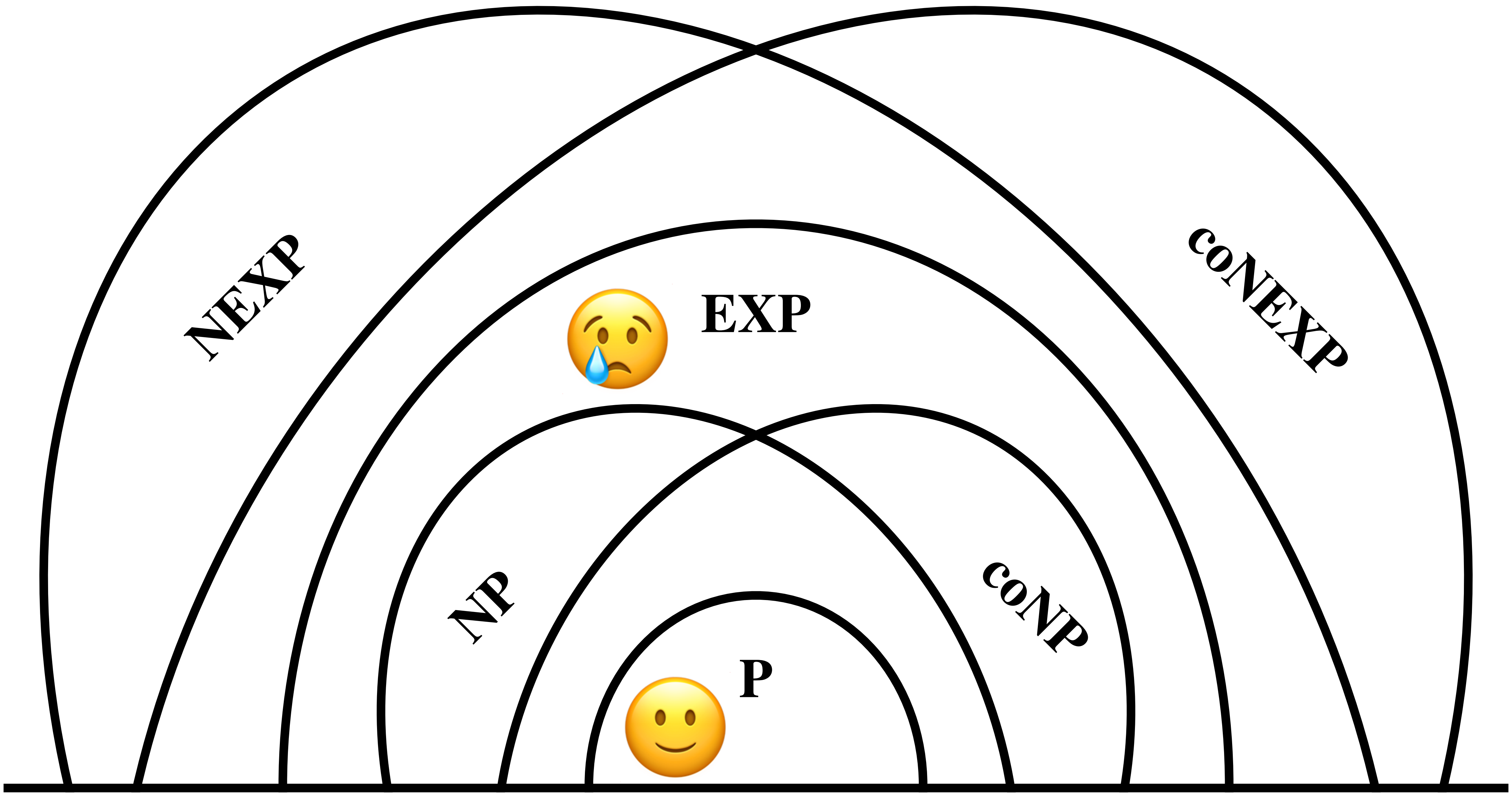
Hiérarchie des classes de complexité



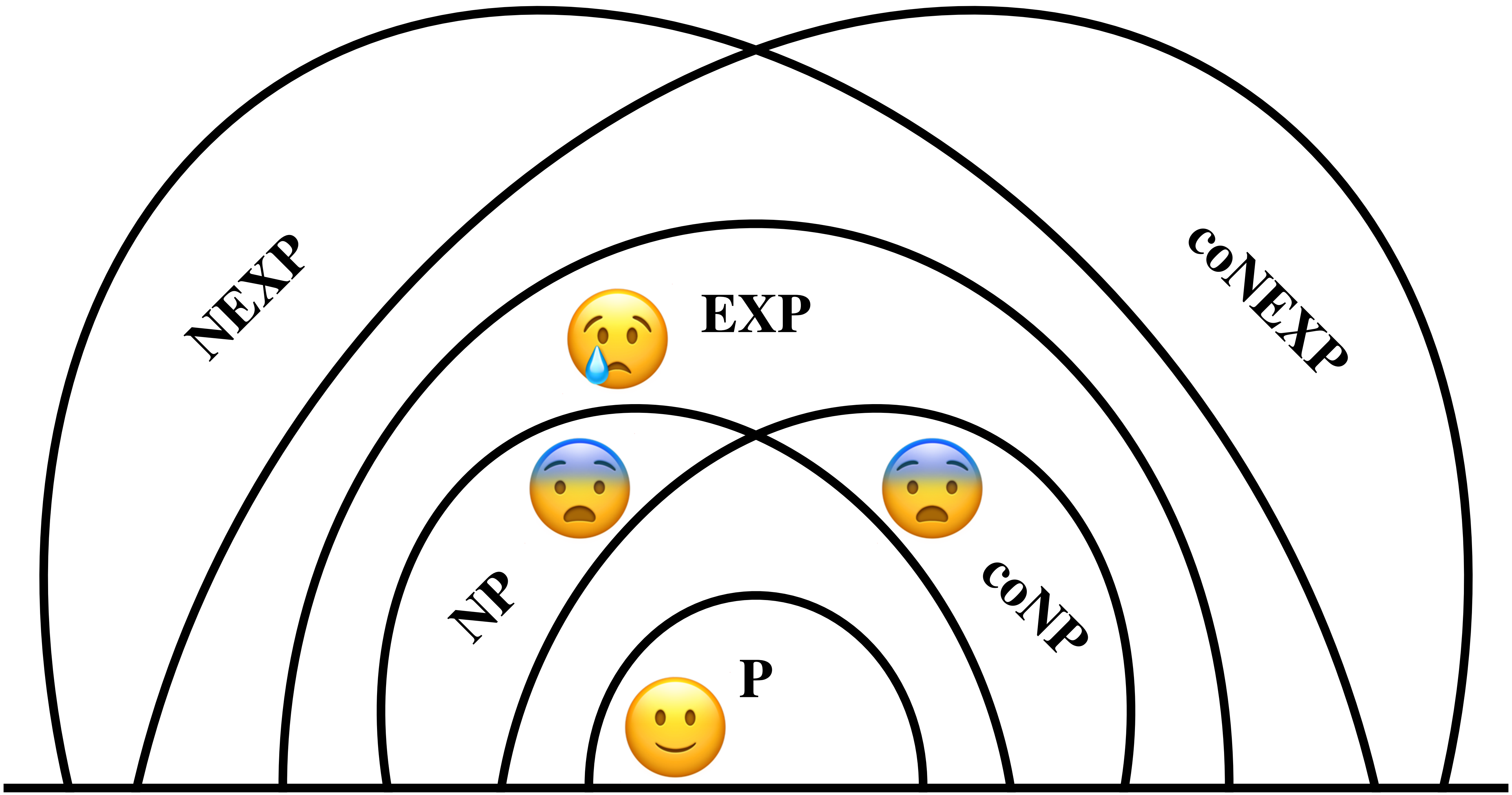
Hiérarchie des classes de complexité



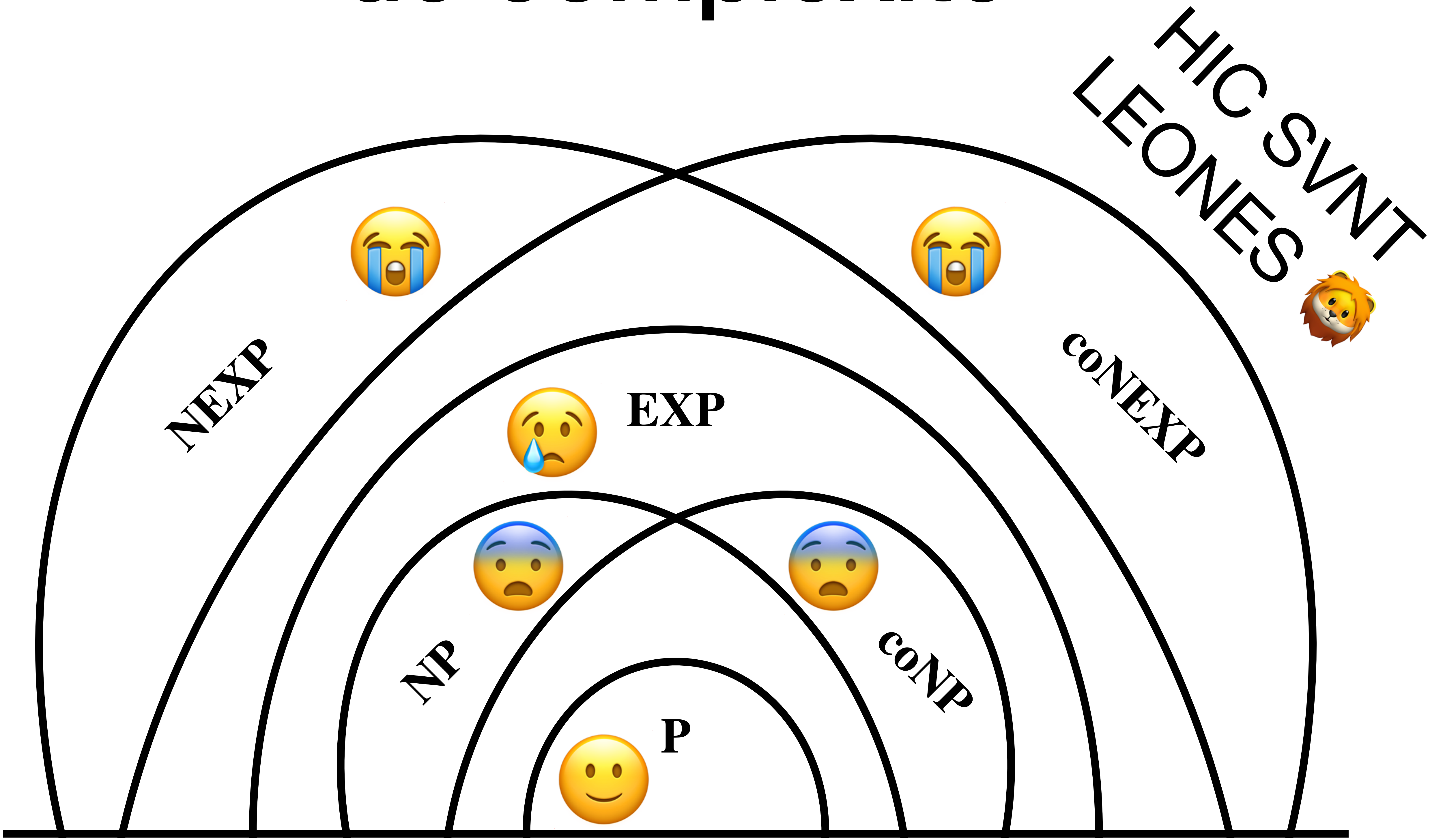
Hiérarchie des classes de complexité



Hiérarchie des classes de complexité



Hiérarchie des classes de complexité



Ordre de difficulté des problèmes

- On veut classer les problèmes (et pas les algorithmes !) en ordre de difficulté

- C'est quoi la difficulté ?

- On voudrait que ce soit liée de quelque façon au temps de calcul

Ordre de difficulté des problèmes

- Par exemple, $L_1 \leq L_2$ si le meilleur algorithme (ou machine de Turing) pour L_1 est plus rapide du meilleur algorithme pour L_2

- Problème : souvent on ne sait pas quel est l'algorithme meilleur pour un problème !

Alternative : réductions

- On dit que L_1 est plus facile que L_2 si on peut reconnaitre facilement L_1 quand on a un moyen de reconnaitre L_2
- Si c'est le cas, résoudre L_2 nous permet de résoudre L_1, donc L_1 n'est pas plus difficile que L_2
- Dit autrement, pour résoudre L_1 on se ramène à L_2…
- …ou, plus formellement, L_1 se réduit à L_2

Définition 3-A (p. 64)
Réductions (many-one) polynomiales

- Une réduction (many-one) en temps polynomial d'un problème L_1 (sur l'alphabet Σ_1) à un problème L_2 (sur l'alphabet Σ_2) est une fonction $f\colon \Sigma_1^\star \to \Sigma_2^\star$ calculable en temps polynomial déterministe telle que

$$\forall x \in \Sigma_1^\star \quad x \in L_1 \iff f(x) \in L_2$$

- Si une telle f existe, on dit que L_1 se réduit à L_2 (via f) et on notera $L_1 \leq_{\mathrm{m}}^{\mathbf{P}} L_2$ (ou parfois, en bref, $L_1 \leq L_2$)

Lemme 3-G (p. 66)

$\leq_m^P$ (pré-)ordonne les langages

- $\leq$ est réflexive : $L_1 \leq L_1$ via l'identité $f\colon\ \Sigma_1^\star \to \Sigma_1^\star$, qui évidemment est calculable en temps polynomial

- $\leq$ est transitive : soit $L_1 \leq L_2$ via $f\colon\ \Sigma_1^\star \to \Sigma_2^\star$ et $L_2 \leq L_3$ via $g\colon\ \Sigma_2^\star \to \Sigma_3^\star$

- alors $x \in L_1 \iff f(x) \in L_2$ et $f(x) \in L_2 \iff g(f(x)) \in L_3$, donc $x \in L_1 \iff g(f(x)) \in L_3$

- si f est calculable en temps polynomial $p(n)$
  alors $|f(x)| \leq p(|x|) = p(n)$

- si g est calculable en temps polynomial $q(n)$, alors la fonction composée $g \circ f$ est calculable en temps polynomial $O(p(n) + q(p(n)))$

Definition 3-H (p. 66)
Problèmes equivalents

- Si $L_1 \leq_{\mathrm{m}}^{\mathbf{P}} L_2$ et $L_2 \leq_{\mathrm{m}}^{\mathbf{P}} L_1$ alors on écrit $L_1 \equiv_{\mathrm{m}}^{\mathbf{P}} L_2$ (parfois, en bref, $L_1 \equiv L_2$)
- On dit que les problèmes L_1 et L_2 sont equivalents pour les reductions (many-one) polynomiales
- Par exemple, on a toujours $L_1 \equiv L_1$

Résoudre L_1 avec une réduction

x

M_2

Résoudre L_1 avec une réduction

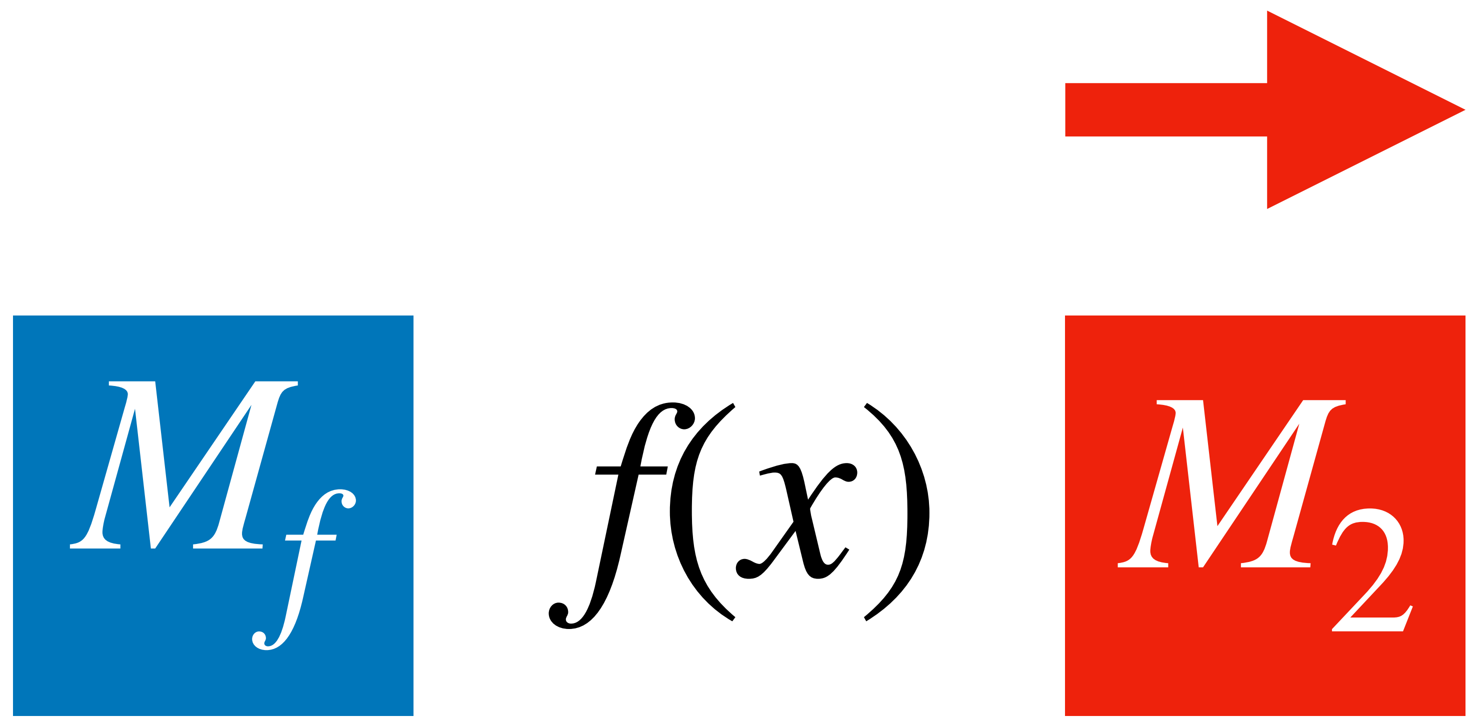
x M_f M_2

Résoudre L_1 avec une réduction

Résoudre L_1 avec une réduction

M_f $f(x)$ M_2

Résoudre L_1 avec une réduction



Résoudre L_1 avec une réduction

M_f M_2 **oui**

Résoudre L_1 avec une réduction

M_f

M_2

oui

non

Résoudre L_1 avec une réduction

M_f

M_2

oui

non

$$\forall x \in \Sigma_1^\star \quad x \in L_1 \iff f(x) \in L_2$$

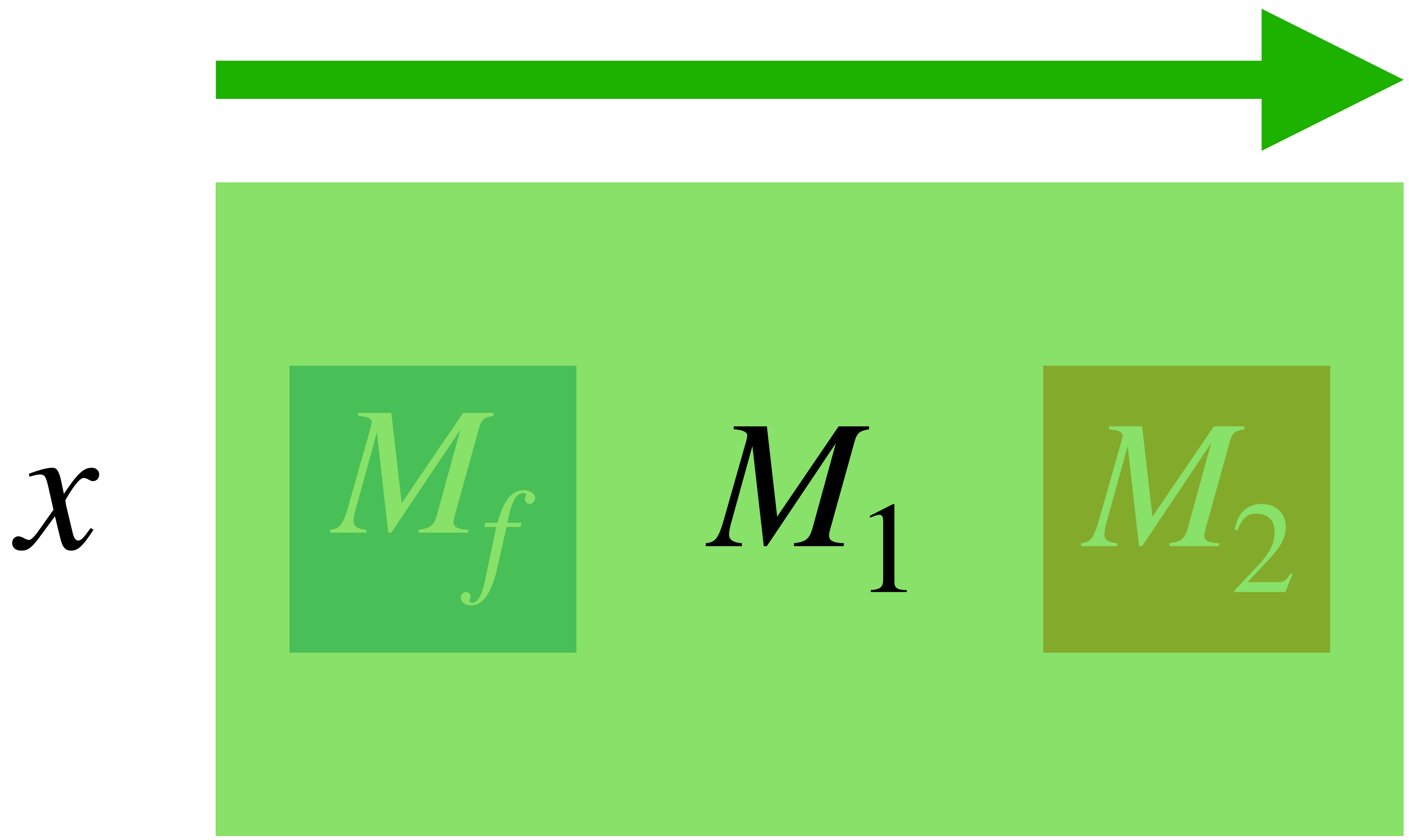
Résoudre L_1 avec une réduction

M_f $\qquad\qquad$ M_2

Résoudre L_1 avec une réduction

Résoudre L_1 avec une réduction

Résoudre L_1 avec une réduction



Résoudre L_1 avec une réduction

oui

Résoudre L_1 avec une réduction

Proposition 3-C (p. 64)

$\mathbf{P}$ est close pour $\leq$

Si $L_2 \in \mathbf{P}$ et $L_1 \leq L_2$, alors $L_1 \in \mathbf{P}$

- Soit f la fonction de réduction de L_1 à L_2 en temps polynomial $p(n)$, et soit M_2 une machine déterministe qui reconnaît L_2 en temps polynomial $q(n)$

- Alors le suivant est un algorithme déterministe pour L_1 : calculer $y = f(x)$ et retourner le résultat de $M_2(y)$

- Ça prend temps $p(n)$ pour calculer f et $q(p(n))$ pour simuler M_2 sur $y = f(x)$, donc temps polynomial, donc $L_1 \in \mathbf{P}$

Proposition 3-C (p. 64)

$\mathbf{P}$ est close pour $\leq$

Ça veut dire que que si L_2 est efficacement resoluble et $L_1 \leq L_2$, c'est-à-dire que L_1 est plus simple que L_2, alors L_1 est aussi efficacement resoluble, conformément à l'intuition

Proposition 3-C (p. 64)

$\mathbf{NP}$ est close pour $\leq$

Si $L_2 \in \mathbf{NP}$ et $L_1 \leq L_2$, alors $L_1 \in \mathbf{NP}$

- Soit f la fonction de réduction de L_1 à L_2 en temps polynomial $p(n)$, et soit M_2 une machine non déterministe qui reconnaît L_2 en temps polynomial $q(n)$

- Alors le suivant est un algorithme non déterministe pour L_1 : calculer $y = f(x)$ et retourner le résultat de $M_2(y)$

- Ça prend temps $p(n)$ pour calculer f et $q(p(n))$ pour simuler M_2 sur $y = f(x)$, donc temps polynomial, donc $L_1 \in \mathbf{NP}$

Proposition 3-C (p. 64)

$\mathbf{NP}$ est close pour $\leq$

Ça veut dire que que si L_2 est efficacement verifiable
et $L_1 \leq L_2$, c'est-à-dire que L_1 est plus simple que L_2,
alors L_1 est aussi efficacement verifiable, conformément
à l'intuition

Exemple 3-I (p. 66)

Une réduction

Problème CLIQUE

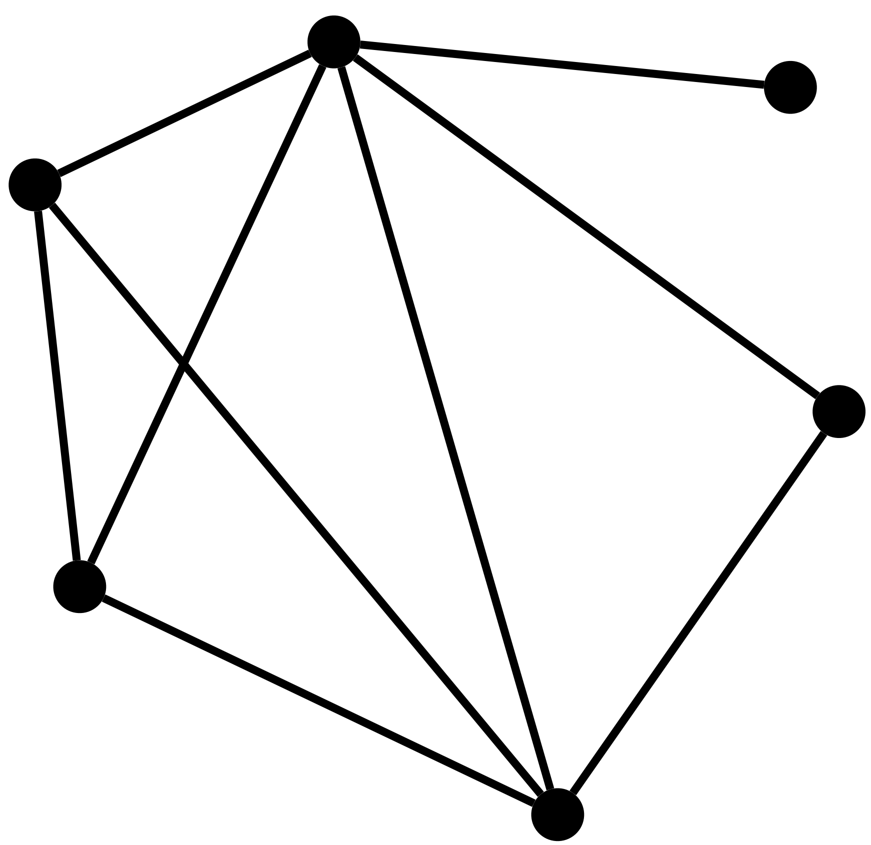
- Entrée : un graphe non orienté G et un entier k
- Question : G a-t-il une clique de taille k ?

Problème ENSEMBLE-INDÉPENDANT

- Entrée : un graphe non orienté G et un entier k
- Question : existe-t-il un ensemble de k sommets indépendants dans G, c'est-à-dire tous non reliés deux à deux ?

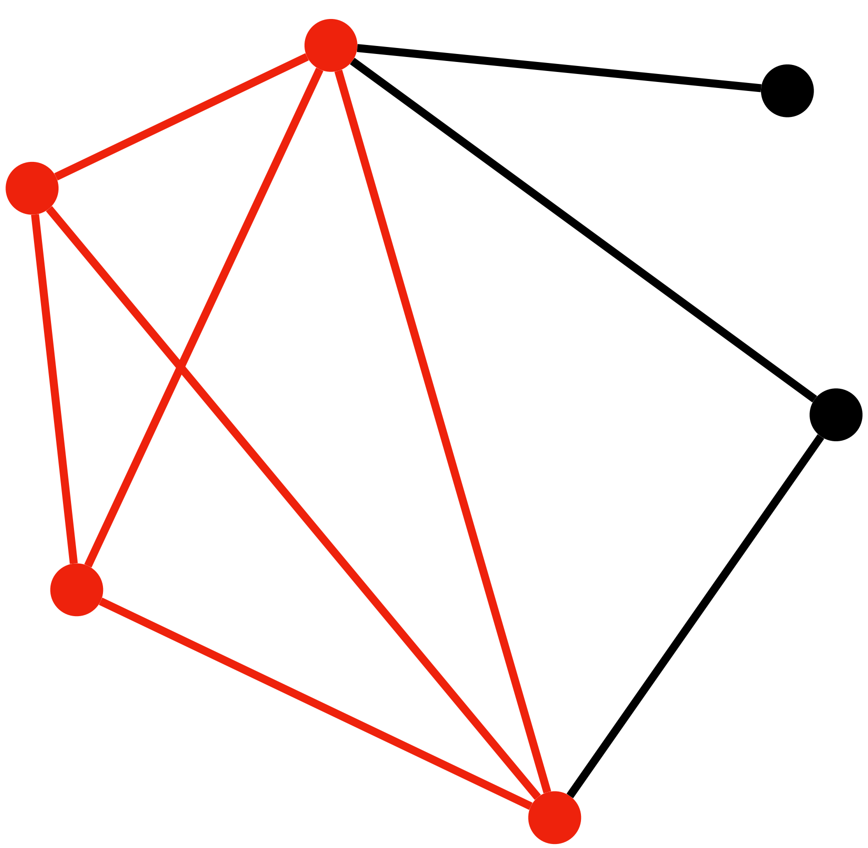
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



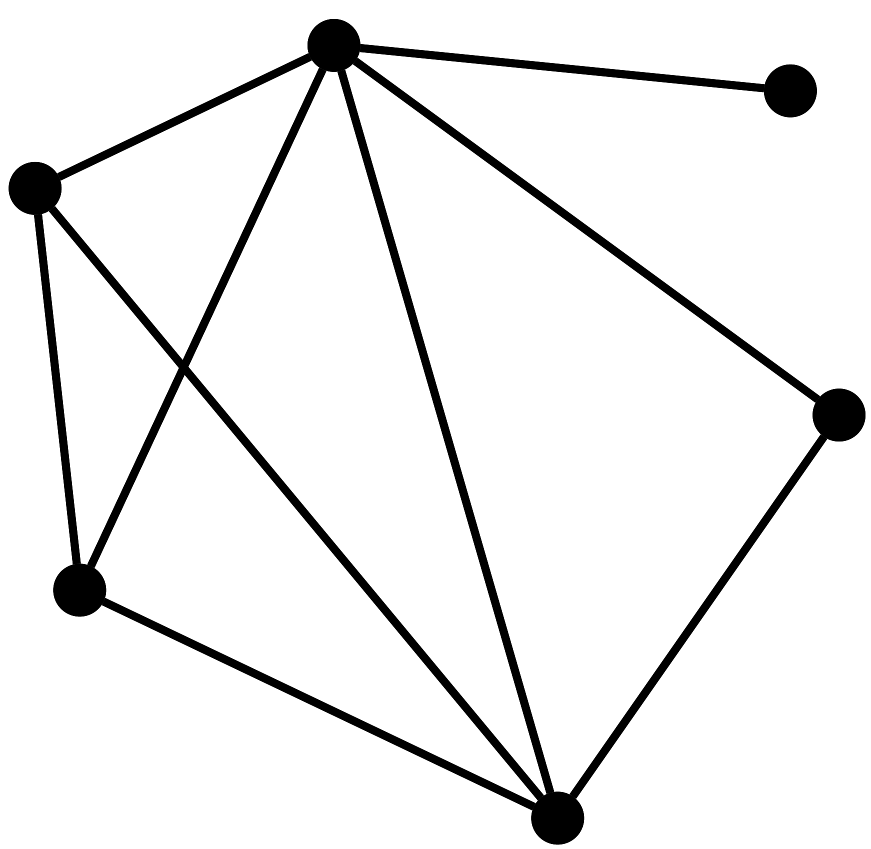
CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



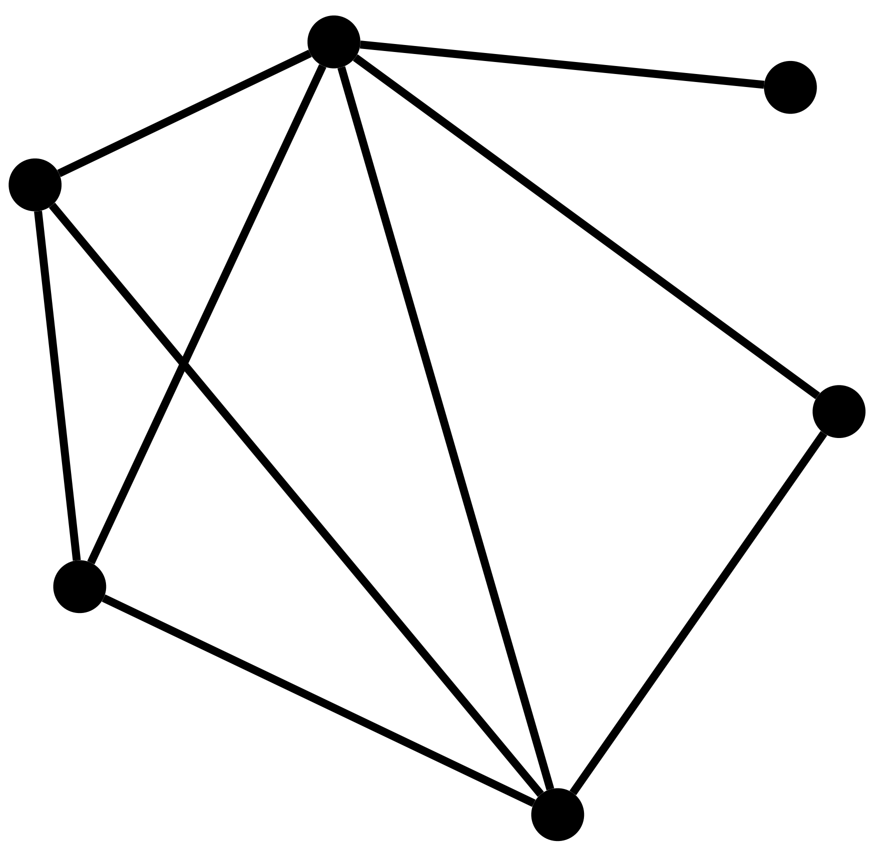
CLIQUE vs ENS-INDÉP

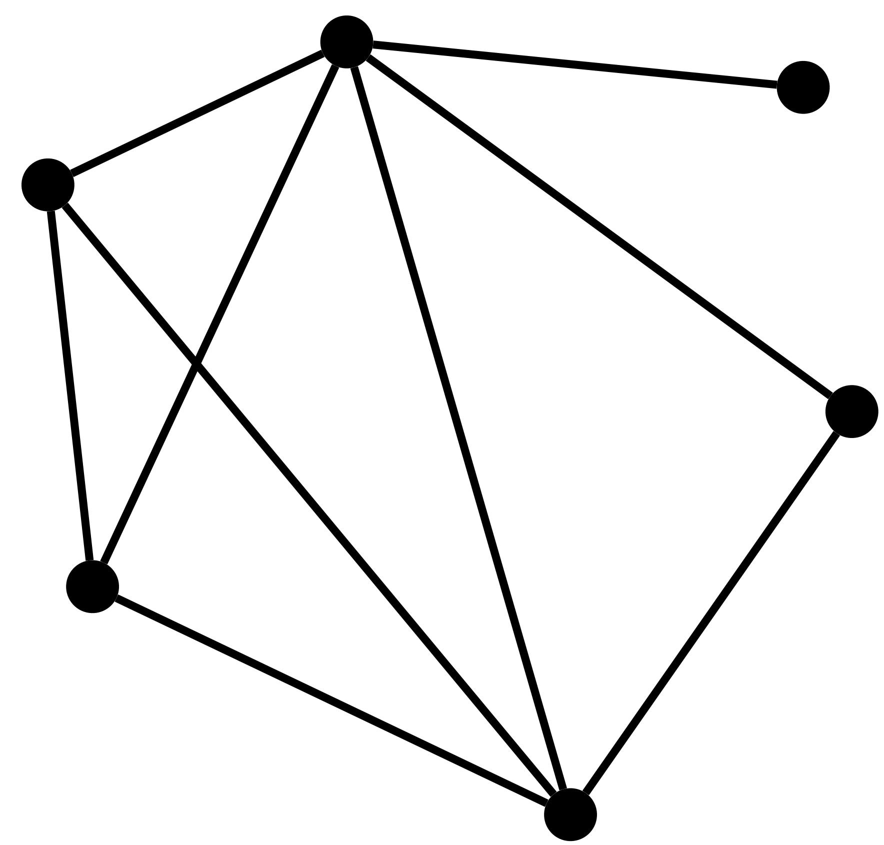
$$G = (V, E)$$



CLIQUE vs ENS-INDÉP

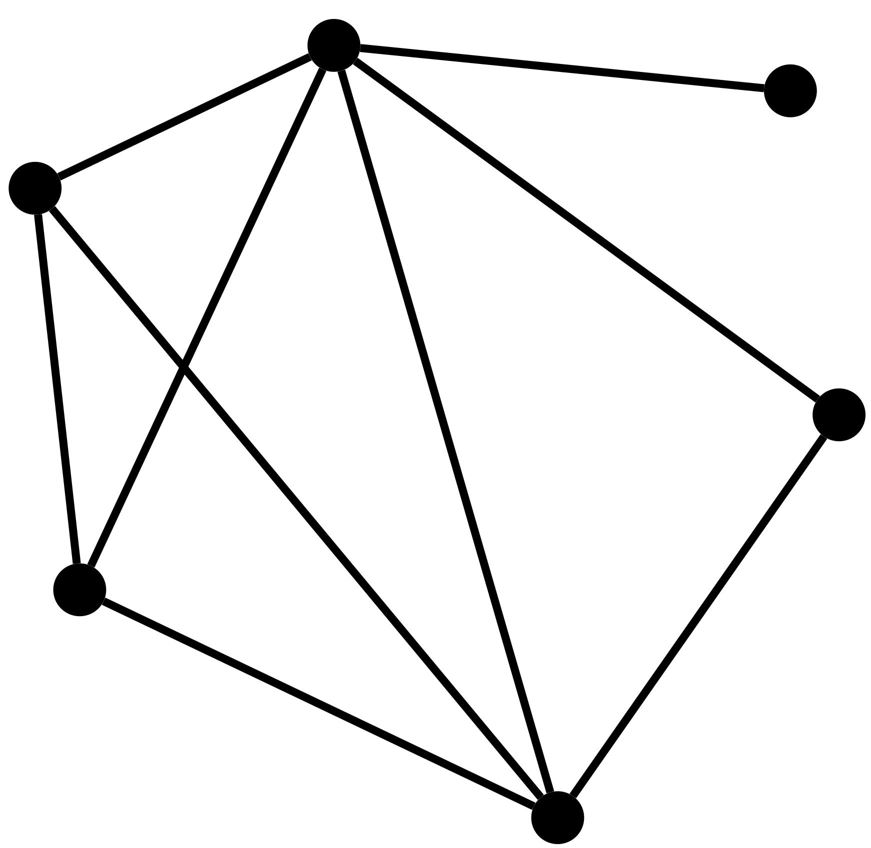
$$G = (V, E)$$

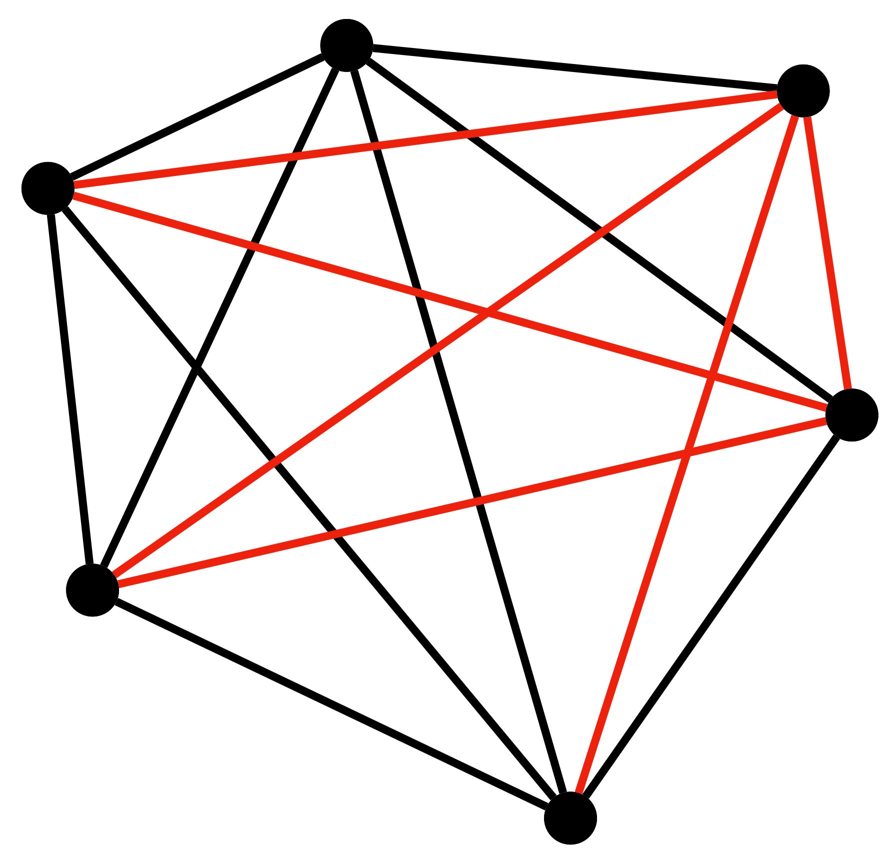




CLIQUE vs ENS-INDÉP

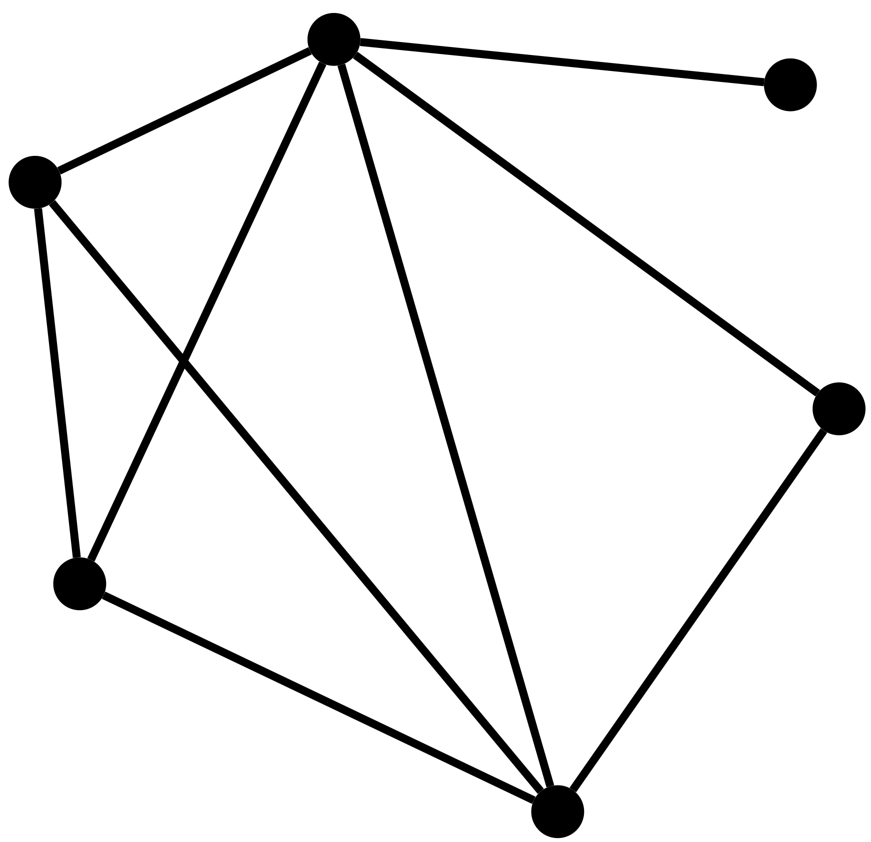
$$G = (V, E)$$

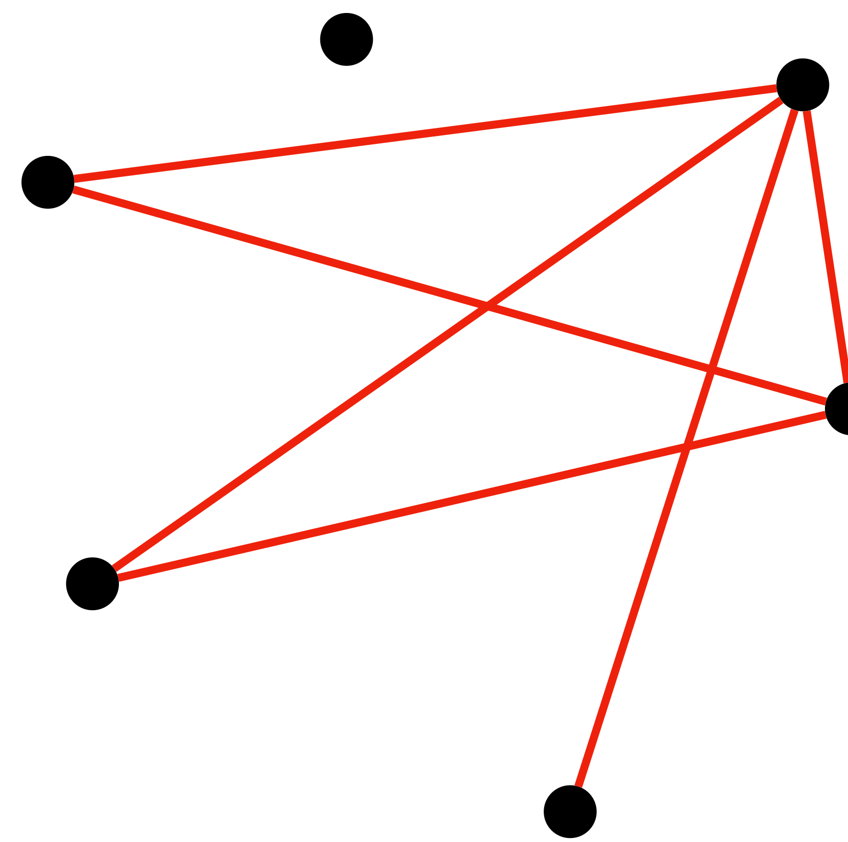




CLIQUE vs ENS-INDÉP

$$G = (V, E)$$

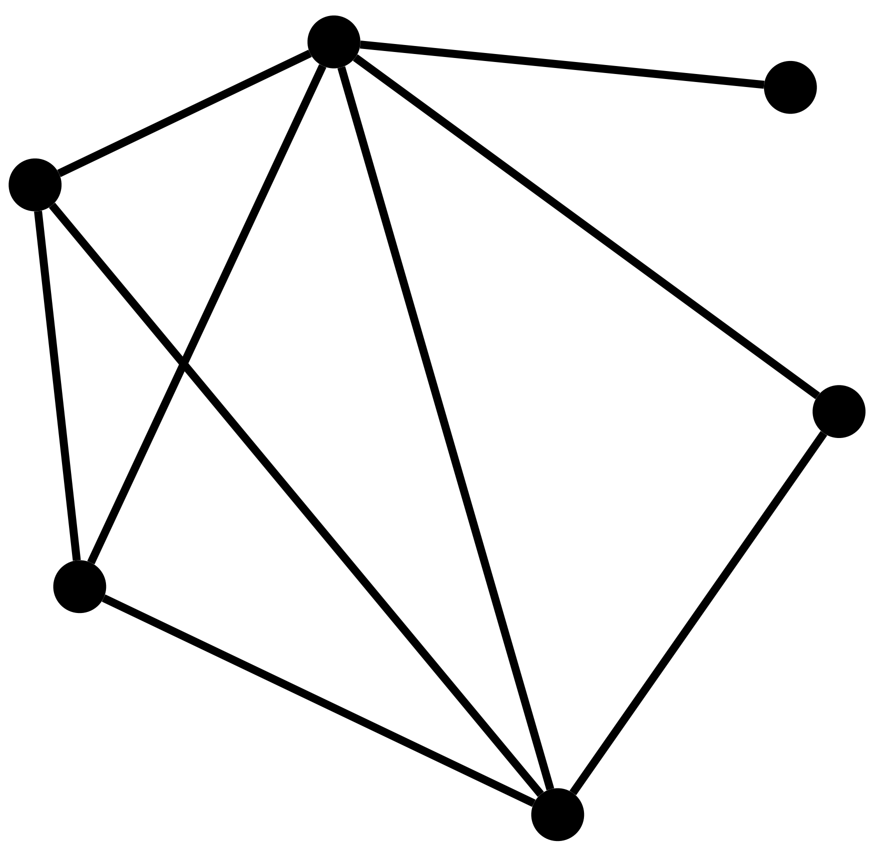


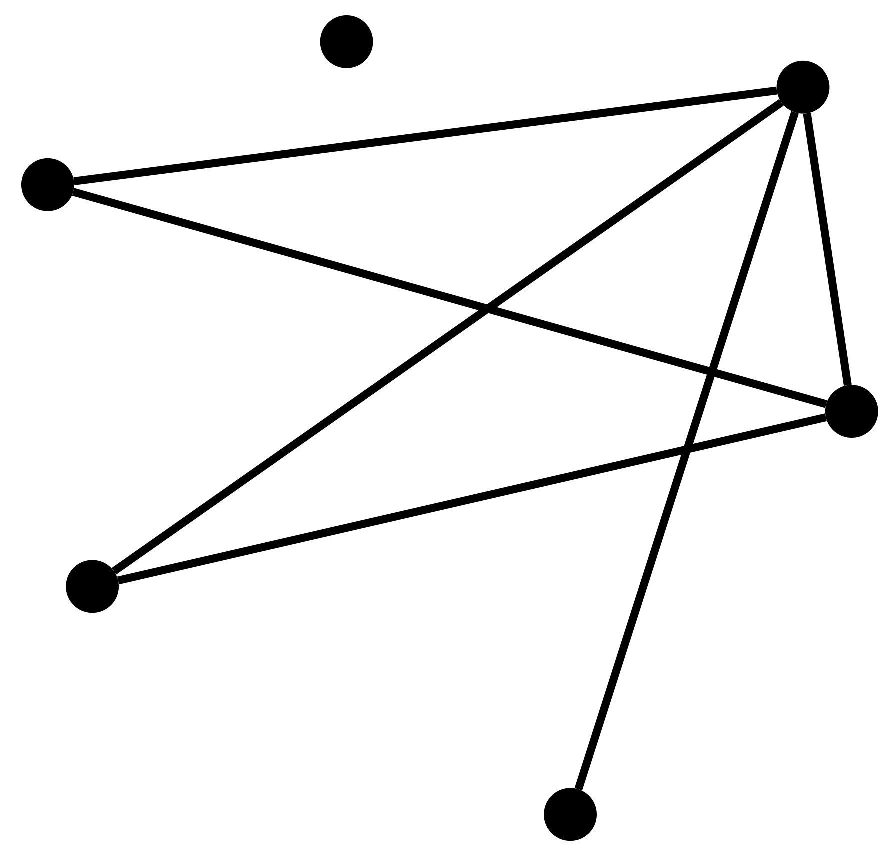


CLIQUE vs ENS-INDÉP

$G = (V, E)$

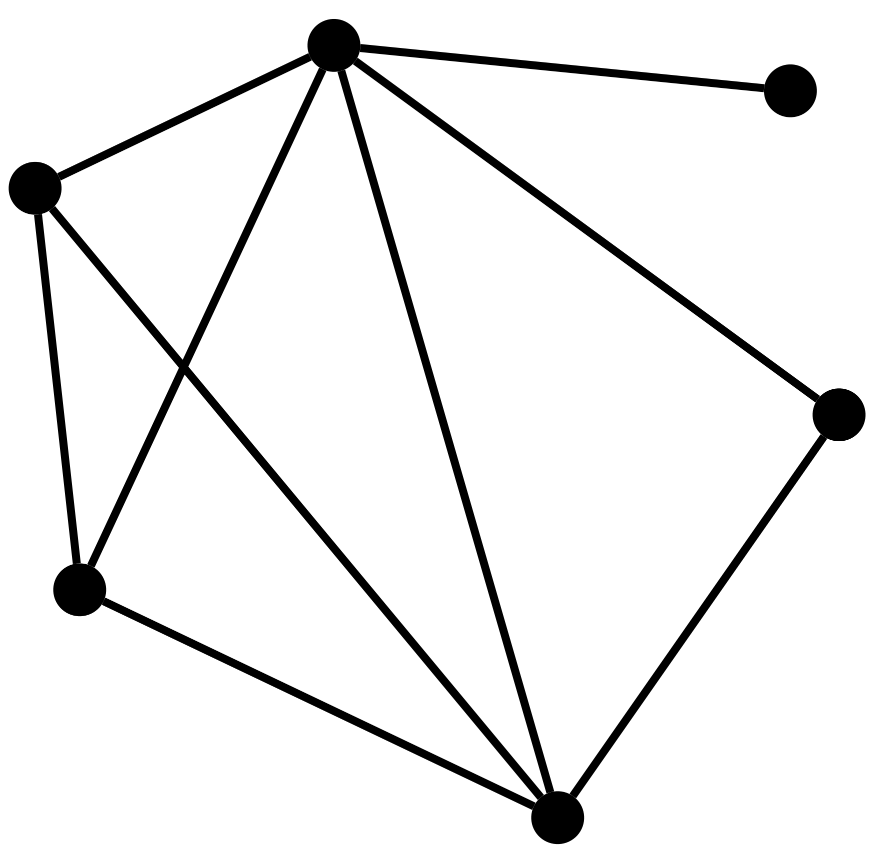
$\overline{G} = (V, V^2 - E)$



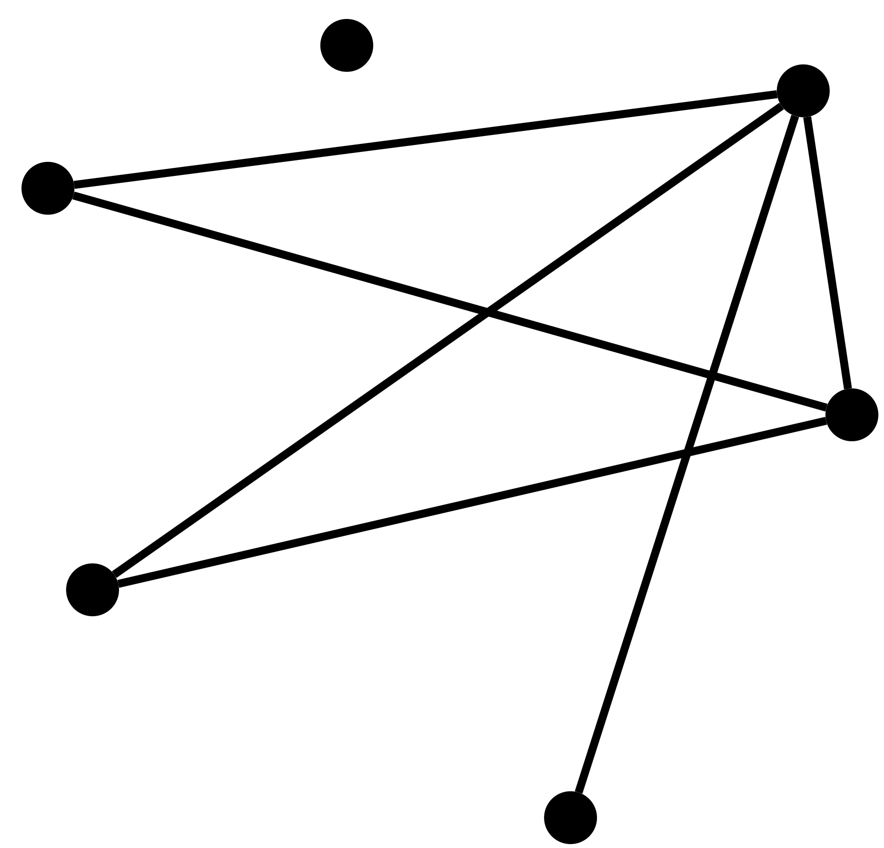


CLIQUE vs ENS-INDÉP

$$G = (V, E)$$



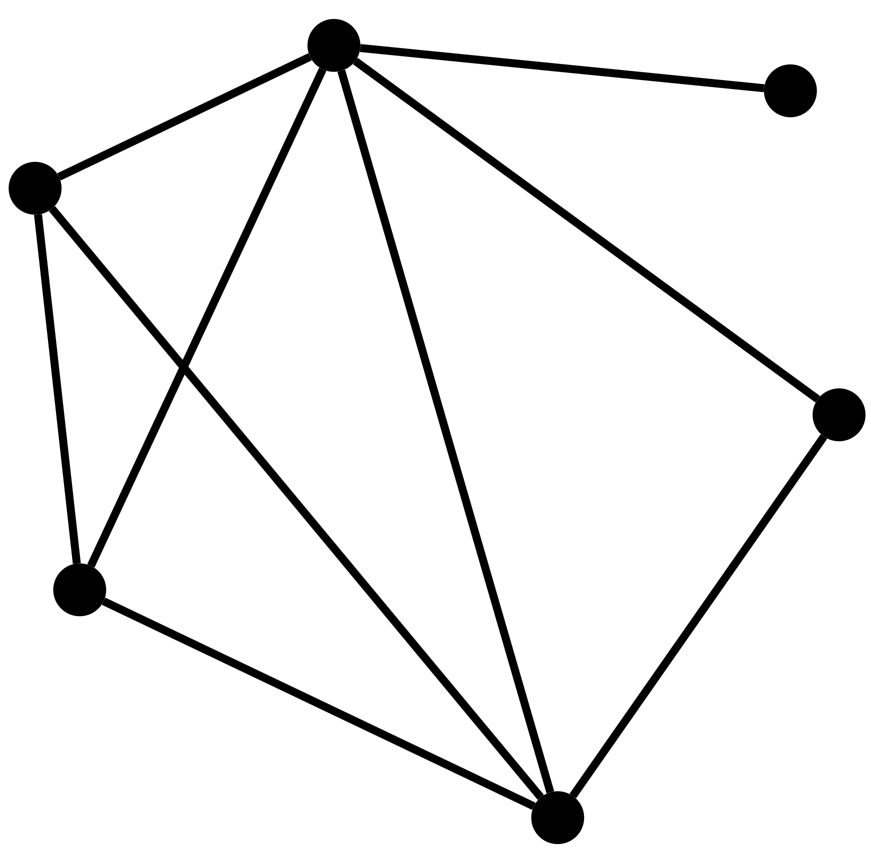
$$\overline{G} = (V, V^2 - E)$$

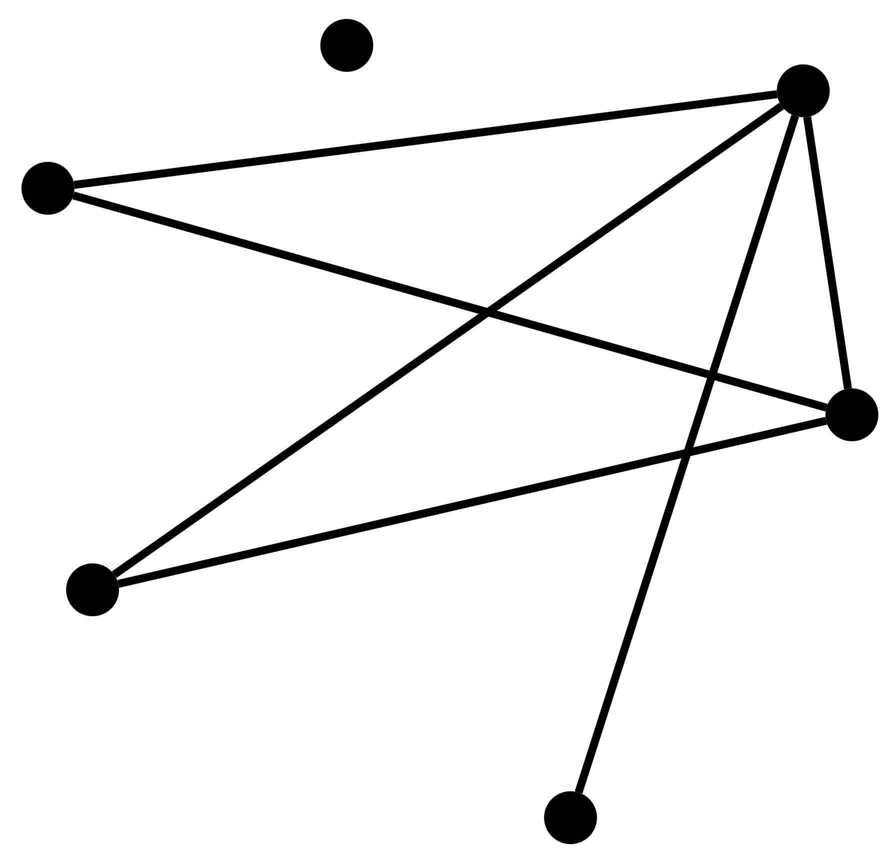


a une clique de taille k

CLIQUE vs ENS-INDÉP

$G = (V, E)$ $\qquad$ $\overline{G} = (V, V^2 - E)$

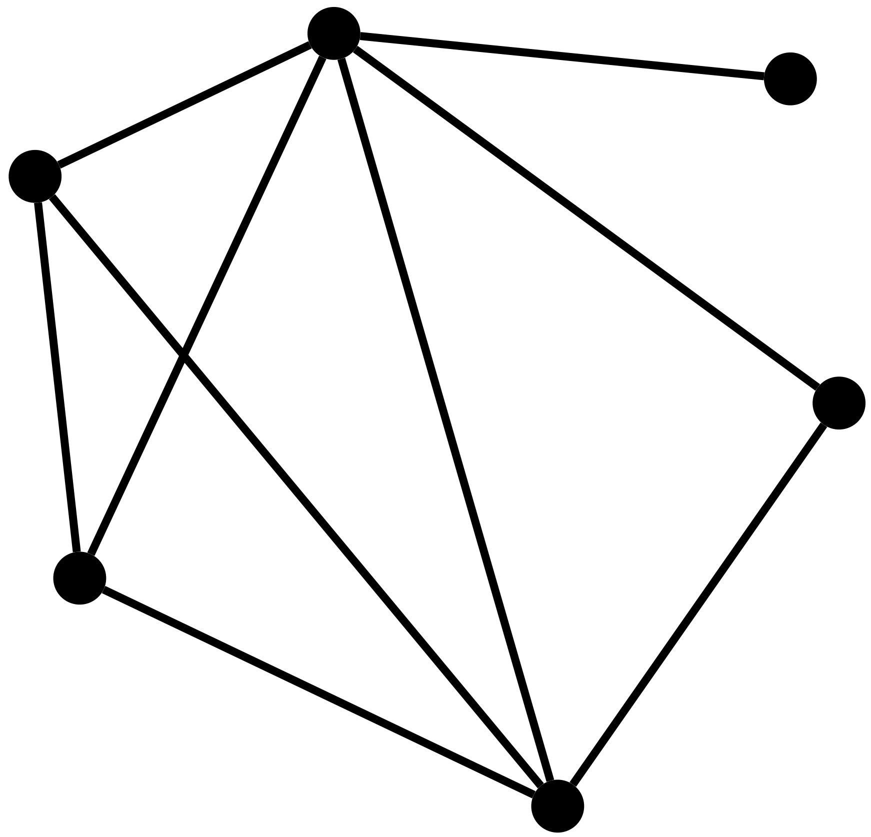


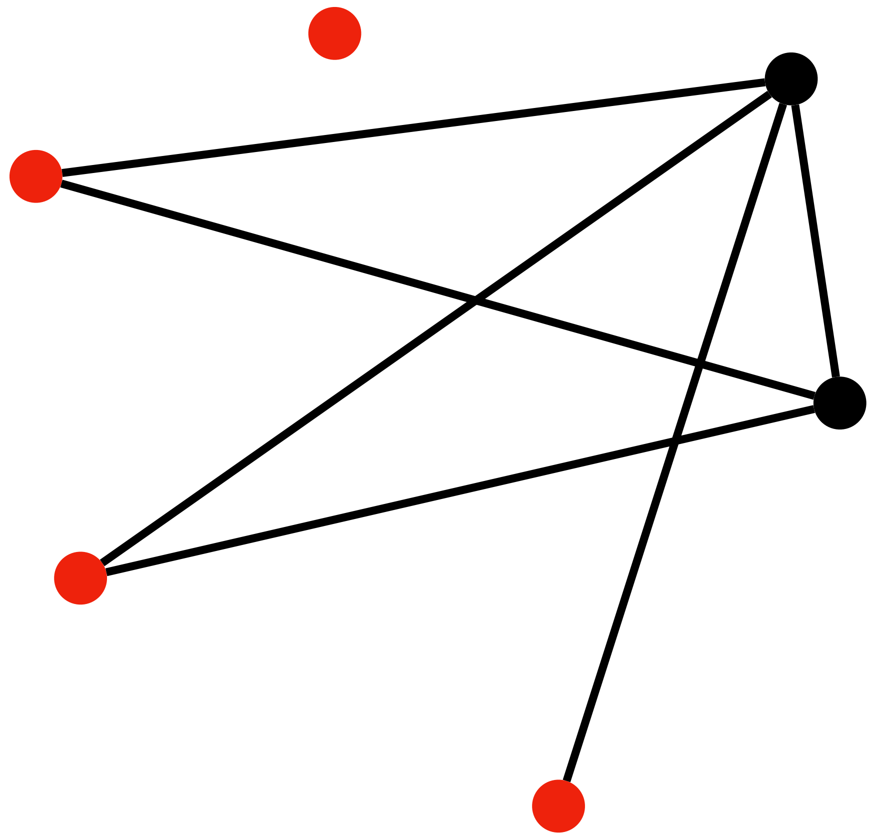


a une clique de taille k $\iff$ a un ens. indép. de taille k

CLIQUE vs ENS-INDÉP

$G = (V, E)$ $\qquad\qquad$ $\overline{G} = (V, V^2 - E)$





a une clique de taille k $\iff$ **a un ens. indép. de taille** k

CLIQUE $\leq$ ENS-INDÉP

- Supposons que CLIQUE et ENS-INDÉP sont définis sur le même alphabet Σ (raisonnable, c'est toujours des graphes !)

- La fonction $f\colon \Sigma^\star \to \Sigma^\star$ définie par $f(V, E, k) = (V, V^2 - E, k)$ est calculable en temps polynomial

- (V, E) a une clique de taille k ssi $f(V, E, k)$ a un ensemble indépendant de taille k

- Donc $(V, E, k) \in$ CLIQUE iff $f(V, E, k) \in$ ENS-INDÉP

Mais aussi ENS-INDÉP $\leq$ CLIQUE !

- On utilise la même fonction f: $\Sigma^\star \to \Sigma^\star$ définie par $f(V, E, k) = (V, V^2 - E, k)$, toujours calculable en temps polynomial*

- (V, E) a un ensemble indépendant de taille k ssi $f(V, E, k)$ a une clique de taille k

- Donc $(V, E, k) \in$ ENS-INDÉP iff $f(V, E, k) \in$ CLIQUE

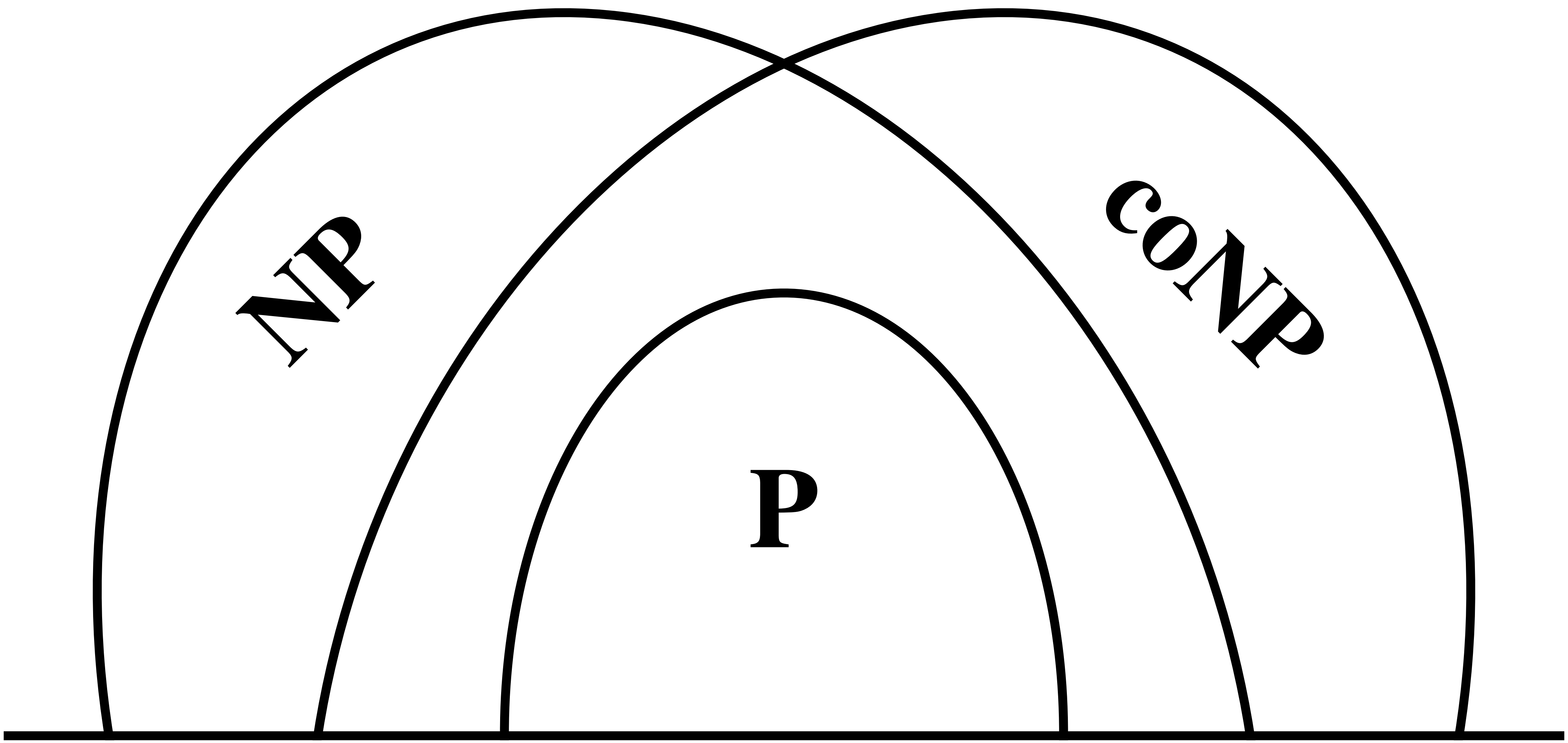
* C'est un cas exceptionnel ! Normalement il faut changer de fonction

ENS-INDÉP $\equiv$ CLIQUE

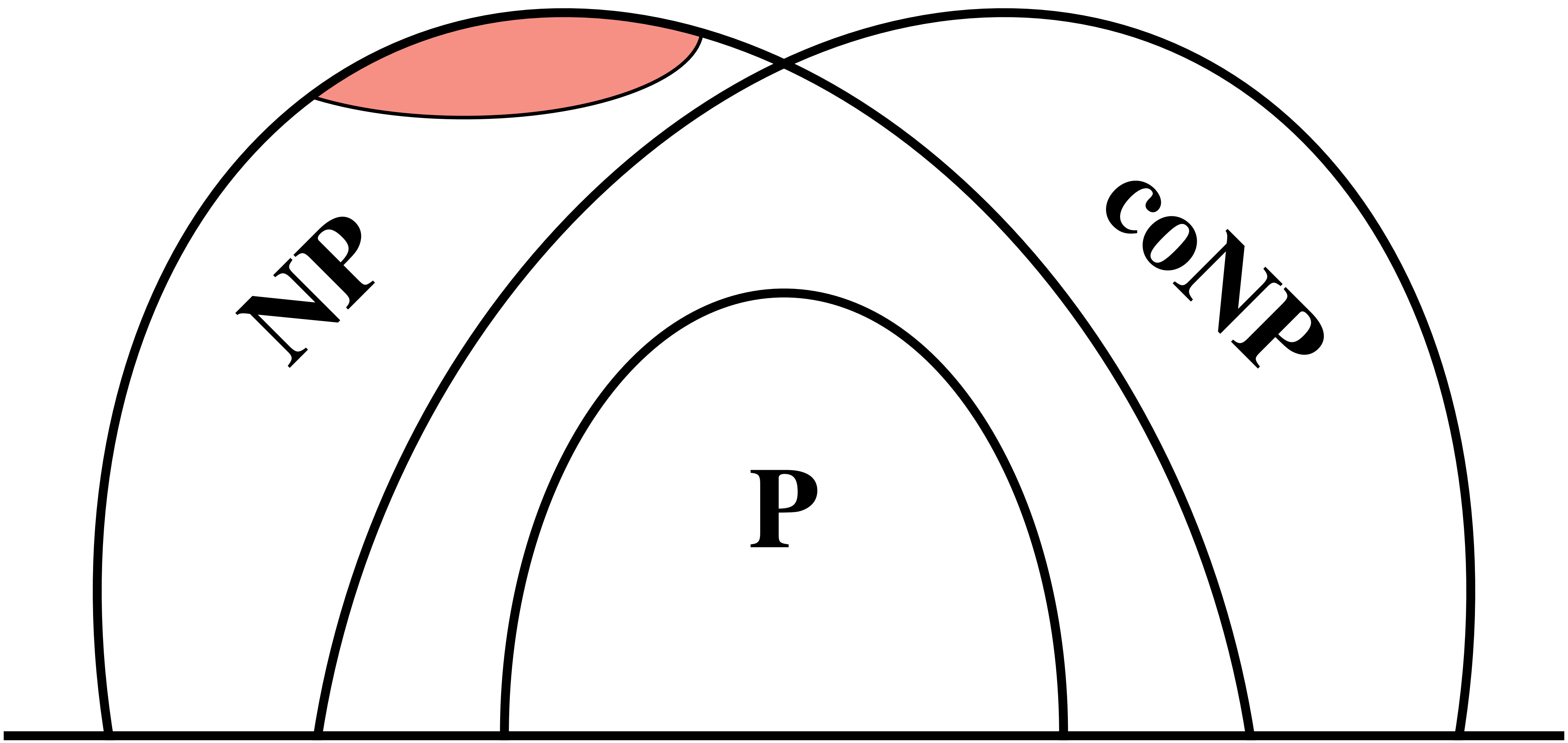
- On sait que CLIQUE $\in \mathbf{NP}$ (exercice du TD)
- Puisque ENS-INDÉP $\leq$ CLIQUE, on a ENS-INDÉP $\in \mathbf{NP}$, parce que $\mathbf{NP}$ est clos pour $\leq$
- Si on découvrait que CLIQUE $\in \mathbf{P}$, on aurait aussi ENS-INDÉP $\in \mathbf{P}$, parce que $\mathbf{P}$ est aussi clos par $\leq$
- Même chose si on découvrait que ENS-INDÉP $\in \mathbf{P}$: ça impliquerait CLIQUE $\in \mathbf{P}$

Complétude,
ou les problèmes les plus difficiles du monde

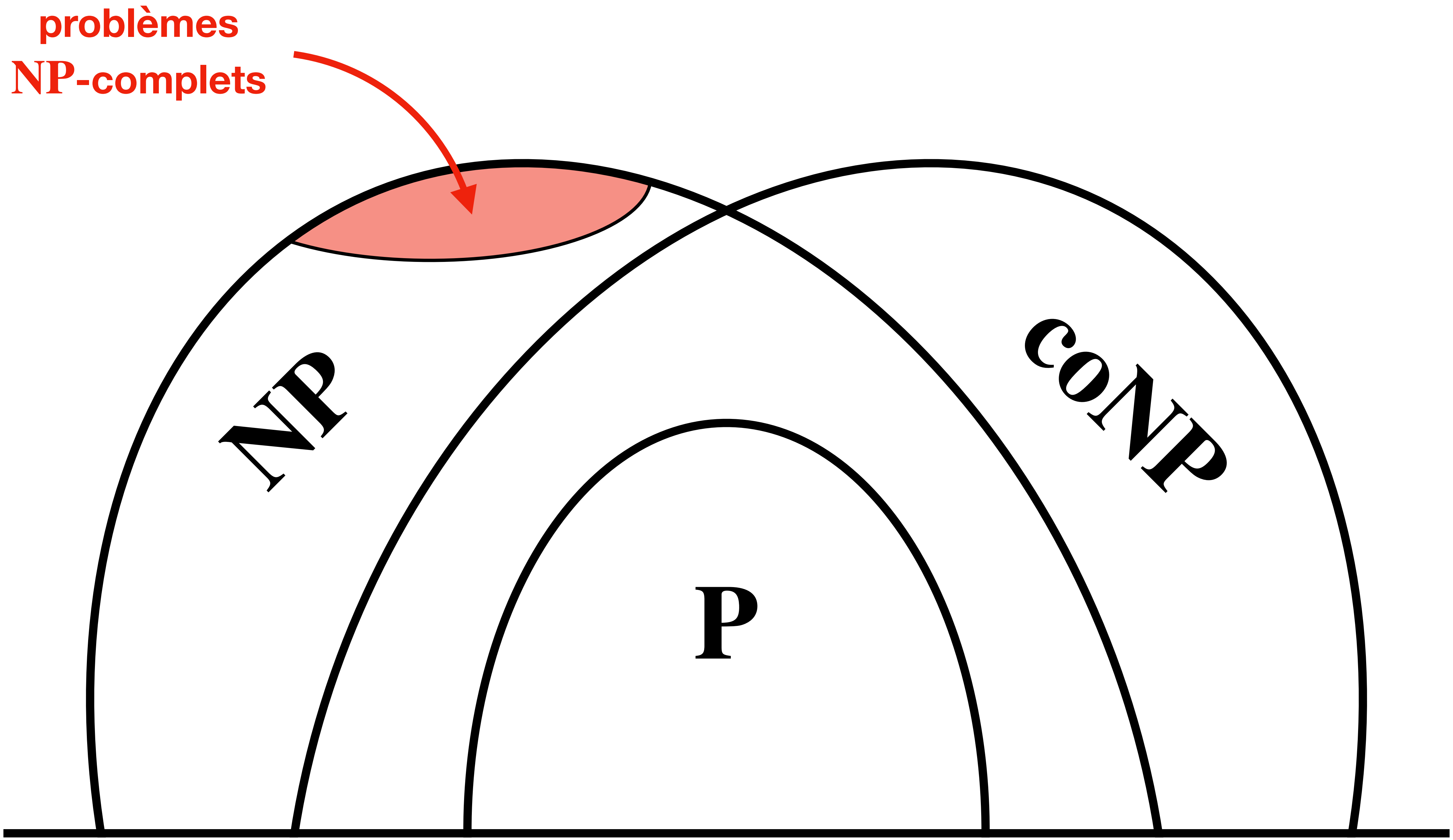
Hiérarchie du « possible »



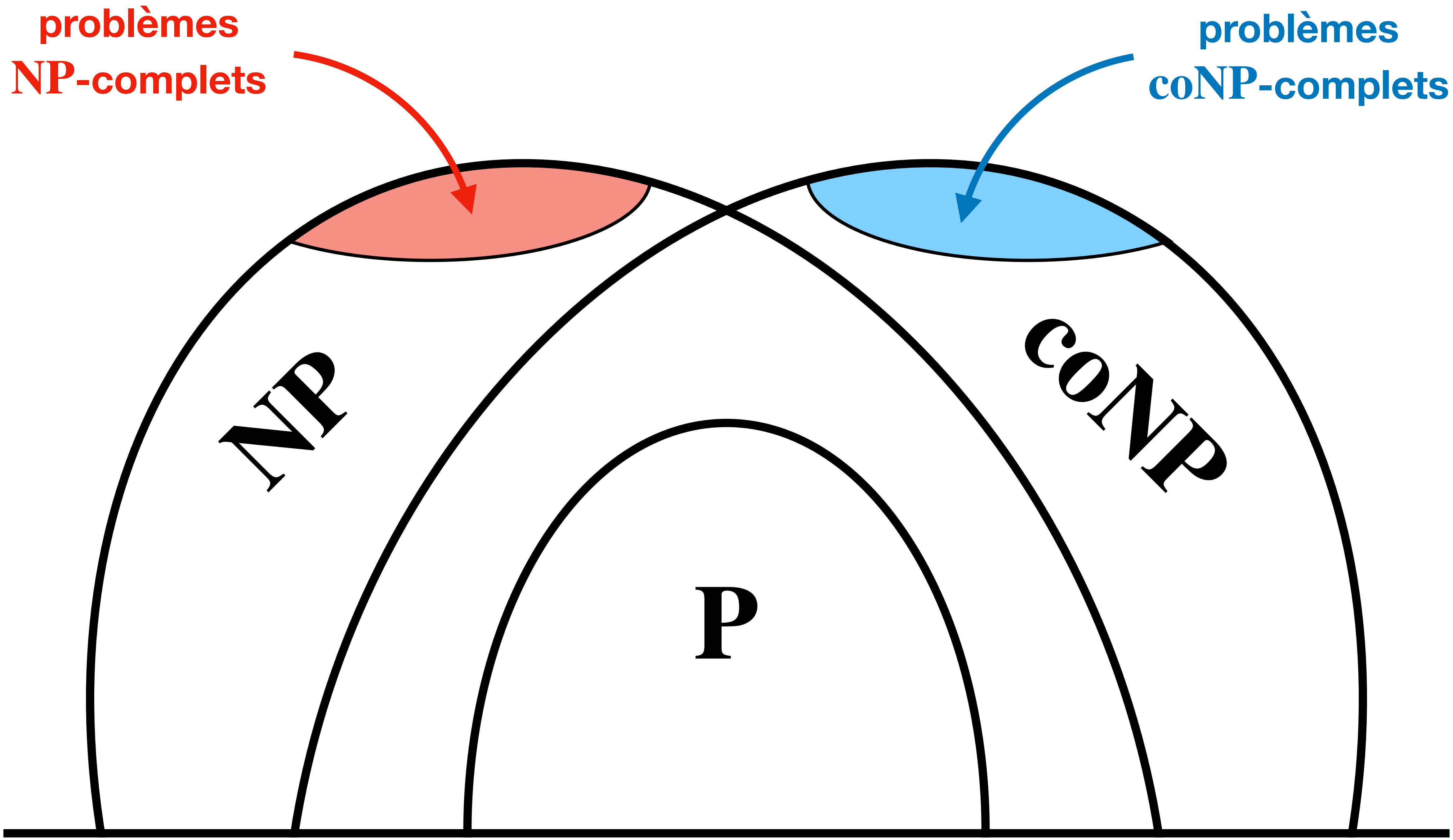
Hiérarchie du « possible »



Hiérarchie du « possible »



Hiérarchie du « possible »



Definition 3-J (p. 67)
Difficulté et complétude

Soit L un problème et $\mathscr{C}$ une classe de complexité

- On dit que L est $\mathscr{C}$-difficile (ou $\mathscr{C}$-dur) si pour tout problème $L' \in \mathscr{C}$ on a $L' \leq L$

- On dit que L est $\mathscr{C}$-complet s'il est $\mathscr{C}$-difficile et en plus on a $L \in \mathscr{C}$

Difficulté et complétude

- Les definitions sont très fortes : il faut que tous les problèmes de $\mathscr{C}$ se réduisent à L !
- À priori c'est n'est même pas évident qu'il existent des problèmes durs ou complets pour une classe…

$\mathbf{P}$ a (beaucoup de) problèmes complets

- Tout problème $L \in \mathbf{P}$ non trivial (c'est-à-dire, $L \neq \emptyset$ et $L \neq \Sigma^\star$) est $\mathbf{P}$-complet pour les reductions en temps polynomial

- Ça veut dire que cette notion de complétude n'est pas très intéressant pour $\mathbf{P}$…

Demonstration

- Soit $L \in \mathbf{P}$ avec $L \neq \emptyset$ et $L \neq \Sigma^\star$
- Alors il existe $a \in L$ et aussi $b \notin L$
- Soit $L' \in \mathbf{P}$ et soit $f(x) = \begin{cases} a & \text{si } x \in L' \\ b & \text{si } x \notin L' \end{cases}$
- Comme $L' \in \mathbf{P}$, la fonction f est calculable en temps polynomial
- Mais aussi $f(x) = a \in L$ ssi $x \in L'$, donc $L' \leq L$