

INFO-F-302, Cours d'Informatique Fondamentale

Emmanuel Filion
Département d'Informatique
Faculté des Sciences
Université Libre de Bruxelles

Année académique 2023-2024

Introduction à la théorie de la complexité

3 Problèmes de décision

- ▶ Informellement, un problème de décision est un problème dont la réponse est oui ou non.
- ▶ on peut définir un problème de décision comme un langage de mots sur un alphabet fini Σ
- ▶ on note Σ^* l'ensemble des mots sur l'alphabet Σ (et ϵ le mot vide)
- ▶ par exemple, pour $\Sigma = \{0, 1\}$, $00100 \in \Sigma^*$, $11 \in \Sigma^*$.
- ▶ un langage sur Σ est un sous-ensemble $L \subseteq \Sigma^*$.

Définition

Un problème de décision est un langage $P \subseteq \Sigma^*$.

Remarque : chaque langage P représente bien un problème dont la réponse est oui ou non, en l'identifiant à sa fonction caractéristique χ_P :

$$\begin{aligned}\chi_P &: \Sigma^* \rightarrow \{0, 1\} \\ u &\mapsto \begin{cases} 1 & \text{si } u \in P \\ 0 & \text{si } u \notin P \end{cases}\end{aligned}$$

Etant donné un mot d'entrée $u \in \Sigma^*$, on doit décider si $u \in P$ ou non.

4 Problèmes de décision : exemples

- ▶ avec $\Sigma = \{1\}$,

$$\text{PRIME}_1 = \{11, 111, 11111, 1111111, \dots\}$$

l'ensemble des nombres premiers en unaire.

- ▶ avec $\Sigma = \{1\}$,

$$\text{PAIR} = \{\epsilon, 11, 1111, \dots\}$$

l'ensemble des nombres pairs.

- ▶ avec $\Sigma = \{0, 1\}$,

$$\text{PRIME} = \{10, 11, 101, 111, \dots\}$$

l'ensemble des nombres premiers en binaire.

5 Notion de codage

- ▶ la définition précédente pour les problèmes de décision est abstraite.
- ▶ comment par exemple représenter le problème suivant comme un langage ?
 - ▶ ENTRÉE : un graphe G non dirigé et un entier $k \in \mathbb{N}$
 - ▶ SORTIE : 1 ssi on peut colorier G avec au plus k couleurs
- ▶ on voudrait représenter l'ensemble $\{(G_1, k_1), (G_2, k_2), \dots\}$ de toutes les paires de graphes G_i coloriables avec au plus k_i couleurs.
- ▶ il faut alors “coder” chaque paire comme un mot.
- ▶ par exemple, avec l'alphabet $\Sigma = \{0, 1, \#, \$\}$, on pourrait coder un graphe G en identifiant chacun de ses sommets par un entier, et chaque arête (i, j) pourrait être codée par le mot $\bar{i} \# \bar{j}$, où \bar{i}, \bar{j} sont les codages binaires des sommets i et j . La paire (G, k) pourrait se coder par le mot :

$\bar{i}_1 \# \bar{j}_1 \$ \bar{i}_2 \# \bar{j}_2 \dots \bar{i}_n \# \bar{j}_n \$ \bar{k}$ où $(i_1, j_1), \dots (i_n, j_n)$ sont les arêtes de G

6 Notion de codage

- ▶ la définition de problème comme un langage de mots est abstraite, et utile en théorie de la calculabilité et complexité.
- ▶ bien souvent, on peut “oublier” le codage et travailler directement avec des représentations plus “concrètes”, par exemple une matrice d’adjacence pour un graphe, ou une liste de voisins.
- ▶ mais attention, le codage peut influencer la complexité, et il faudra parfois revenir à cette notion.
- ▶ par exemple, quelle est la complexité (en temps) de l'algorithme suivant :
 - ▶ ENTREE : $n \in \mathbb{N}$
 - ▶ POUR i de 2 à $\lceil \sqrt{n} \rceil$ FAIRE
 - ▶ SI $i|n$ ALORS RETOURNER 0
 - ▶ RETOURNER 1

6 Notion de codage

- ▶ la définition de problème comme un langage de mots est abstraite, et utile en théorie de la calculabilité et complexité.
- ▶ bien souvent, on peut “oublier” le codage et travailler directement avec des représentations plus “concrètes”, par exemple une matrice d’adjacence pour un graphe, ou une liste de voisins.
- ▶ mais attention, le codage peut influencer la complexité, et il faudra parfois revenir à cette notion.
- ▶ par exemple, quelle est la complexité (en temps) de l’algorithme suivant :
 - ▶ ENTREE : $n \in \mathbb{N}$
 - ▶ POUR i de 2 à $\lceil \sqrt{n} \rceil$ FAIRE
 - ▶ SI $i|n$ ALORS RETOURNER 0
 - ▶ RETOURNER 1
- ▶ cela dépend du codage de n

7 Problème d'optimisation

- ▶ un problème d'optimisation est un problème où l'on veut maximiser ou minimiser une certaine quantité
 - ▶ par exemple, trouver la longueur d'un plus court chemin dans un graphe, d'un sommet s à un sommet t .

7 Problème d'optimisation

- ▶ un problème d'optimisation est un problème où l'on veut maximiser ou minimiser une certaine quantité
 - ▶ par exemple, trouver la longueur d'un plus court chemin dans un graphe, d'un sommet s à un sommet t .
- ▶ on peut associer un problème de décision à un problème d'optimisation, en donnant une borne
 - ▶ par exemple, est-ce qu'il existe un chemin de longueur au plus k de s à t ?

7 Problème d'optimisation

- ▶ un problème d'optimisation est un problème où l'on veut maximiser ou minimiser une certaine quantité
 - ▶ par exemple, trouver la longueur d'un plus court chemin dans un graphe, d'un sommet s à un sommet t .
- ▶ on peut associer un problème de décision à un problème d'optimisation, en donnant une borne
 - ▶ par exemple, est-ce qu'il existe un chemin de longueur au plus k de s à t ?
- ▶ si on sait résoudre le problème de décision associé à un problème d'optimisation, on peut parfois résoudre le problème d'optimisation
 - ▶ par exemple, pour trouver le plus court chemin de s à t dans un graphe à n sommets, on peut faire une recherche dichotomique.

8 Algorithme de décision

Définition

Un problème $P \subseteq \Sigma^*$ est décidé par un algorithme A si pour tout mot $u \in \Sigma^*$,

- ▶ A termine et retourne 1 si $u \in P$,
- ▶ A termine et retourne 0 si $u \notin P$.

Un problème est décidable s'il existe un algorithme qui le décide.

Par exemple, selon le codage des entiers adopté, l'algorithme vu précédemment décide PRIME_1 ou PRIME .

9 La classe \mathcal{P}

Définition

La classe \mathcal{P} est la classe des problèmes pouvant être décidés en temps polynomial. Plus précisément, un problème $P \subseteq \Sigma^*$ est dans \mathcal{P} si il existe un algorithme A et une constante k tel que pour tout mot u de longueur n ,

- ▶ A retourne 1 en temps $O(n^k)$ si $u \in P$
- ▶ A retourne 0 en temps $O(n^k)$ si $u \notin P$

9 La classe \mathcal{P}

Définition

La classe \mathcal{P} est la classe des problèmes pouvant être décidés en temps polynomial. Plus précisément, un problème $P \subseteq \Sigma^*$ est dans \mathcal{P} si il existe un algorithme A et une constante k tel que pour tout mot u de longueur n ,

- ▶ A retourne 1 en temps $O(n^k)$ si $u \in P$
- ▶ A retourne 0 en temps $O(n^k)$ si $u \notin P$

Exemples de problèmes dans \mathcal{P} :

- ▶ décider si un tableau est trié
- ▶ décider si un entier codé en unary est premier (facile)
- ▶ décider si un entier codé en binaire est premier (difficile, problème resté ouvert pendant longtemps)

9 La classe \mathcal{P}

Définition

La classe \mathcal{P} est la classe des problèmes pouvant être décidés en temps polynomial. Plus précisément, un problème $P \subseteq \Sigma^*$ est dans \mathcal{P} si il existe un algorithme A et une constante k tel que pour tout mot u de longueur n ,

- ▶ A retourne 1 en temps $O(n^k)$ si $u \in P$
- ▶ A retourne 0 en temps $O(n^k)$ si $u \notin P$

Exemples de problèmes dans \mathcal{P} :

- ▶ décider si un tableau est trié
- ▶ décider si un entier codé en unary est premier (facile)
- ▶ décider si un entier codé en binaire est premier (difficile, problème resté ouvert pendant longtemps)

Les notions d'algorithme et leur complexité peuvent être définies formellement par le modèle des machines de Turing.

10 Algorithme de vérification

- ▶ informellement, un algorithme de vérification est un algorithme qui, étant donnée une solution “candidate” à un problème de décision, décide si oui ou non cette solution est valide.
- ▶ par exemple, pour le coloriage de graphe avec au plus k couleurs, une solution candidate serait un coloriage c , et un algorithme de vérification devrait vérifier que (1) chaque sommet est colorié par c , (2) deux sommets voisins ne sont pas coloriés de la même couleur par c .

10 Algorithme de vérification

- ▶ informellement, un algorithme de vérification est un algorithme qui, étant donnée une solution “candidate” à un problème de décision, décide si oui ou non cette solution est valide.
- ▶ par exemple, pour le coloriage de graphe avec au plus k couleurs, une solution candidate serait un coloriage c , et un algorithme de vérification devrait vérifier que (1) chaque sommet est colorié par c , (2) deux sommets voisins ne sont pas coloriés de la même couleur par c .

Définition

Un algorithme de vérification pour un problème $P \subseteq \Sigma^*$ est un algorithme de décision¹ A prenant deux mots en argument, qui termine pour toute entrée, tel que

$$P = \{u \in \Sigma^* \mid \exists v \in \Sigma^*, A(u, v) = 1\}$$

Lorsque $A(u, v) = 1$, v est appelé un certificat pour u .

1. qui retourne 1 ou 0

11 La classe NP

La classe NP est la classe des problèmes pouvant être vérifiés en temps polynomial.

Formellement :

Définition

Un problème P est dans NP si il existe un algorithme de vérification A de complexité polynomiale en temps, et une constante k , tels que pour tout entrée u , les deux affirmations suivantes sont équivalentes :

1. $u \in P$
2. il existe un certificat v de longueur polynomiale dans u ($|v| = O(|u|^k)$) tel que $A(u, v) = 1$.

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ?

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ?

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ? (certificat = partition)

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ? (certificat = partition)
- ▶ *bin packing* : peut-on ranger n objets de volume v_i dans k boîtes de volumes C ?

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ? (certificat = partition)
- ▶ *bin packing* : peut-on ranger n objets de volume v_i dans k boîtes de volumes C ? (certificat = rangement)

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ? (certificat = partition)
- ▶ *bin packing* : peut-on ranger n objets de volume v_i dans k boîtes de volumes C ? (certificat = rangement)
- ▶ tous les problèmes de la classe $\mathcal{P} (\mathcal{P} \subseteq \mathcal{NP})$

12 Exemples de problèmes dans \mathcal{NP}

- ▶ décider qu'un ensemble de clauses est satisfaisable (certificat = valuation)
- ▶ *voyageur de commerce* : étant donnés n villes, les distances entre les villes, et un entier d , on voudrait savoir s'il existe un cycle de longueur $\leq d$ qui passe par chaque ville une et une seule fois (certificat = un cycle)
c de longueur au plus d.
- ▶ *coloriage de graphes* : peut-on colorier un graphe avec moins de k couleurs sans que deux sommets aient la même couleur ? (certificat = coloriage)
- ▶ *2-partition* : peut-on partitionner un ensemble de n entiers en deux tas égaux ? (certificat = partition)
- ▶ *bin packing* : peut-on ranger n objets de volume v_i dans k boîtes de volumes C ? (certificat = rangement)
- ▶ tous les problèmes de la classe $\mathcal{P} (\mathcal{P} \subseteq \mathcal{NP})$: on peut prendre l'algorithme A qui décide le problème en temps polynomial et le transformer en un algorithme de vérification qui ignore le certificat.

13 $\mathcal{NP} \subseteq \text{ExpTime}$

- ▶ tout problème de \mathcal{NP} peut être décidé par un algorithme de complexité exponentielle en temps. Pourquoi ?

13 $\mathcal{NP} \subseteq \text{ExpTime}$

- ▶ tout problème de \mathcal{NP} peut être décidé par un algorithme de complexité exponentielle en temps. Pourquoi ?
- ▶ étant donné un mot u en entrée, il suffit d'énumérer tous les certificats de longueur au plus $a \cdot |u|^k$ (pour la constante a de la notation $O()$) et d'appeler l'algorithme de vérification. Exemple ?

13 $\mathcal{NP} \subseteq \text{ExpTime}$

- ▶ tout problème de \mathcal{NP} peut être décidé par un algorithme de complexité exponentielle en temps. Pourquoi ?
- ▶ étant donné un mot u en entrée, il suffit d'énumérer tous les certificats de longueur au plus $a.|u|^k$ (pour la constante a de la notation $O()$) et d'appeler l'algorithme de vérification. Exemple ?
- ▶ pour SAT : énumérer toutes les interprétations possibles et les tester
- ▶ pour le coloriage : énumérer tous les colorages possibles et les tester

14 Conjecture

Conjecture fameuse en théorie de la complexité :

$$\mathcal{P} \neq \mathcal{NP}$$

Autrement dit, on ne sait pas encore démontrer que

- ▶ vérifier une solution candidate en temps polynomial (dans la taille de l'entrée du problème),

est plus puissant que

- ▶ trouver une solution en temps polynomial.

Temps non-déterministe polynomial

- ▶ une définition équivalente de NP peut être donnée
- ▶ c'est la classe des problèmes pouvant être décidés en temps polynomial par un algorithme non-déterministe
- ▶ cet algorithme doit s'arrêter après un nombre d'étapes polynomial
- ▶ il a le droit de faire des choix aléatoires (tirer à pile ou face)
- ▶ si la réponse au problème est OUI, alors il doit exister une exécution de l'algorithme (i.e. une suite de tirages) après laquelle l'algorithme répondra OUI

Temps non-déterministe polynomial

- ▶ une définition équivalente de NP peut être donnée
- ▶ c'est la classe des problèmes pouvant être décidés en temps polynomial par un algorithme non-déterministe
- ▶ cet algorithme doit s'arrêter après un nombre d'étapes polynomial
- ▶ il a le droit de faire des choix aléatoires (tirer à pile ou face)
- ▶ si la réponse au problème est OUI, alors il doit exister une exécution de l'algorithme (i.e. une suite de tirages) après laquelle l'algorithme répondra OUI
- ▶ **Exemple : SAT** choisir aléatoirement une valeur de vérité pour chaque variable, et vérifier en temps linéaire qu'elles satisfont la formule.

Temps non-déterministe polynomial

- ▶ une définition équivalente de NP peut être donnée
- ▶ c'est la classe des problèmes pouvant être décidés en temps polynomial par un algorithme non-déterministe
- ▶ cet algorithme doit s'arrêter après un nombre d'étapes polynomial
- ▶ il a le droit de faire des choix aléatoires (tirer à pile ou face)
- ▶ si la réponse au problème est OUI, alors il doit exister une exécution de l'algorithme (i.e. une suite de tirages) après laquelle l'algorithme répondra OUI
- ▶ **Exemple : SAT** choisir aléatoirement une valeur de vérité pour chaque variable, et vérifier en temps linéaire qu'elles satisfont la formule.
- ▶ Pourquoi cette définition est équivalente ?

16 Réductions, NP -dureté et NP -complétude

Certains problèmes contiennent toute la difficulté de n'importe quel problème de la classe NP . On les appelle les problèmes NP -durs ou NP -difficiles. Ceci se définit par la notion de réduction :

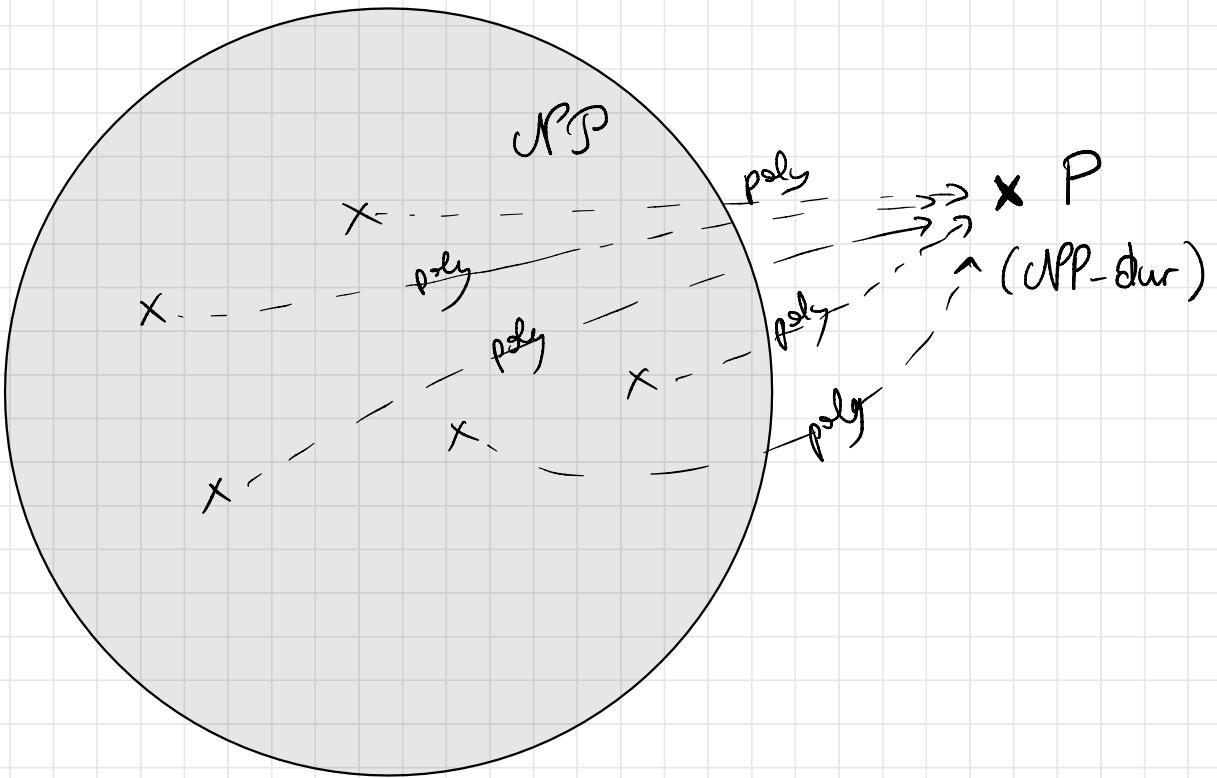
Définition

Un problème de décision P est NP -dur si tout problème P' de NP se réduit à P en temps polynomial, i.e. qu'il existe un algorithme T de complexité polynomial en temps, qui transforme tout mot u' en un mot $T(u')$ tel que $u' \in P'$ si et seulement si $T(u') \in P$.

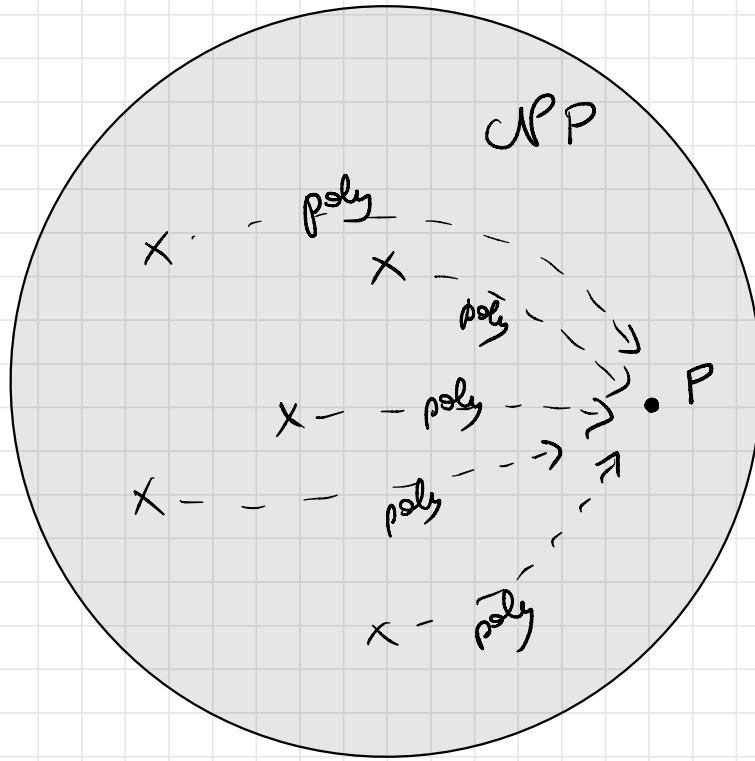
Un problème qui est à la fois dans NP et NP -dur est dit NP -complet. Ce sont les problèmes les "plus difficiles" de NP .

EXEMPLE DÉJÀ VU DANS LE COURS "SAT"



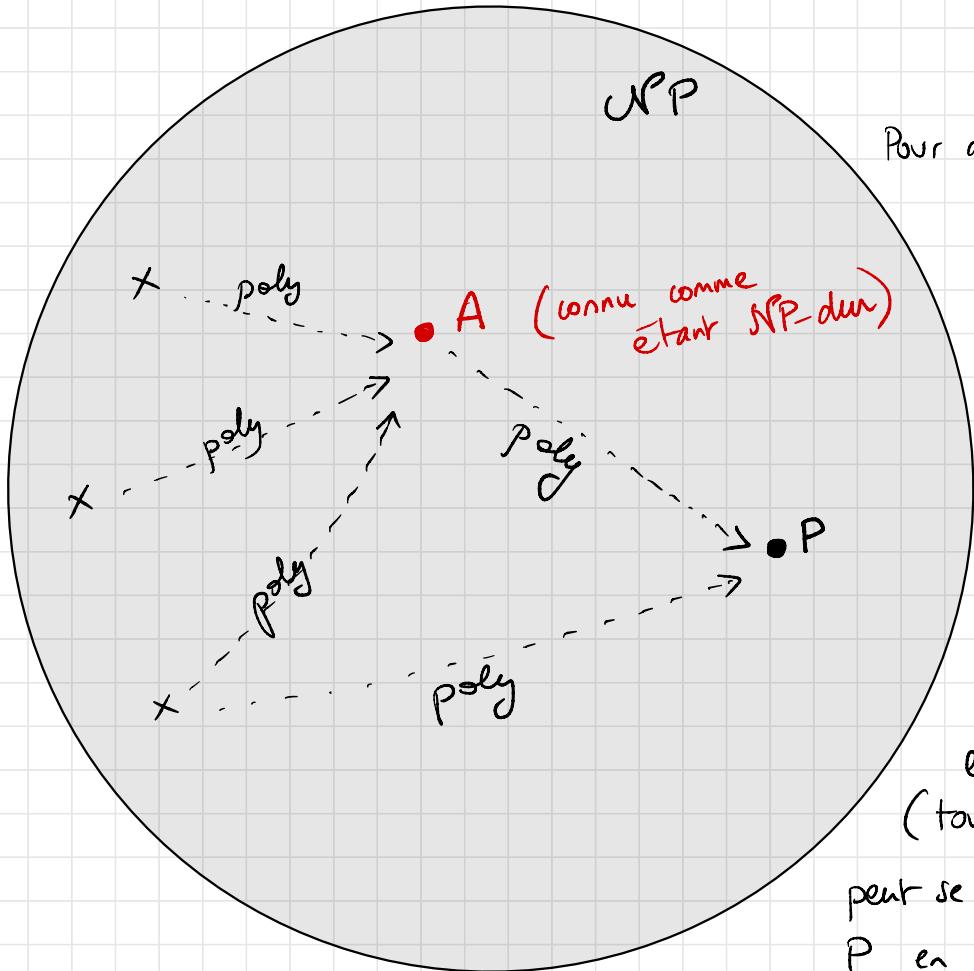


Un problème P peut être NP-dur sans être dans NP.



P : NP -complet

S'il est en plus dans NP , on dit qu'il est NP -complet.



Pour démontrer que

P est

NP-complet :

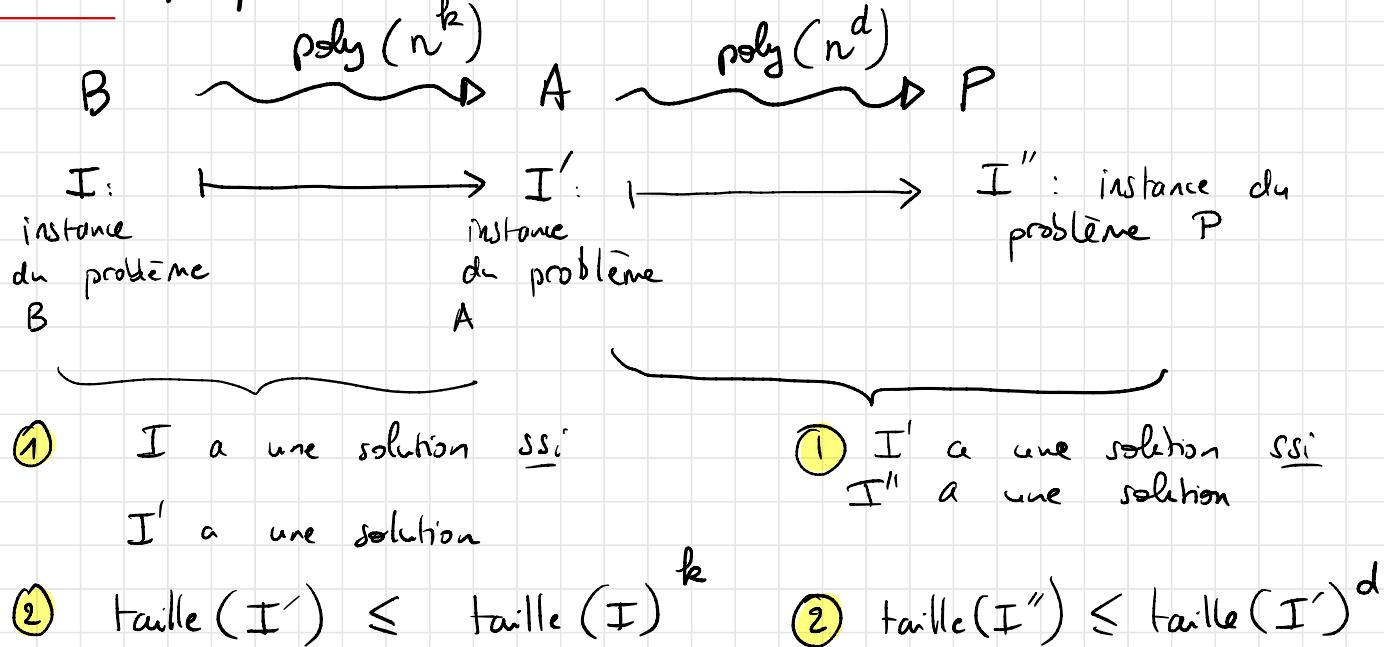
① démontre qu'il est dans NP

② démontrer qu'il est NP-dur.

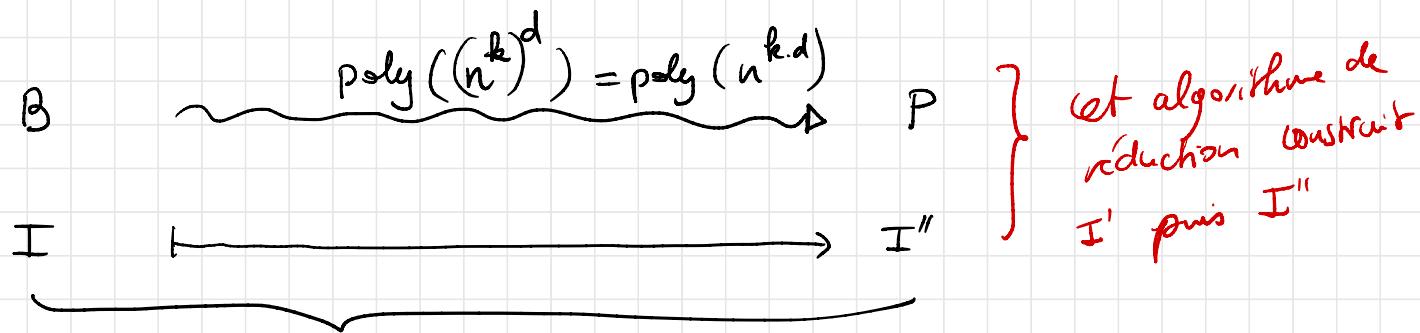
(tout problème de NP peut se réduire au problème P en temps polynomial)

Si on sait que A est NP-dur et que A se réduit à P, alors tout problème B ∈ clP se réduit à P en temps polynomial, par composition des réductions.

DETAILS



Par composition des réductions on obtient une réduction polynomiale de B à P :



- ① I a une solution ssi I'' a une solution
- ② $\text{taille}(I'') \leq \text{taille}(I)^{k \cdot d}$

17 Problèmes \mathcal{NP} -complets

- ▶ Pour montrer qu'un problème est \mathcal{NP} -complet, on montre qu'il est dans \mathcal{NP} et qu'il est \mathcal{NP} -dur. Pour montrer qu'il est \mathcal{NP} -dur, en général on part d'un problème connu comme étant \mathcal{NP} -dur, qu'on réduit dans notre problème en temps polynomial.

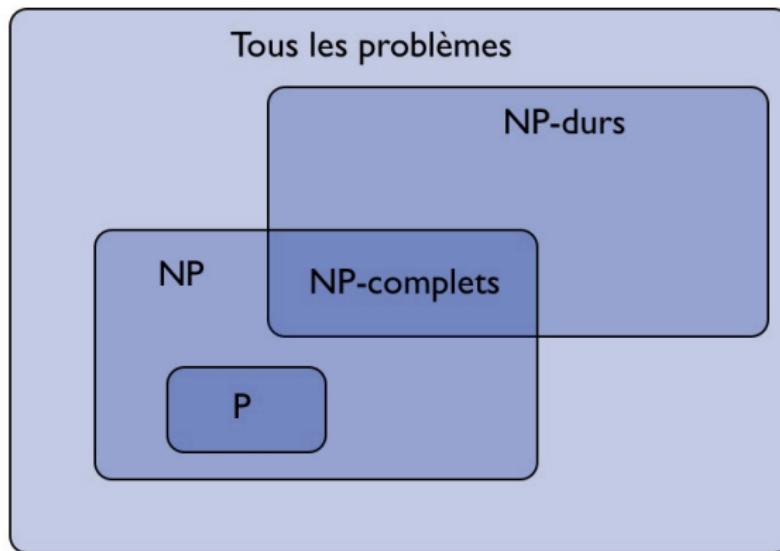
17 Problèmes \mathcal{NP} -complets

- ▶ Pour montrer qu'un problème est \mathcal{NP} -complet, on montre qu'il est dans \mathcal{NP} et qu'il est \mathcal{NP} -dur. Pour montrer qu'il est \mathcal{NP} -dur, en général on part d'un problème connu comme étant \mathcal{NP} -dur, qu'on réduit dans notre problème en temps polynomial.
- ▶ Il fallait bien un premier problème \mathcal{NP} -dur !

17 Problèmes \mathcal{NP} -complets

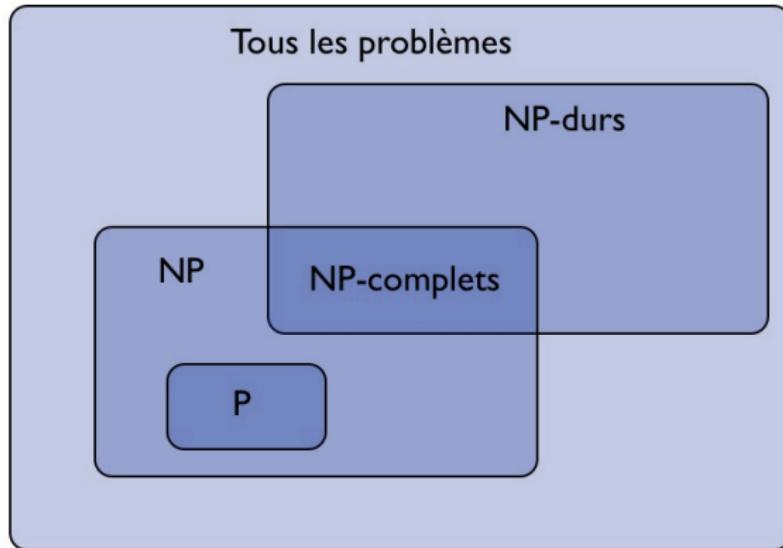
- ▶ Pour montrer qu'un problème est \mathcal{NP} -complet, on montre qu'il est dans \mathcal{NP} et qu'il est \mathcal{NP} -dur. Pour montrer qu'il est \mathcal{NP} -dur, en général on part d'un problème connu comme étant \mathcal{NP} -dur, qu'on réduit dans notre problème en temps polynomial.
- ▶ Il fallait bien un premier problème \mathcal{NP} -dur !
- ▶ **Théorème de Cook (1971)** : SAT est \mathcal{NP} -complet.

18 Vue d'ensemble



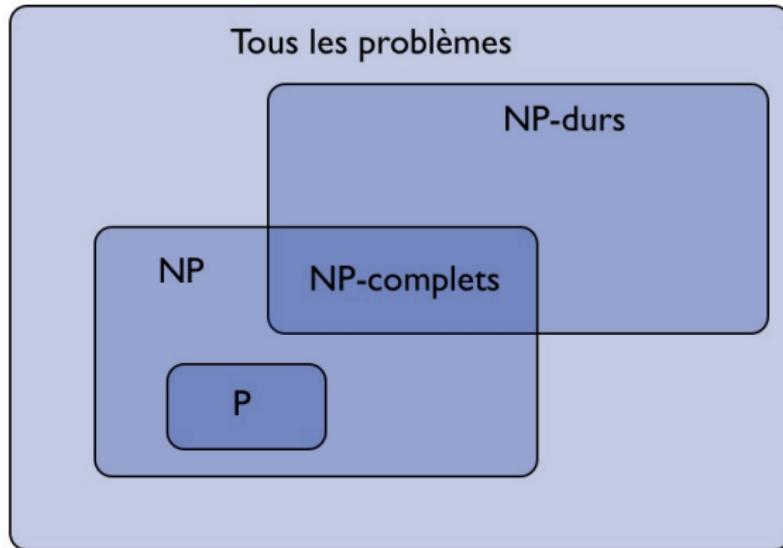
18 Vue d'ensemble

Si $P \neq NP$ et qu'un problème A est NP -complet, que peut-on en déduire sur la complexité du problème A (algorithmique en temps) ?



18 Vue d'ensemble

Si $P \neq NP$ et qu'un problème A est NP -complet, que peut-on en déduire sur la complexité du problème A (algorithmique en temps) ?



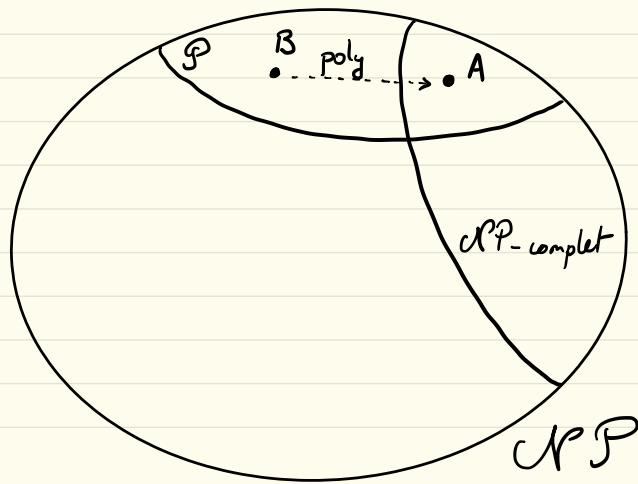
On peut en déduire que A ne peut pas être résolu en temps polynomial.

Théorème: Si $\mathcal{P} \neq \mathcal{NP}$ et un problème A est \mathcal{NP} -complet, alors $A \notin \mathcal{P}$.

On va supposer que $A \in \mathcal{P}$ et en déduire la contradiction $\mathcal{P} = \mathcal{NP}$.

On sait que $\mathcal{P} \subseteq \mathcal{NP}$, il nous reste à démontrer $\mathcal{NP} \subseteq \mathcal{P}$.

Soit B un problème de \mathcal{NP} , montrons qu'il est dans \mathcal{P} .



Comme A est \mathcal{NP} -complet, B se réduit à A en temps polynomial.

Comme $A \in \mathcal{P}$, nous avons donc un algorithme en temps polynomial pour résoudre B :

Entrée: Instance I de B

- ① Transformer I en une instance I' de A $O(n^c)$
- ② Résoudre I' (avec l'algo pour A) $O(n^{c'})$

Supposons que la réduction se fasse en temps $O(n^c)$ pour c une constante.

Supposons que A se résout en temps $O(n^c)$ pour c' un constanté.

Alors B peut se résoudre, avec l'algo précédent, en temps $O(n^{cc'})$:

- Puisque l'étape ① est en temps $O(n^c)$ où n est la taille de I , la taille de I' est en $O(n^c)$
- L'étape ② est appliquée sur une entrée de taille $O(n^c)$, donc elle prend $O((n^c)^{c'}) = O(n^{cc'})$



Consequence: Si vous prouvez qu'un problème A est dans P et est NP-complet, soit vous avez démontré $P=NP$ (peu probable), soit votre preuve est fausse ($A \notin P$ ou A n'est pas NP-complet).

Autre conséquence: Si vous ne trouvez pas d'algorithme en temps polynomial pour votre problème, alors peut-être qu'il est NP-complet, et dans ce cas vous avez peu de chances d'en trouver un (soit vous avez résolu la conjecture soit il n'y en a pas).

19 Exemples de Problèmes \mathcal{NP} -complets

- ▶ SAT
- ▶ 3-SAT (exactement 3 variables par clauses)
- ▶ voyageur de commerce (est-ce qu'il existe un cycle de longueur au plus d qui passe par chaque sommet exactement une fois ?)
- ▶ coloriage de graphes (même avec 3 couleurs)
- ▶ bin-packing
- ▶ couverture d'arête : étant donné un graphe et un entier k , peut-on trouver un sous-ensemble S de sommets tels que chaque arête a au moins une de ses extrémités dans S ?
- ▶ ...
- ▶ voir <http://www.nada.kth.se/~viggo/problemlist/compendium.html>

20 Voyageur de Commerce

- ▶ TSP en anglais (Traveling Salesman Problem)
- ▶ différentes applications : <http://www.tsp.gatech.edu/>
- ▶ 500\$ à qui trouve le circuit de distance minimale traversant les 115475 villes des USA
- ▶ record : tour de la Suède, 24 978 villes



21 Réduction de SAT vers 3-SAT

Théorème

3SAT est NP-complet.

①

On doit démontrer que 3-SAT est dans CPP

On sait que SAT est dans CPP et

comme 3-SAT est un cas particulier de
SAT, on obtient que 3-SAT est dans CPP.

②

On doit démontrer que 3-SAT est NP-dur.

Pour cela, on va démontrer que SAT se réduit à 3-SAT en temps polynomial, car on sait que SAT est NP-dur.

Concrètement, on doit construire un algorithme R de complexité polynomiale en temps qui transforme toute instance S de SAT en une instance $R(S)$ de 3-SAT telle que

S est satisfaisable ssi $R(S)$ est satisfaisable.

SOLUTION

Sur un exemple

$$(x_0 \vee x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_5 \vee x_7)$$

$$(x_0 \vee x_1 \vee y) \wedge (\neg y \vee \neg x_2 \vee x_3 \vee x_4)$$

$$(\neg y \vee \neg x_2 \vee z) \wedge (\neg z \vee x_3 \vee x_4)$$

- Toute clause $C = l_1 \vee l_2 \vee \dots \vee l_k$ est transformée en une conjonction de clauses $T(C)$ comme suit :
 - si $k=1$: $T(C) = l_1 \vee l_1 \vee l_1$
 - si $k=2$: $T(C) = l_1 \vee l_2 \vee l_2$
 - si $k=3$: $T(C) = C$
 - si $k > 3$: $T(C) = (l_1 \vee l_2 \vee y) \wedge T(\neg y \vee l_3 \vee \dots \vee l_k)$

nouvelle variable
qui n'apparaît
nulle part.

on applique
récursivement la procédure,
sur $k-1$ littéraux.
- Pour transformer un ensemble de clauses $S = C_1 \wedge C_2 \wedge \dots \wedge C_n$, on applique T sur chaque clause en prenant soin de ne jamais introduire des variables qui ont déjà été utilisées.

Plus précisément, construire une instance de 3-SAT pour une conjonction de clauses S dont X est l'ensemble des variables utilisées, se fait en appelant $T(S, X)$ où T est l'algorithme récursif suivant :

$T(C_1 \wedge \dots \wedge C_n : \text{conjonction de clauses}, I : \text{ensemble de variables}) =$

Si $n > 1$ ALORS $\varphi := T(C_1, I)$; $\varphi_2 := T(C_2 \wedge \dots \wedge C_n, I \cup \underbrace{\text{var}(\varphi_1)}_{\substack{\text{I est l'ensemble des variables interdites lorsqu'on doit créer une nouvelle variable.}}})$

RETOURNER $\varphi_1 \wedge \varphi_2$

Si $n = 1$ et $C_1 = l_1$ ALORS RETOURNER $l_1 \vee l_1 \vee l_1$

Si $n = 1$ et $C_1 = l_1 \vee l_2$ ALORS RETOURNER $l_1 \vee l_2 \vee l_2$

Si $n = 1$ et $C_1 = l_1 \vee l_2 \vee l_3$ ALORS RETOURNER C_1

Si $n = 1$ et $C_1 = l_1 \vee \dots \vee l_k$, $k > 3$ ALORS

$y :=$ variable qui n'est pas dans I ; RETOURNER $(l_1 \vee l_2 \vee y) \wedge T(\neg y \vee l_3 \vee \dots \vee l_k, I \cup \{y\})$

Il reste à démontrer (preuve laissée en exercice) :

①

Que la réduction est correcte, i.e. que

S est satisfaisable si et seulement si $T(S, X)$ est satisfaisable

②

Que la réduction est en temps polynomial

Indices Pour ①, il faut faire une preuve par induction sur la structure de S et utiliser le fait que pour toute clause $C = l_1 \vee l_2 \vee \dots \vee l_k$ avec $k > 3$ et tout y qui n'est pas dans C , on a :

$V \models C$ si il existe $V' \models (l_1 \vee l_2 \vee y) \wedge (y \vee l_3 \vee \dots \vee l_k)$ avec $V'(x) = V(x)$ pour tout $x \in \text{var}(C)$.

En effet, si $V \models C$, alors il y a deux cas :

cas A : $V \models l_1 \vee l_2$, alors on prend $V'(y) = 0$ et $V'(x) = V(x)$ pour tout $x \in \text{var}(C)$

cas B : $V \models l_3 \vee \dots \vee l_k$, alors on prend $V'(y) = 1$ et $V'(x) = V(x)$ pour tout $x \in \text{var}(C)$

Pour ②, il faut remarquer que pour transformer une clause $\ell_1 \vee \dots \vee \ell_k$, on doit créer $k-3$ clauses supplémentaires et $k-3$ variables.

Exemple : $x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5 \vee \neg x_6$ devient

$$(x_1 \vee \neg x_2 \vee y_1) \quad 1$$

$$(\neg y_1 \vee x_3 \vee y_2) \quad 1$$

$$(\neg y_2 \vee x_4 \vee y_3) \quad 1$$

$$(\neg y_3 \vee \neg x_5 \vee \neg x_6)$$

Bin Packing

On veut ranger les objets suivants dans des sacs de capacité 10kg.

Objets	1	2	3	4	5	6	7
Poids	3	4	4	3	3	2	1

Peut-on réussir avec 3 sacs ?

Bin Packing

On veut ranger les objets suivants dans des sacs de capacité 10kg.

Objets	1	2	3	4	5	6	7
Poids	3	4	4	3	3	2	1

Peut-on réussir avec 3 sacs ? oui, on forme les sacs d'objets suivants :
 $\{1, 2\}$, $\{3, 4, 5\}$, $\{6, 7\}$.

Bin Packing

On veut ranger les objets suivants dans des sacs de capacité 10kg.

Objets	1	2	3	4	5	6	7
Poids	3	4	4	3	3	2	1

Peut-on réussir avec 3 sacs ? oui, on forme les sacs d'objets suivants :
 $\{1, 2\}$, $\{3, 4, 5\}$, $\{6, 7\}$.
et avec 2 sacs ?

Bin Packing

On veut ranger les objets suivants dans des sacs de capacité 10kg.

Objets	1	2	3	4	5	6	7
Poids	3	4	4	3	3	2	1

Peut-on réussir avec 3 sacs ? oui, on forme les sacs d'objets suivants : $\{1, 2\}$, $\{3, 4, 5\}$, $\{6, 7\}$.

et avec 2 sacs ?

oui : $\{1, 2, 4\}$, $\{3, 5, 6, 7\}$. Cette solution est optimale (on ne peut pas réussir avec un seul sac puisque la somme totale de tous les poids dépasse 10).

Bin Packing est \mathcal{NP} -complet

Supposons qu'on ait k sacs de capacité C .

- ▶ **BinPacking est dans \mathcal{NP}** : on peut vérifier en temps polynomial si une solution candidate, i.e. une assignation de chaque objet à un numéro de sac entre 1 et k , satisfait bien la contrainte de capacité.

Bin Packing est \mathcal{NP} -complet

Supposons qu'on ait k sacs de capacité C .

- ▶ **BinPacking est dans \mathcal{NP}** : on peut vérifier en temps polynomial si une solution candidate, i.e. une assignation de chaque objet à un numéro de sac entre 1 et k , satisfait bien la contrainte de capacité.
- ▶ **BinPacking est \mathcal{NP} -dur.** Nous allons le démontrer en faisant une réduction en temps polynomial du problème 2-partition, qui est un problème \mathcal{NP} -complet.

Bin Packing est \mathcal{NP} -complet

Supposons qu'on ait k sacs de capacité C .

- ▶ **BinPacking est dans \mathcal{NP}** : on peut vérifier en temps polynomial si une solution candidate, i.e. une assignation de chaque objet à un numéro de sac entre 1 et k , satisfait bien la contrainte de capacité.
- ▶ **BinPacking est \mathcal{NP} -dur.** Nous allons le démontrer en faisant une réduction en temps polynomial du problème 2-partition, qui est un problème \mathcal{NP} -complet.
 - ▶ **Instance de 2-partition** : n entiers c_1, \dots, c_n . On veut savoir s'il existe $S \subseteq \{1, \dots, n\}$ tel que $\sum_{i \in S} c_i = \sum_{i \notin S} c_i$. Soit $S = \sum_{i=1}^n c_i$. On peut supposer que S est paire (sinon il n'y trivialement pas de solution).
 - ▶ **Réduction vers BinPacking** : On prend $C = S/2$ et les objets $1, \dots, n$ de poids c_1, \dots, c_n , et $k = 2$. Il y a une solution à 2-partition si et seulement si il y a une à l'instance de BinPacking ainsi construite. De plus, on peut construire cette instance de BinPacking en temps polynomial (dans la taille de l'instance de 2-partitions). BinPacking est donc \mathcal{NP} -dur (même pour $k = 2$).

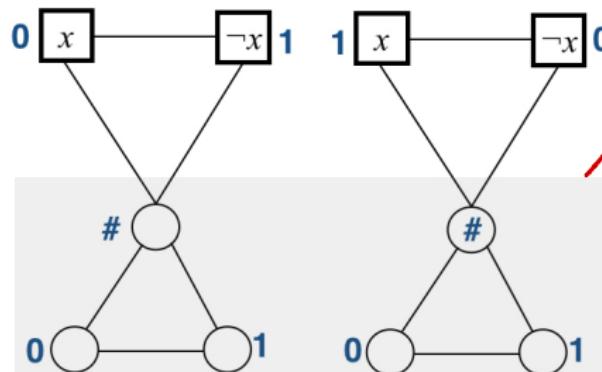
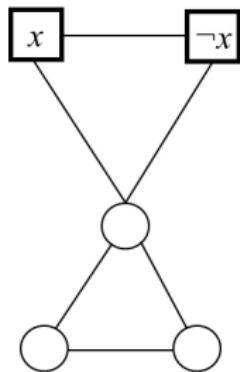
GRAPHCOLOR est \mathcal{NP} -complet

Nous allons démontrer que le problème de coloriage de graphe avec k couleurs est \mathcal{NP} -complet.

- ▶ le problème est dans \mathcal{NP} , en prenant pour solution candidate un coloriage, i.e. une fonction c des sommets vers $\{1, \dots, k\}$. Il est facile de vérifier en temps polynomial que cette solution est valide.
- ▶ pour montrer que GRAPHCOLOR est \mathcal{NP} -dur, nous allons réduire le problème 3SAT en temps polynomial dans GRAPHCOLOR
- ▶ on doit donc prouver l'existence d'un algorithme qui, à partir d'un ensemble S de clauses avec 3 littéraux, crée en temps polynomial un graph~~e~~ G et un entier k tel que S est satisfaisable ssi G est coloriable avec k couleurs.
- ▶ ici on va prendre $k = 3$, et prendre les couleurs 0, 1, et #.

25 Réduction de 3SAT vers GRAPHCOLOR

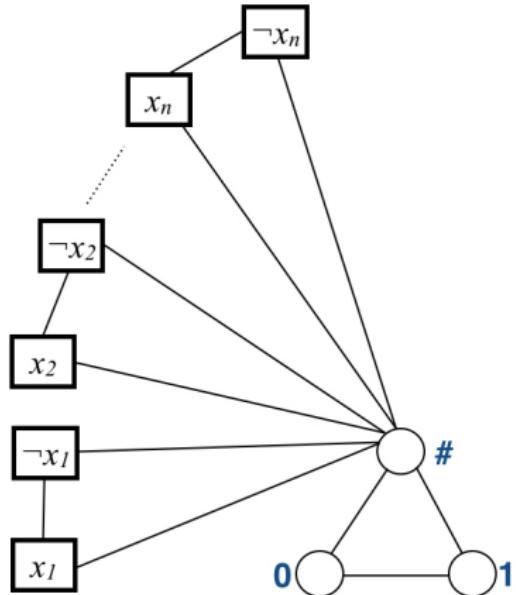
colorages possibles (modulo permutation)



grâce à permettre les couleurs, on peut supposer que ces trois noeuds sont toujours colorés comme cela (dans le cas où il existe un coloriage du graphe final)

- ▶ motif de base pour chaque proposition. Un littéral et sa négation ne peuvent pas être coloriés avec la même couleur
- ▶ Comme les colorages sont équivalents modulo permutation, on supposera que le noeud central est toujours colorié par #.

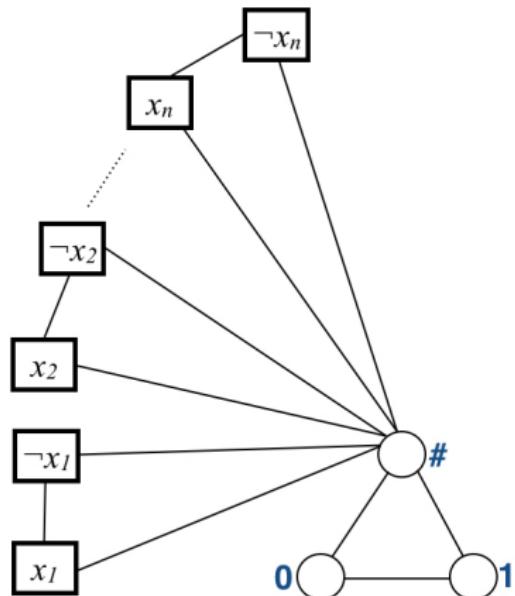
26 Réduction de 3SAT vers GRAPHCOLOR



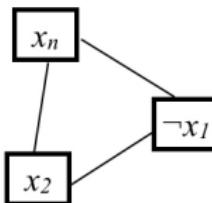
- ▶ on le fait pour les n propositions.

} on appellera cet ensemble de nœuds "le triangle", et on supposera donc qu'il est toujours colorié de cette manière dès lors qu'il existe un coloriage

27 Réduction de 3SAT vers GRAPHCOLOR

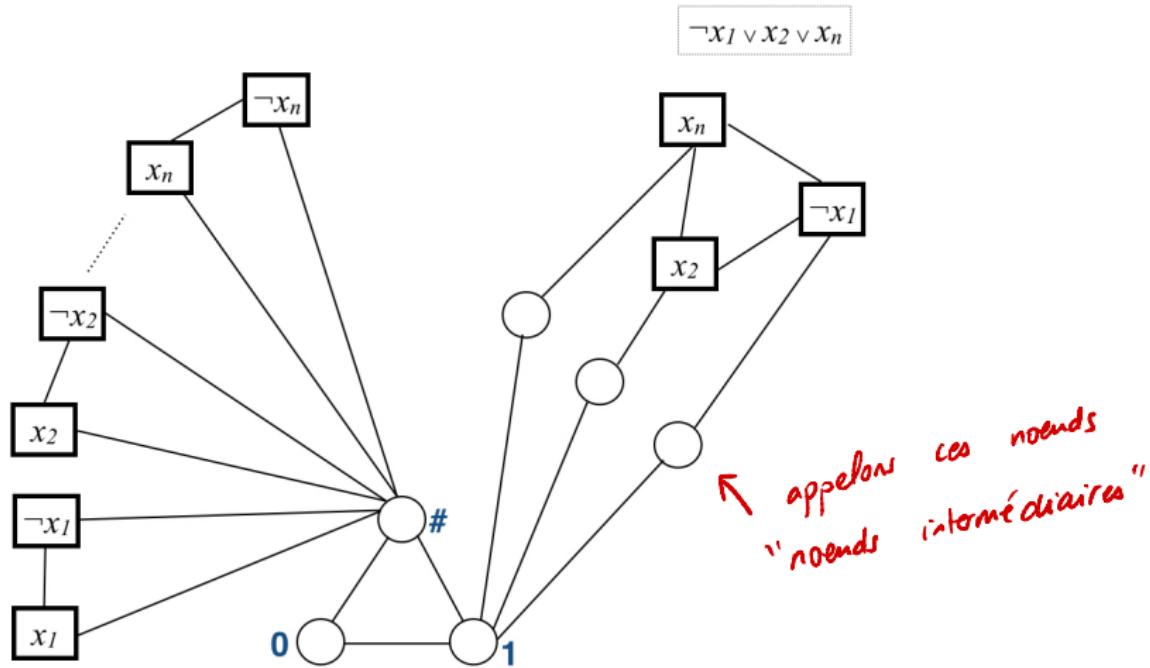


$$\neg x_1 \vee x_2 \vee x_3$$

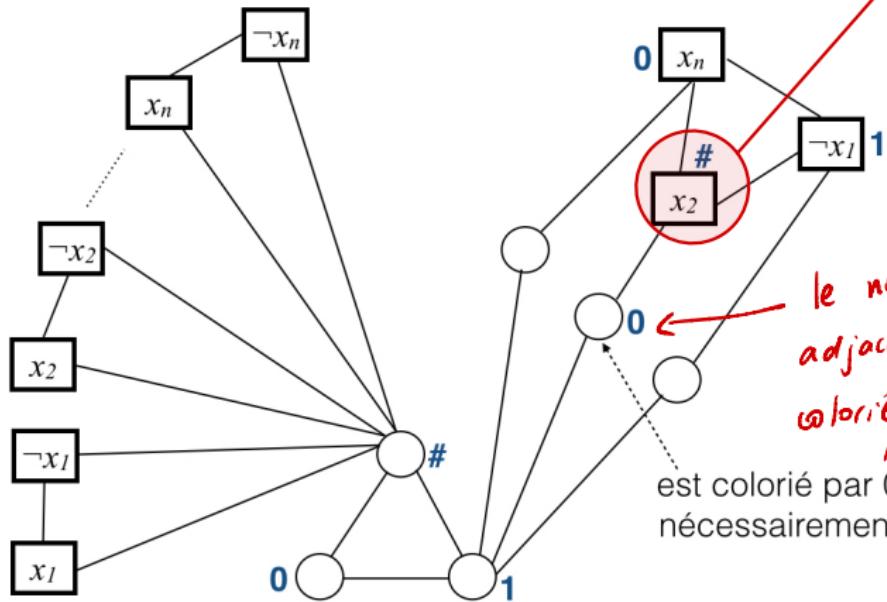


- ▶ encodage d'une clause

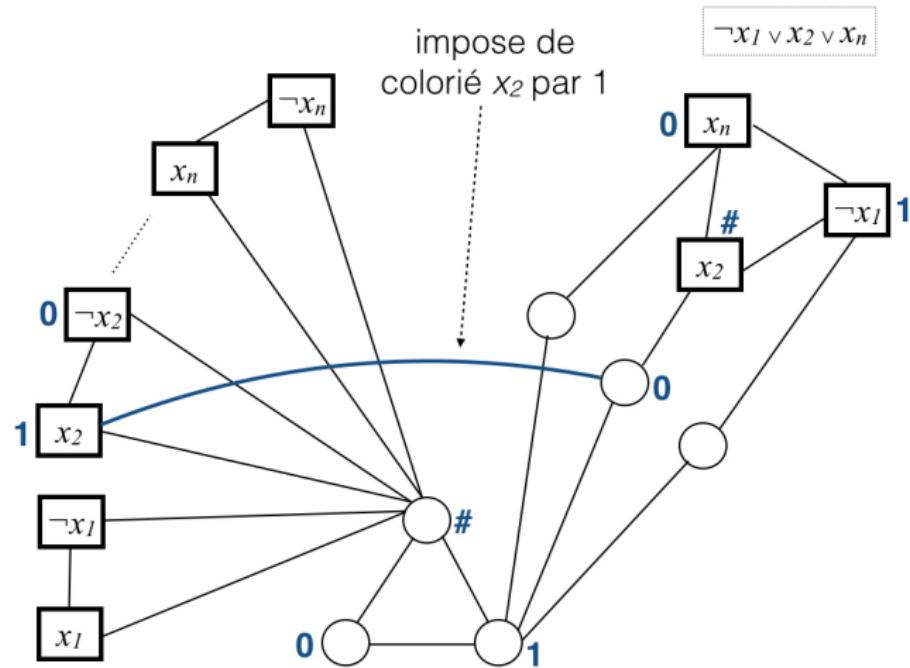
28 Réduction de 3SAT vers GRAPHCOLOR



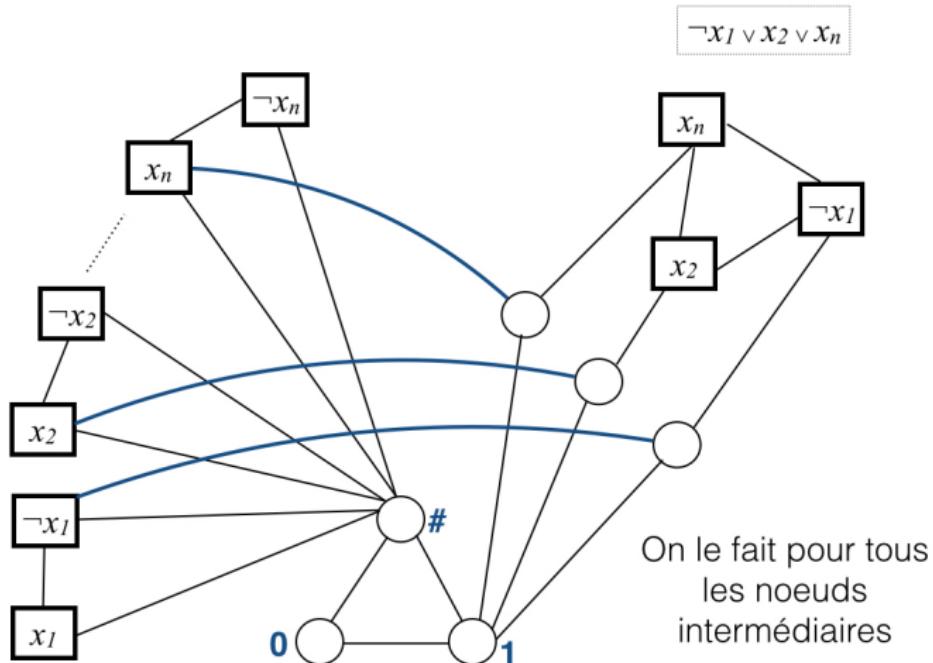
29 Réduction de 3SAT vers GRAPHCOLOR



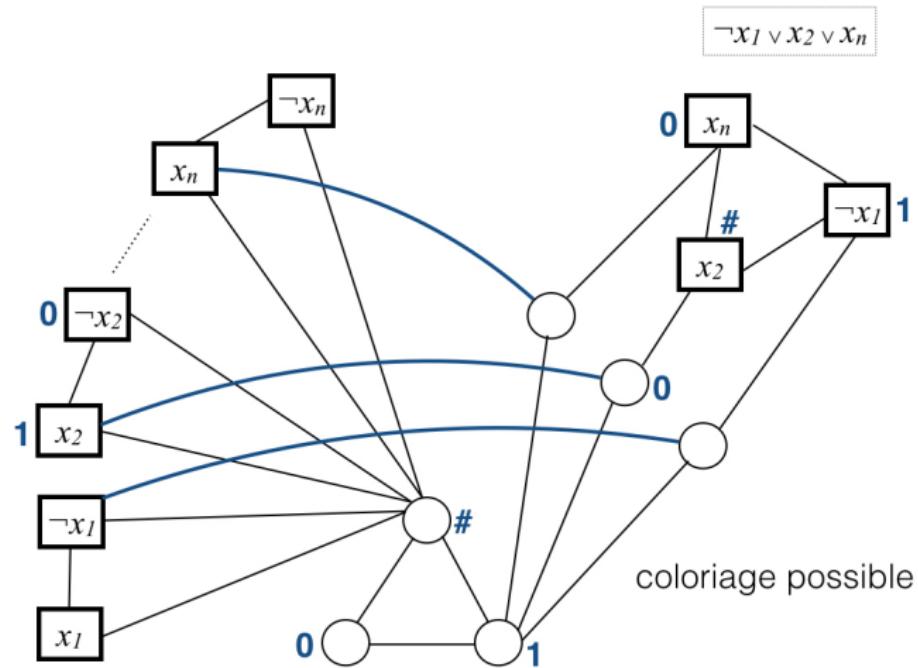
30 Réduction de 3SAT vers GRAPHCOLOR



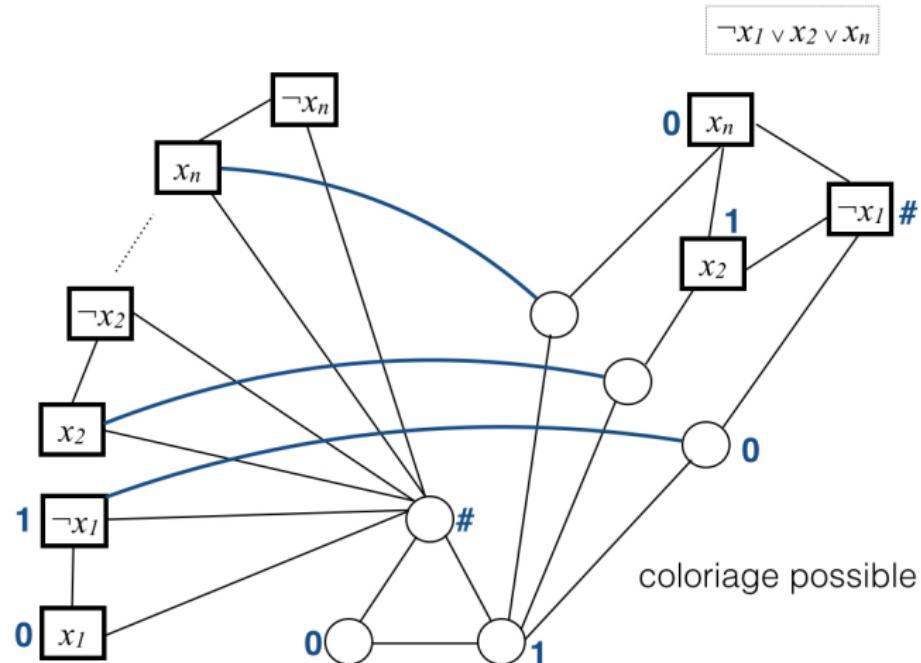
31 Réduction de 3SAT vers GRAPHCOLOR



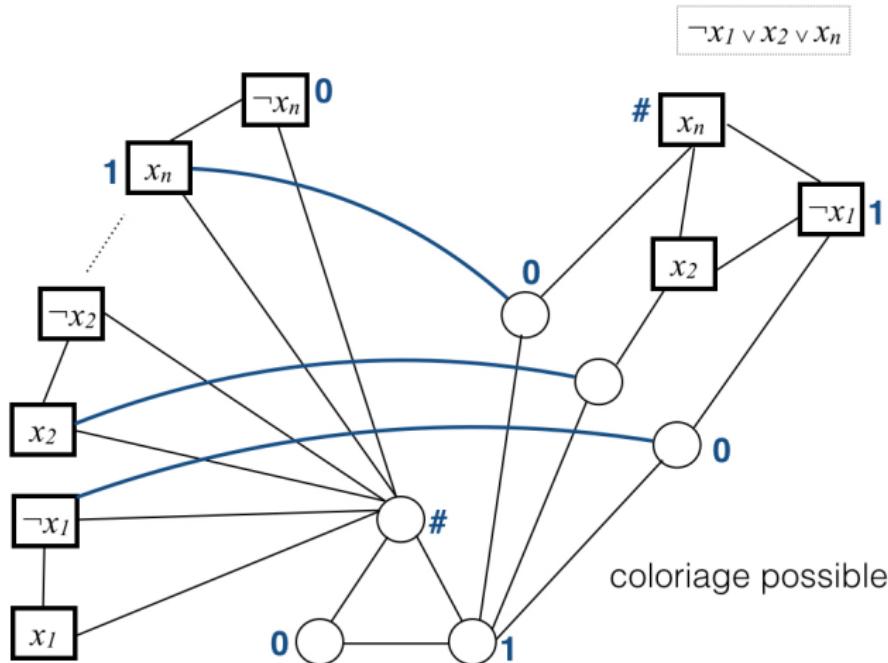
32 Réduction de 3SAT vers GRAPHCOLOR



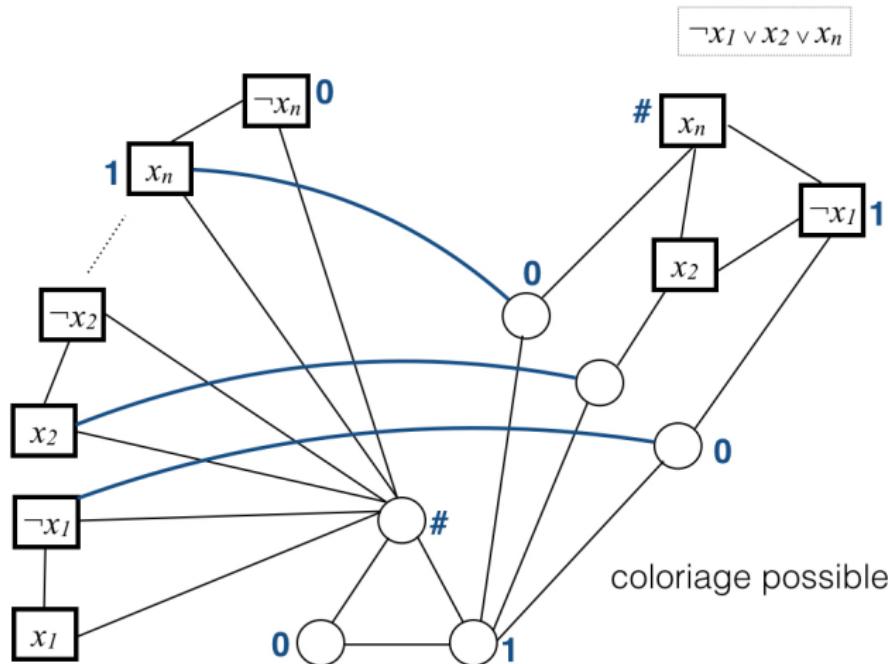
33 Réduction de 3SAT vers GRAPHCOLOR



34 Réduction de 3SAT vers GRAPHCOLOR



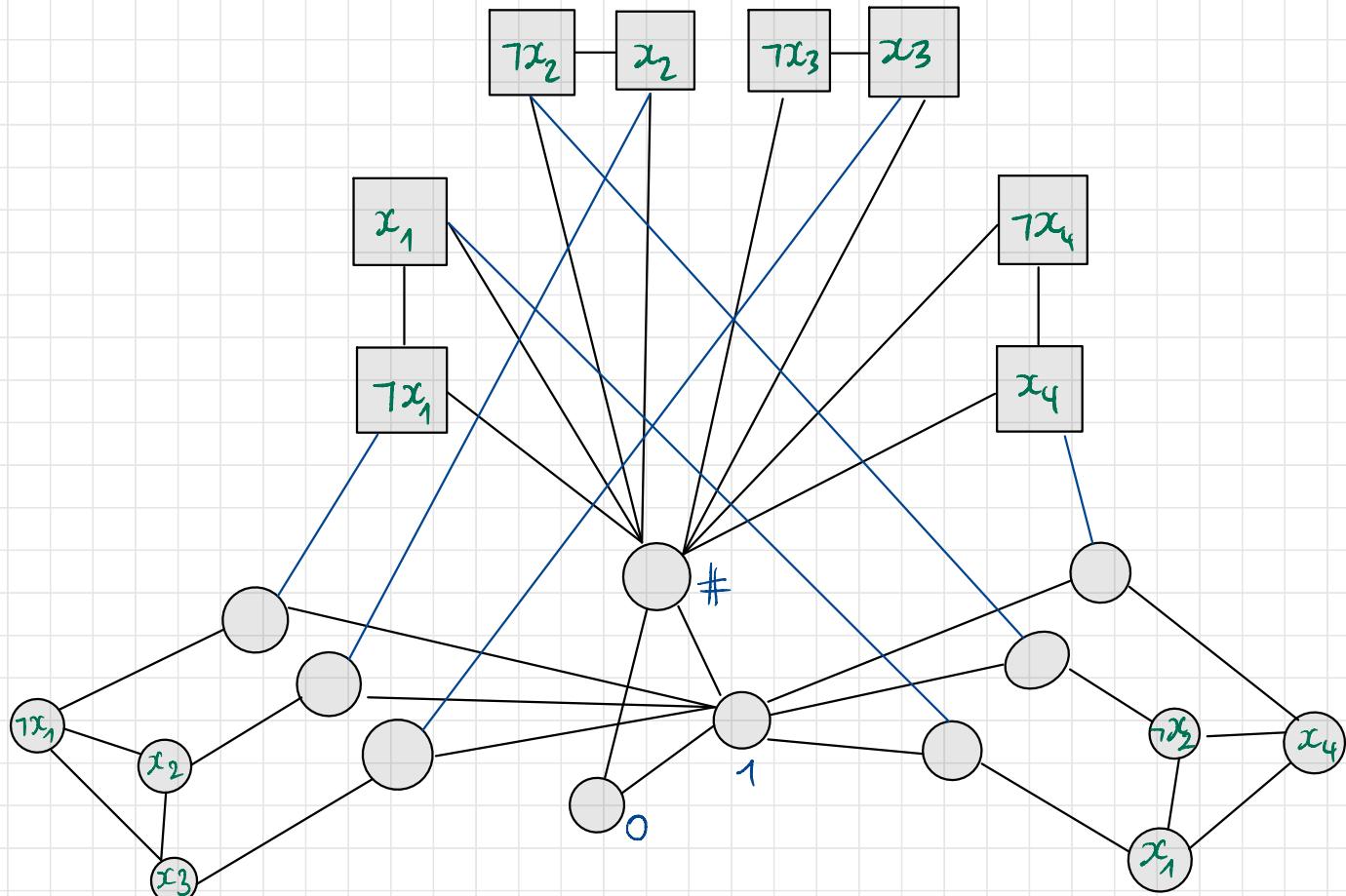
34 Réduction de 3SAT vers GRAPHCOLOR



- il faut le faire pour toutes les clauses

EXEMPLE

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

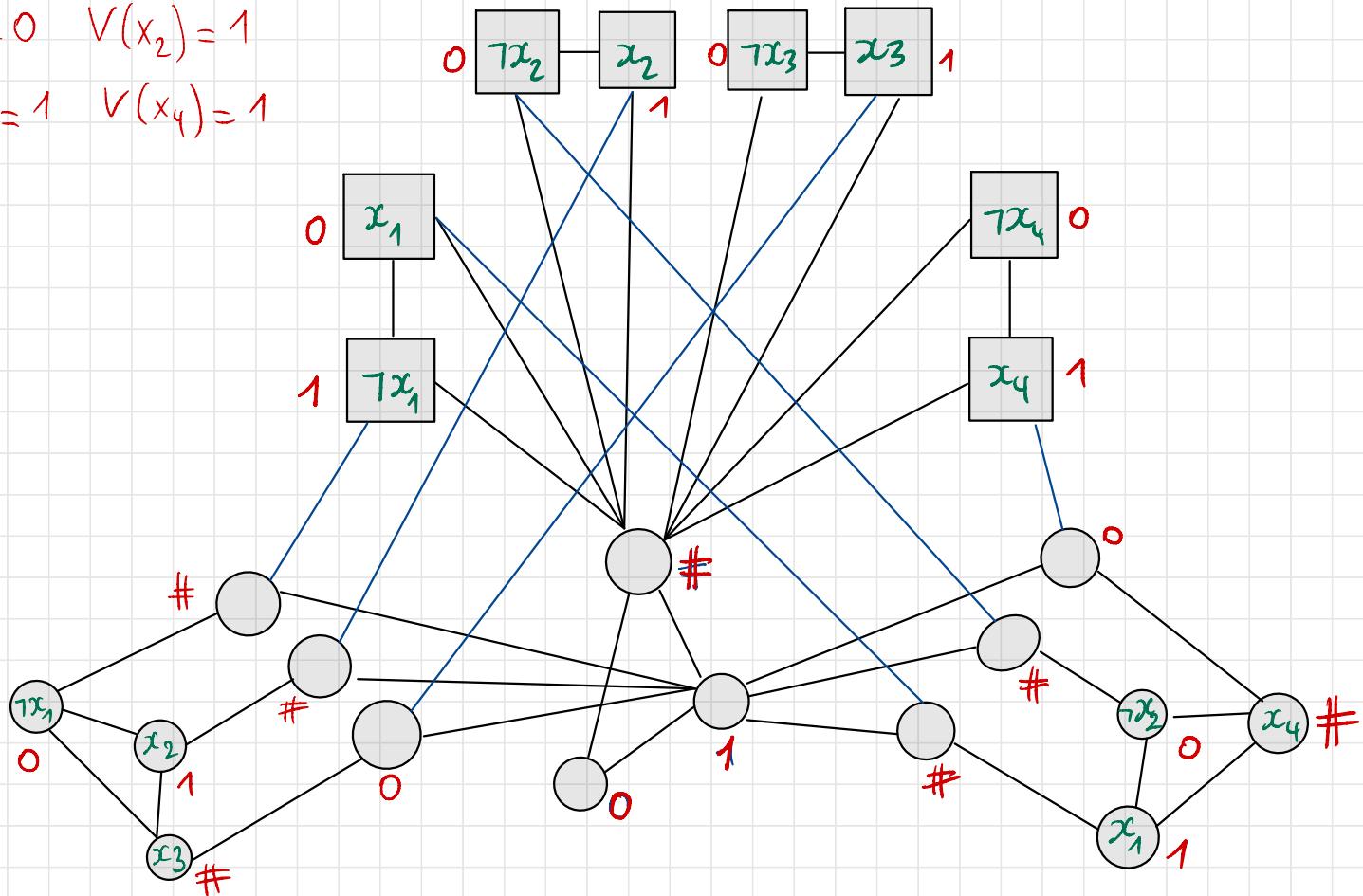


EXEMPLE

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

$$V(x_1) = 0 \quad V(x_2) = 1$$

$$V(x_3) = 1 \quad V(x_4) = 1$$



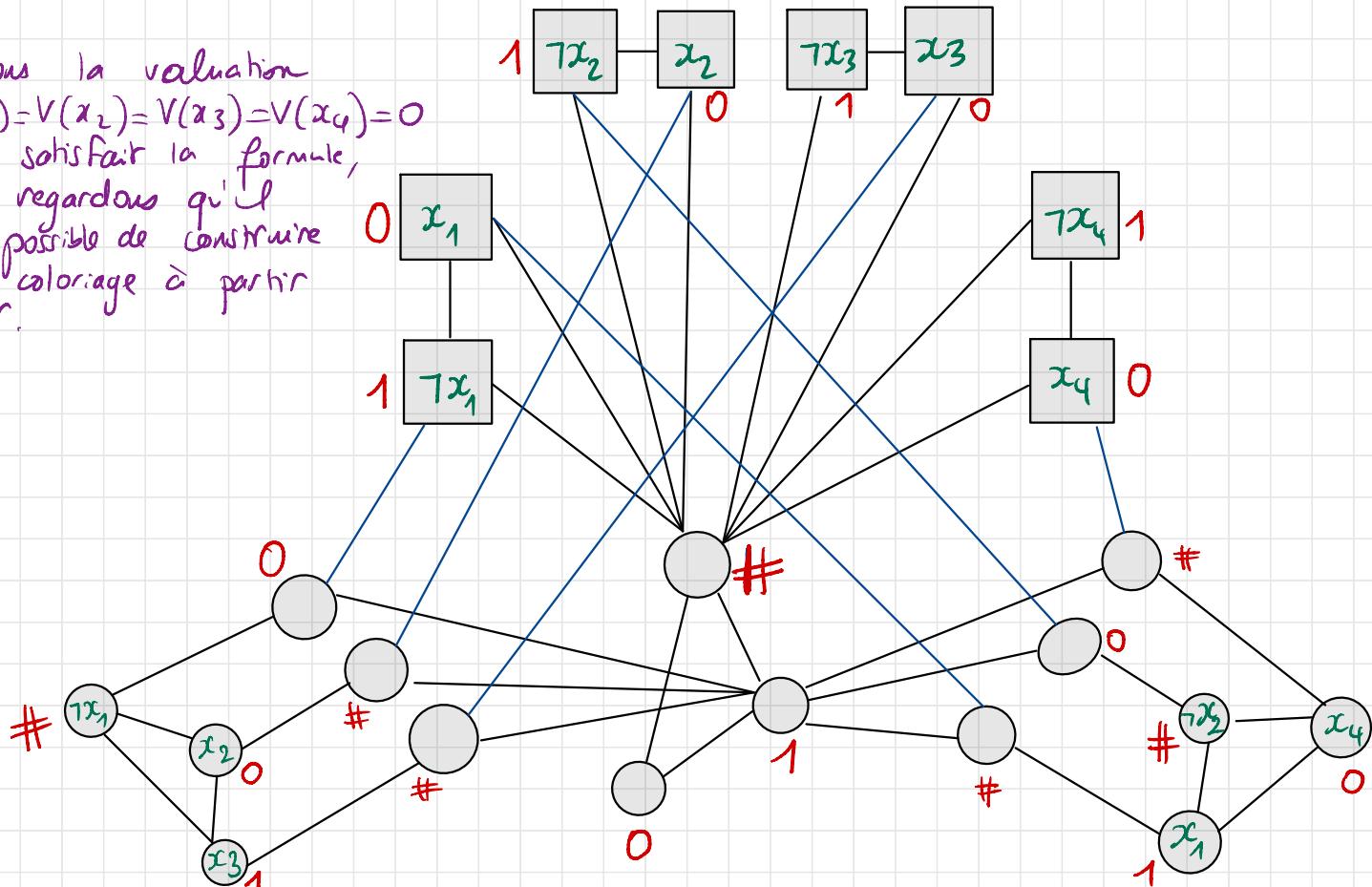
EXEMPLE

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

Prenons la valuation

$$V(x_1) = V(x_2) = V(x_3) = V(x_4) = 0$$

qui satisfait la formule,
et regardons qu'il
est possible de construire
un coloriage à partir
de \vee .



EXEMPLE

Donner une valuation qui correspond à ce coloriage.

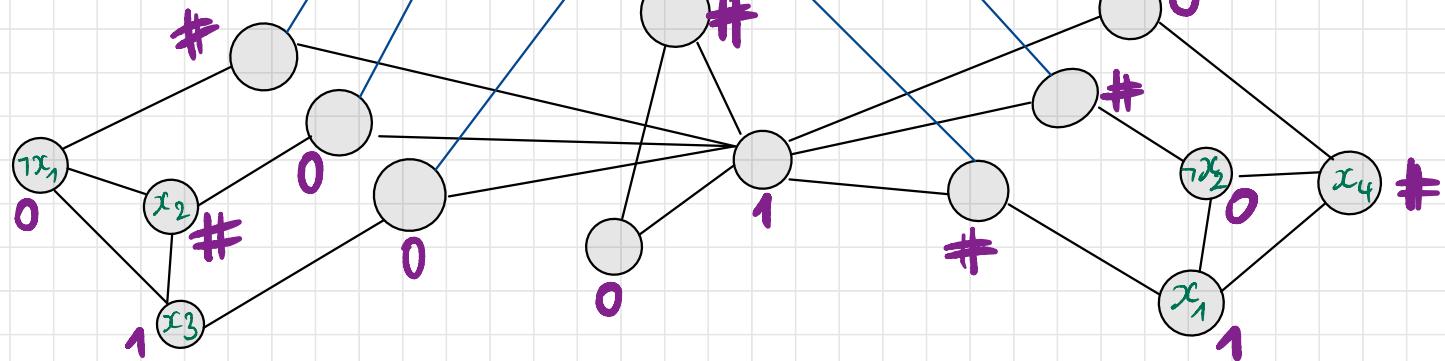
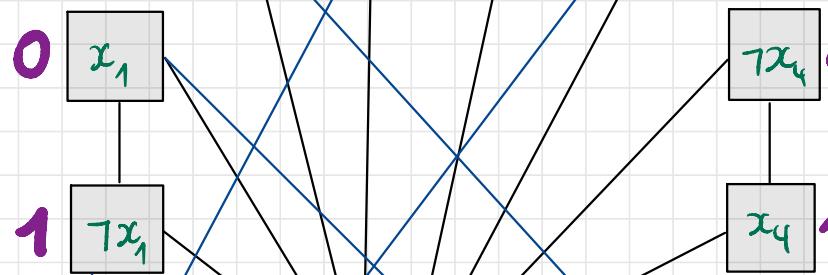
$$V(x_1) = 0$$

$$V(x_2) = 1$$

$$V(x_3) = 1$$

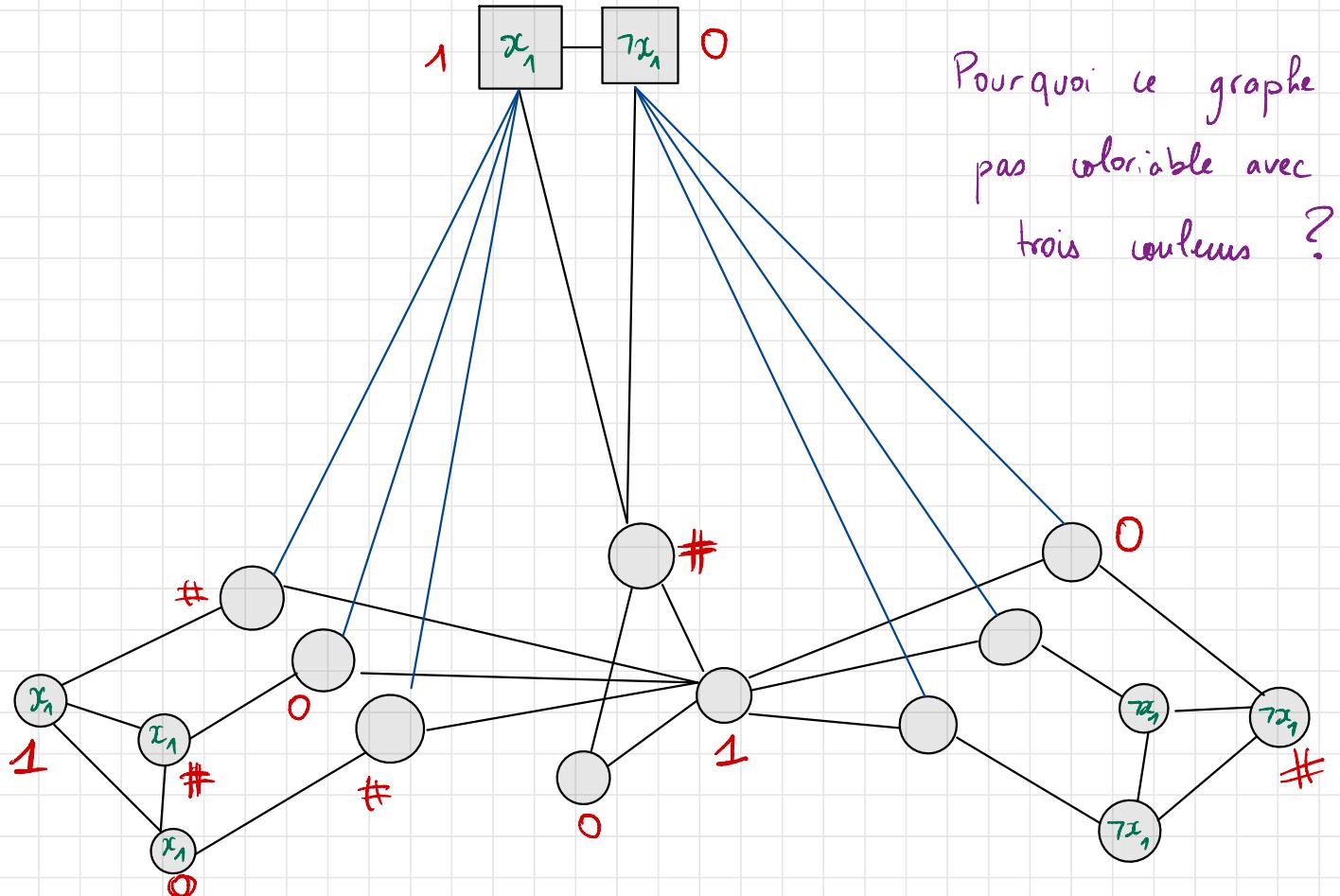
$$V(x_4) = 1$$

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$$



EXEMPLE AVEC UNE FORMULE INSATISFAISABLE

$$(x_1 \vee \bar{x}_1 \vee x_1) \wedge (\neg x_1 \vee \neg \bar{x}_1 \vee \neg x_1)$$



Pourquoi ce graphe n'est pas colorable avec trois couleurs ?

Formellement, la construction est la suivante.

À partir d'un ensemble de clauses $S = \{C_1, \dots, C_m\}$

formées sur un ensemble de variables $\{x_1, \dots, x_n\}$,

on construit un graphe G_S tel que

① G_S est colorable avec 3 couleurs si et seulement si S est satisfaisable

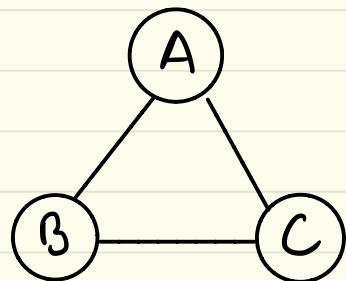
② La taille de G_S est polynomiale dans la taille de S

CONSTRUCTION DE G_S

Supposons que C_i est de la forme $\ell_1^{i^1} \vee \ell_2^{i^2} \vee \ell_3^{i^3}$ où
 $\ell_1^{i^1}, \ell_2^{i^2}, \ell_3^{i^3}$ sont des littéraux.

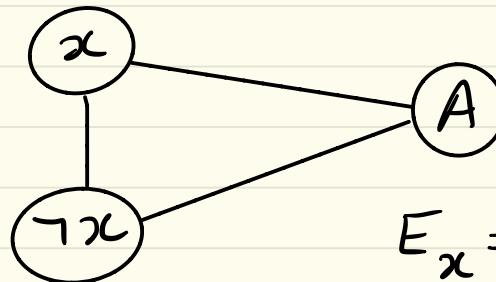
Ensemble de sommets: $V = \{x_1, \neg x_1, \dots, x_n, \neg x_n, \bar{\ell}_1^1, \bar{\ell}_2^1, \bar{\ell}_3^1, \dots, \bar{\ell}_1^m, \bar{\ell}_2^m, \bar{\ell}_3^m, A, B, C, \bar{\ell}_1^1, \bar{\ell}_2^1, \bar{\ell}_3^1, \dots, \bar{\ell}_1^m, \bar{\ell}_2^m, \bar{\ell}_3^m\}$

Ensemble d'arêtes:



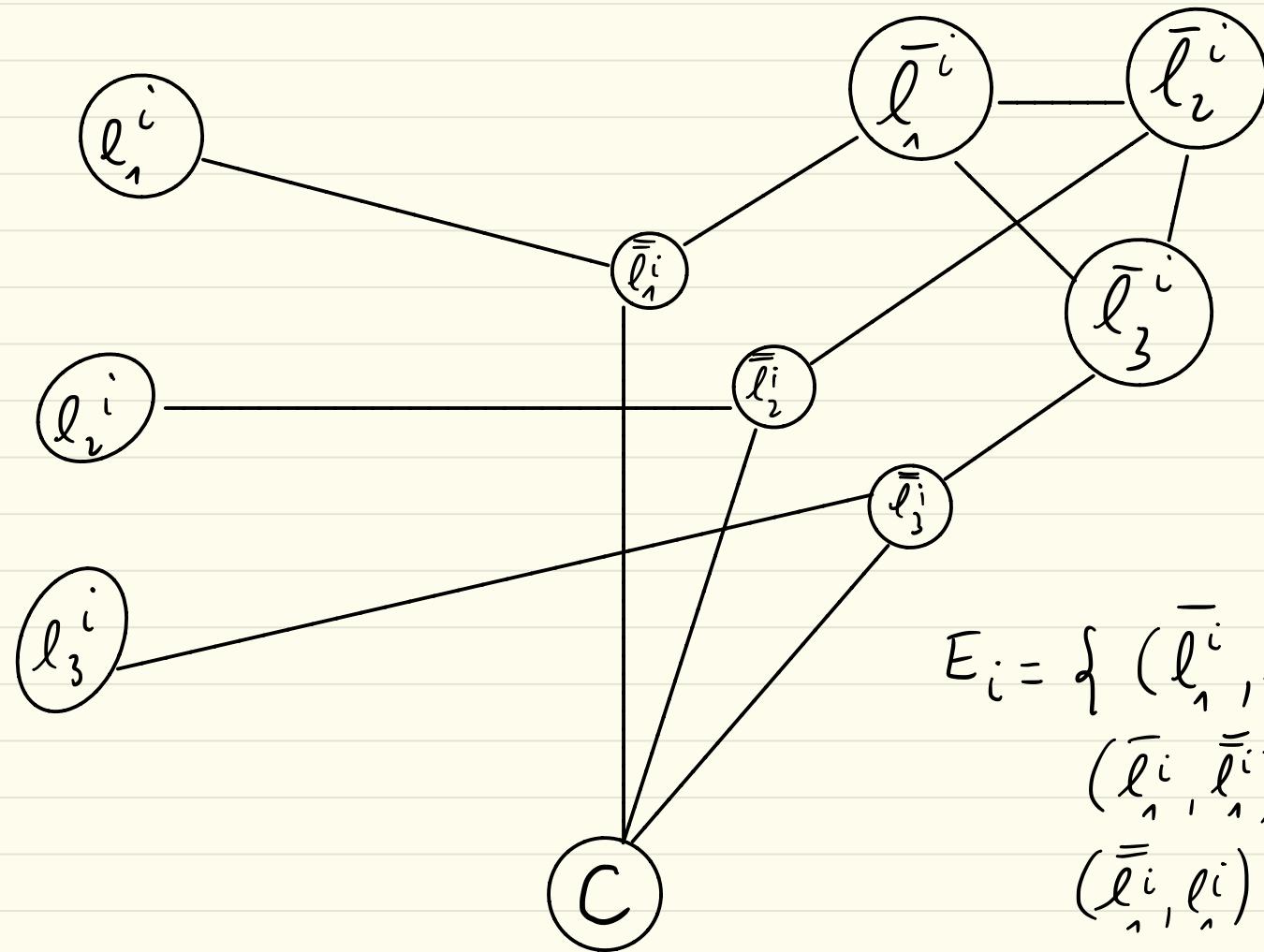
$$E_0 = \{ (A, B), (B, C), (C, A) \}$$

Pour chaque variable x :



$$E_x = \{ (A, x), (x, \bar{x}), (\bar{x}, A) \}$$

Pour chaque clause $C_i = l_1^i \vee l_2^i \vee l_3^i$:



$$E_i = \{ (\bar{l}_1^i, \bar{l}_2^i), (\bar{l}_2^i, \bar{l}_3^i), (\bar{l}_3^i, \bar{l}_4^i), \\ (\bar{l}_1^i, \bar{\bar{l}}_1^i), (\bar{l}_2^i, \bar{\bar{l}}_2^i), (\bar{l}_3^i, \bar{\bar{l}}_3^i), \\ (\bar{\bar{l}}_1^i, l_1^i), (\bar{\bar{l}}_2^i, l_2^i), (\bar{\bar{l}}_3^i, l_3^i), \\ (\bar{\bar{l}}_1^i, C), (\bar{\bar{l}}_2^i, C), (\bar{\bar{l}}_3^i, C) \}$$

L'ensemble des arêtes est $E = E_0 \cup \dots \cup E_m \cup E_{x_1} \cup \dots \cup E_{x_n}$

CORRECTION DE LA CONSTRUCTION

①

On montre d'abord que si S est satisfaisable, alors G_S est coloriable avec 3 couleurs.

Soit VFC . On note $\{0, 1, \#\}$ les 3 couleurs.

On applique le coloriage suivant :

$$x_1 \rightsquigarrow V(x_1) \quad A \rightsquigarrow \#$$

$$\neg x_1 \rightsquigarrow V(\neg x_1) \quad B \rightsquigarrow 0$$

⋮

$$x_n \rightsquigarrow V(x_n)$$

$$\neg x_n \rightsquigarrow V(\neg x_n)$$

Pour chaque clause C_i , il existe

au moins un littéral l_j^i ($j \in \{1, 2, 3\}$)

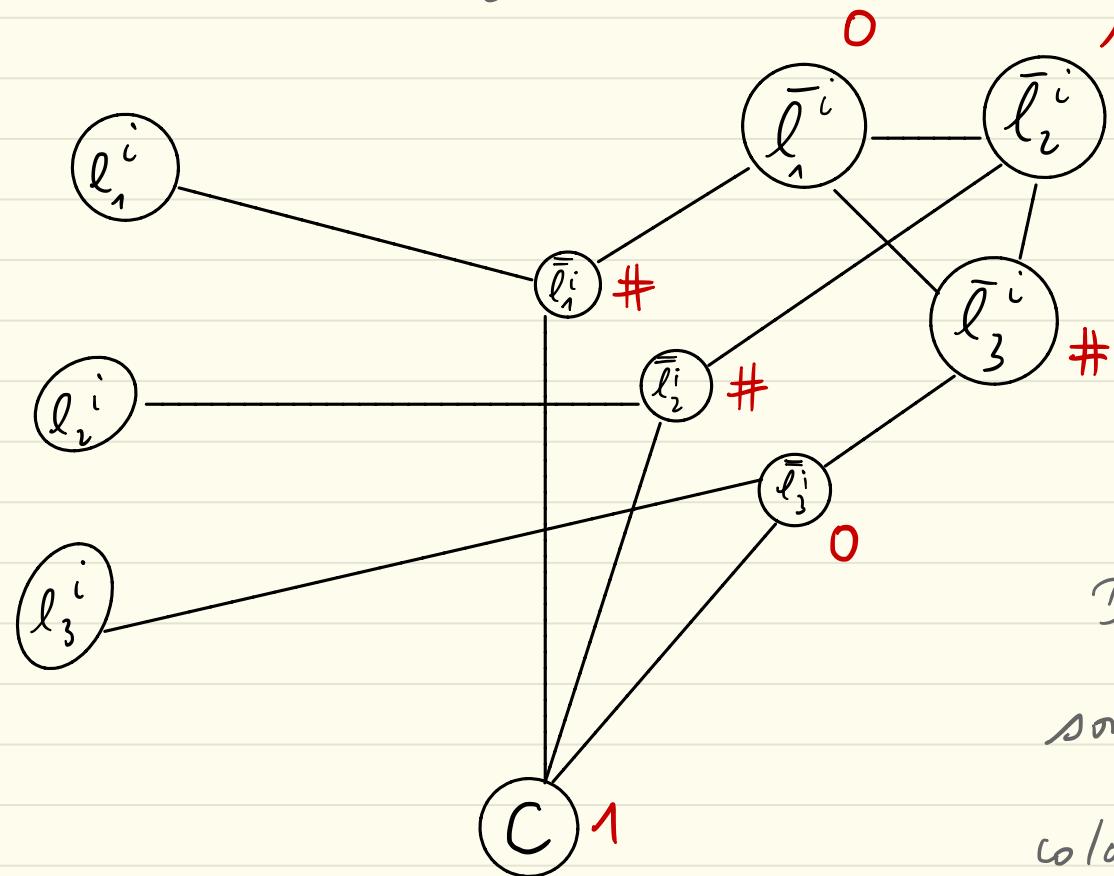
$C \rightsquigarrow 1$ tel que $V(l_j^i) = 1$. On colore

alors \bar{l}_j^i par $\#$, $\bar{\bar{l}}_j^i$ par 0.

D'après le choix des couleurs pour $x_1, \neg x_1, \dots$
la couleur de l_j^i est $V(l_j^i) = 1$

Les deux autres sommets $\bar{l}_{j'}^i$ et $\bar{l}_{j''}^i$ ($j' < j''$) sont coloriés par 0 et 1.
 Les deux sommets \bar{l}_1^i et \bar{l}_2^i sont par #.

Exemple avec $j = 2$:



Le coloriage est correcte car nécessairement on \bar{l}_3^i colorié par $V(\bar{l}_3^i) = 1$ donc il

n'y a pas de conflit avec la couleur de \bar{l}_3^i (0).

D'autre-part \bar{l}_1^i et \bar{l}_2^i sont par définition du coloriage coloriés par des couleurs différentes de #.

2

On montre que si G_S est colorable avec 3 couleurs, alors S est satisfaisable.

Appelons $\{0, 1, \#\}$ des trois couleurs.

On peut supposer que A est coloré par $\#$, B par 0 et C par 1, sinon on applique une permutation de l'ensemble des couleurs (ce qui préserve le fait d'être un coloriage correct).

On définit la valuation V par

$V(x_i) = \text{couleur du sommet } x_i \cdot \begin{cases} \text{nécessairement} \\ \text{coloré par 0 ou} \\ 1 \text{ car } x_i \text{ est relié} \\ \text{à } A \text{ qui est coloré} \\ \text{par } \# \end{cases}$

On va montrer que $V \models S$.

Remarque: On a bien que la couleur de $\neg x_i$ est $\neg V(x_i)$ car x_i et $\neg x_i$ sont reliés, et eux-mêmes reliés à A , coloré par $\#$.

Prenons pour cela une clause $C_i = l_1^{i_1} \vee l_2^{i_2} \vee l_3^{i_3}$ de S .

Nécessairement, un des sommets $\overline{l_j^{i_j}}$ est colorié par $\#$ puisque le coloriage est correcte. Donc $\overline{\overline{l_j^{i_j}}}$ est colorié par 0, car il est relié à C (colorié par 1) et $\overline{l_j^{i_j}}$ (colorié par $\#$). Donc $\overline{l_j^{i_j}}$ est colorié par 1, et donc par définition de V et la remarque précédente, on a $V(\overline{l_j^{i_j}}) = 1$.

Donc $V \models C_i$

③ Il est facile de définir un algorithme de complexité polynomiale en temps, qui construit G_S à partir de S . Notons aussi que le nombre de sommets de G_S est $2n + 3 + 6m$, donc polynomial dans la taille de S , ce qui termine la preuve.

Temps non-déterministe polynomial

- ▶ une définition équivalente de NP peut être donnée
- ▶ c'est la classe des problèmes pouvant être décidés en temps polynomial par un algorithme non-déterministe
- ▶ cet algorithme doit s'arrêter après un nombre d'étapes polynomial
- ▶ il a le droit de faire des choix aléatoires (tirer à pile ou face)
- ▶ si la réponse au problème est OUI, alors il doit exister une exécution de l'algorithme (i.e. une suite de tirages) après laquelle l'algorithme répondra OUI
- ▶ **Exemple : SAT** choisir aléatoirement une valeur de vérité pour chaque variable, et vérifier en temps linéaire qu'elles satisfont la formule.
- ▶ Pourquoi cette définition est équivalente ?
- ▶ idée : certificat = suite de tirages

Autres classes

- ▶ PSPACE : problèmes résolubles en espace polynomial
- ▶ EXPTIME : problèmes résolubles en temps exponentiel
- ▶ NPSPACE : résolubles par algorithme non-déterministes en espace polynomial (NPSPACE = PSPACE, Théorème de Savitch)
- ▶ ...
- ▶ kEXPTIME, kNEXPTIME
- ▶ décidables
- ▶ https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Autres classes

- ▶ PSPACE : problèmes résolubles en espace polynomial
- ▶ EXPTIME : problèmes résolubles en temps exponentiel
- ▶ NPSPACE : résolubles par algorithme non-déterministes en espace polynomial (NPSPACE = PSPACE, Théorème de Savitch)
- ▶ ...
- ▶ kEXPTIME, kNEXPTIME
- ▶ décidables
- ▶ https://complexityzoo.uwaterloo.ca/Complexity_Zoo

$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq 2EXPTIME \subseteq \dots \subseteq \dots \subseteq$ décidables

37 Problèmes Indécidables : Définition

Problème Indécidable

Un problème de décision P sur un alphabet Σ est indécidable s'il n'existe pas d'algorithme² A prenant un mot sur Σ et retournant, en un nombre fini d'étapes de calcul, la valeur 0 ou 1 , tel que pour tout mot x sur Σ , $A(x)$ retourne 1 si et seulement si $x \in P$.

Autrement dit, un problème est indécidable s'il n'existe pas d'algorithme pour le résoudre.

Nous allons voir deux exemples.

2. Entendez par algorithme un programme Java, C, Python ... La formalisation des notions d'algorithme et de problème n'est pas l'objet de ce cours, mais il y aura néanmoins une introduction aux machines de Turing.

38 Problème de l'Arrêt

Exemple de programmes qui ne s'arrête pas :

```
while(true) print "bonjour" ;
```

Définition

ENTREE : le code source d'un programme P lisant des chaînes de caractères, et une chaîne de caractères x

SORTIE : 1 si et seulement si le programme P s'arrête sur l'entrée x , et 0 sinon.

38 Problème de l'Arrêt

Exemple de programmes qui ne s'arrête pas :

```
while(true) print "bonjour" ;
```

Définition

ENTRÉE : le code source d'un programme P lisant des chaînes de caractères, et une chaîne de caractères x

SORTIE : 1 si et seulement si le programme P s'arrête sur l'entrée x , et 0 sinon.

Théorème (Turing, 1936)

Le problème de l'arrêt est indécidable.

39 Preuve d'Indécidabilité du Problème de l'Arrêt

On procède par l'absurde, en supposant qu'il existe un programme $\text{HALT}(c_P, x)$ qui décide le problème de l'arrêt, pour tout programme P donné par son code c_P , et toute chaîne de caractère x .

A partir de HALT , on définit le programme PARADOX suivant :

```
PARADOX(c : string)
  if HALT(c,c) then loop forever
  else stop
```

On appelle $\text{PARADOX}(c_{\text{PARADOX}})$, où c_{PARADOX} est le code source du programme PARADOX ... que se passe-t-il ?

40 Preuve d'Indécidabilité du Problème de l'Arrêt

Deux cas :

- ▶ si $\text{PARADOX}(c_{\text{PARADOX}})$ s'arrête, alors c'est que $\text{HALT}(c_{\text{PARADOX}}, c_{\text{PARADOX}}) = 0$, i.e. $\text{PARADOX}(c_{\text{PARADOX}})$ ne s'arrête pas.
- ▶ si $\text{PARADOX}(c_{\text{PARADOX}})$ ne s'arrête pas, alors c'est que $\text{HALT}(c_{\text{PARADOX}}, c_{\text{PARADOX}}) = 1$, i.e. $\text{PARADOX}(c_{\text{PARADOX}})$ s'arrête.

Dans les deux cas on obtient une contradiction, c'est donc que le programme HALT n'existe pas.

41 Problème de la Correspondance de Post (PCP)

Soit $\Sigma = \{0, 1\}$. Un mot u sur Σ est une séquence finie d'éléments de Σ .

Deux mots u et v peuvent être concaténés pour former un nouveau mot noté uv . L'élément neutre pour la concaténation est noté ϵ (mot vide).

Par exemple : $u = 01$ est un mot, $v = 110$ est un mot, $uv = 01110$, $\epsilon u = u\epsilon = u = 01$.

Le problème de la Correspondance de Post (PCP) se formule de la manière suivante :

Entrée

$(u_1, v_1), \dots, (u_n, v_n)$, $n \geq 1$, n paires de mots (possiblement vides) sur Σ .

Sortie

1 si et seulement si il existe une séquence finie d'indices $i_1, \dots, i_k \in \{1, \dots, n\}$ telle que $k \geq 1$ et

$$u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}$$

42 PCP : exemples

- ▶ Instance : $(u_1, v_1) = (100, 00)$, $(u_2, v_2) = (0, 01)$,

42 PCP : exemples

- ▶ Instance : $(u_1, v_1) = (100, 00)$, $(u_2, v_2) = (0, 01)$, Solution (parmi d'autres) : 2,1

$$u_2 u_1 = 0100 = v_2 v_1$$

- ▶ Instance : $(u_1, v_1) = (1, 100)$, $(u_2, v_2) = (0, \epsilon)$,

42 PCP : exemples

- ▶ Instance : $(u_1, v_1) = (100, 00)$, $(u_2, v_2) = (0, 01)$, Solution (parmi d'autres) : 2,1

$$u_2 u_1 = 0100 = v_2 v_1$$

- ▶ Instance : $(u_1, v_1) = (1, 100)$, $(u_2, v_2) = (0, \epsilon)$, Solution : 1,2,2

$$u_1 u_2 u_2 = 100 = v_1 v_2 v_2$$

- ▶ Instance : $(u_1, v_1) = (1, 0)$, $(u_2, v_2) = (0, 1)$,

42 PCP : exemples

- ▶ Instance : $(u_1, v_1) = (100, 00)$, $(u_2, v_2) = (0, 01)$, Solution (parmi d'autres) : 2,1

$$u_2 u_1 = 0100 = v_2 v_1$$

- ▶ Instance : $(u_1, v_1) = (1, 100)$, $(u_2, v_2) = (0, \epsilon)$, Solution : 1,2,2

$$u_1 u_2 u_2 = 100 = v_1 v_2 v_2$$

- ▶ Instance : $(u_1, v_1) = (1, 0)$, $(u_2, v_2) = (0, 1)$, Pas de solution

- ▶ Instance :

$$(u_1, v_1) = (0, 100), (u_2, v_2) = (01, 00), (u_3, v_3) = (110, 11) ?$$

42 PCP : exemples

- ▶ Instance : $(u_1, v_1) = (100, 00)$, $(u_2, v_2) = (0, 01)$, Solution (parmi d'autres) : 2,1

$$u_2 u_1 = 0100 = v_2 v_1$$

- ▶ Instance : $(u_1, v_1) = (1, 100)$, $(u_2, v_2) = (0, \epsilon)$, Solution : 1,2,2

$$u_1 u_2 u_2 = 100 = v_1 v_2 v_2$$

- ▶ Instance : $(u_1, v_1) = (1, 0)$, $(u_2, v_2) = (0, 1)$, Pas de solution

- ▶ Instance :

$(u_1, v_1) = (0, 100)$, $(u_2, v_2) = (01, 00)$, $(u_3, v_3) = (110, 11)$?

$$u_3 u_2 u_3 u_1 = (110)(01)(110)(0) = 110011100 = (11)(00)(11)(100) = v_3 v_2 v_3 v_1$$

43 Théorèmes

Théorème

Le problème de Correspondance de Post est indécidable.

Résultat admis.

44 Autres Problèmes Indécidables

- ▶ **10é problème de Hilbert** : résolution d'équations diophantiennes (à solution entière). Indécidabilité prouvée en 1971 (Matiyasevic).

Entrée $p(x_1, \dots, x_n)$ un polynôme à coefficients entiers

Sortie $\exists i_1 \dots \exists i_n \in \mathbb{Z} \cdot p(i_1, \dots, i_n) = 0$?

Remarque : c'est décidable dans les réels (Tarski, 1951).

44 Autres Problèmes Indécidables

- ▶ **10é problème de Hilbert** : résolution d'équations diophantiennes (à solution entière). Indécidabilité prouvée en 1971 (Matiyasevic).

Entrée $p(x_1, \dots, x_n)$ un polynôme à coefficients entiers

Sortie $\exists i_1 \dots \exists i_n \in \mathbb{Z} \cdot p(i_1, \dots, i_n) = 0$?

Remarque : c'est décidable dans les réels (Tarski, 1951).

- ▶ **Complexité de Kolmogorov** : étant donné un mot binaire w et un entier $k \in \mathbb{N}$, décider s'il existe un programme Java qui écrit w et dont le code est un fichier d'au plus k bits. **Remarque** : le choix du formalisme qui écrit le mot (Java,Lisp,C,machine de Turing) influe sur la complexité.

45 Autres Problèmes Indécidables

- ▶ **PCP avec $n = 7$** : dans cette variante on se donne 7 paires de mots $(u_1, v_1), \dots, (u_7, v_7)$. **Remarque** : décidable pour $n \leq 2$, ouvert pour $3 \leq n \leq 6$.
- ▶ **Castor affairé** : étant donné un programme (donné par son code source, par exemple Java) qui écrit des 0 et des 1 et qui s'arrête, on veut décider si il est maximal, i.e. qu'il n'existe pas un autre programme dont le code source n'est pas plus grand, mais qui écrit un mot strictement plus grand.

46 Comment démontrer qu'un problème est indécidable ?

- ▶ Pour montrer l'indécidabilité d'un problème de décision P_1 , on peut partir d'un problème de décision P_2 connu indécidable et on montre l'existence d'un algorithme, appelé *réduction*, qui pour toute instance I_2 de P_2 construit une instance I_1 de P_1 telle que I_1 a une solution si et seulement si I_2 a une solution.
- ▶ De cette façon, on montre que P_1 est "aussi difficile" que P_2 . S'il existait un algorithme pour résoudre P_1 , alors cela nous donnerait un algorithme pour résoudre P_2 : appliquer la réduction puis l'algorithme pour P_1 . Cela contredirait le fait que P_2 soit indécidable, donc il n'existe pas d'algorithme pour résoudre P_1 .

46 Comment démontrer qu'un problème est indécidable ?

- ▶ Pour montrer l'indécidabilité d'un problème de décision P_1 , on peut partir d'un problème de décision P_2 connu indécidable et on montre l'existence d'un algorithme, appelé *réduction*, qui pour toute instance I_2 de P_2 construit une instance I_1 de P_1 telle que I_1 a une solution si et seulement si I_2 a une solution.
- ▶ De cette façon, on montre que P_1 est "aussi difficile" que P_2 . S'il existait un algorithme pour résoudre P_1 , alors cela nous donnerait un algorithme pour résoudre P_2 : appliquer la réduction puis l'algorithme pour P_1 . Cela contredirait le fait que P_2 soit indécidable, donc il n'existe pas d'algorithme pour résoudre P_1 .
- ▶ Certains problèmes ont directement été montré indécidables : le problème de terminaison d'un programme par exemple, mais aussi le problème de validité dans la logique du premier ordre (travaux d'Alonzo Church et Alan Turing).
- ▶ Remarque : la notion de réduction est la même que pour démontrer la NP-dureté d'un problème, à la seule différence qu'on ne demande pas que l'algorithme qui calcule la réduction le fasse en temps polynomial.

47 Exercice

1. Montrer que le problème suivant est indécidable. Etant donnés deux programmes P et Q qui retournent tous les deux une valeur Booléenne, est-ce que les deux programmes sont équivalents, i.e. est-ce que pour toute entrée x , on a $P(x) = Q(x)$?

47 Exercice

1. Montrer que le problème suivant est indécidable. Etant donnés deux programmes P et Q qui retournent tous les deux une valeur Booléenne, est-ce que les deux programmes sont équivalents, i.e. est-ce que pour toute entrée x , on a $P(x) = Q(x)$?
 - ▶ on va faire une réduction à partir du problème de l'arrêt, qui est indécidable.
 - ▶ étant donné un programme R , on considère les deux programmes :

procedure AUX(x):	procedure CONST(x):
$r := R(x)$;	return 1;
return 1 ;	
 - ▶ pour toute entrée x , on a $AUX(x) = CONST(x)$ si et seulement si $R(x)$ s'arrête sur l'entrée x . Si le problème d'équivalence était décidable, alors le problème de l'arrêt le serait aussi.