

Complexité CM7

Antonio E. Porreca
aeporreca.org/complexite

! Correction de l'exercice 2.3 du contrôle continu !

Montrer que le problème **QUATRE-ENNEMIS** est dans **P** :

- **Entrée** : un graphe orienté $G = (V, E)$
- **Question** : le graphe G contient-il un ensemble V' de quatre sommets tels que aucune paire de sommets dans V' n'est relié par un arc ?

! Correction de l'exercice 2.3

du contrôle continu !

fonction quatre-ennemis(V, E)

pour $u_0 \in V$ **faire**

$O(|V|)$

pour $u_1 \in V - \{u_0\}$ **faire**

$O(|V|)$

pour $u_2 \in V - \{u_0, u_1\}$ **faire**

$O(|V|)$

pour $u_3 \in V - \{u_0, u_1, u_2\}$ **faire**

$O(|V|)$

$V' := \{u_0, u_1, u_2, u_3\}$

$bon := \text{vrai}$

pour v dans V' **faire**

pour w dans V' **faire**

si $(v, w) \in E$ **alors**

$bon := \text{false}$

si bon **alors**

retourner true

retourner false

$O(1)$

$O(|V|^4) \leq O(n^4)$

**Précédemment
dans Complexité...**

! 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 $f: \Sigma_1^* \rightarrow \Sigma_2^*$ calculable en temps polynomial telle que

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

- Si une telle f existe, on dit que **L_1 se réduit à L_2** (via f) et on notera **$L_1 \leq_m^P L_2$** (ou parfois, en bref, **$L_1 \leq L_2$**)

! Definition 3-J (p. 67) !

Difficulté et complétude

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

- On dit que L est \mathcal{C} -difficile (ou \mathcal{C} -dur) si pour tout problème $L' \in \mathcal{C}$ on a $L' \leq L$
- On dit que L est \mathcal{C} -complet s'il est \mathcal{C} -difficile et en plus on a $L \in \mathcal{C}$

P a (beaucoup de) problèmes complets

- Tout problème $L \in \mathbf{P}$ non trivial (c'est-à-dire, $L \neq \emptyset$ et $L \neq \Sigma^*$) est **P-complet** pour les reductions en temps polynomial
- Ça veut dire que cette notion de complétude n'est pas très intéressant pour **P**...

**Et maintenant,
la suite**

Il existe bien des problèmes
NP-complets,
ou : on est pas là pour
perdre notre temps

Proposition 3-M (p. 68)

La prédiction est **NP**-complète

Le problème suivant est **NP**-complet :

$$A = \{(\langle N \rangle, x, 1^t) : N(x) \text{ accepte en temps } \leq t\}$$

code d'une MT
non déterministe

mot d'entrée
pour la MT N

entier $t \in \mathbb{N}$
en unaire



Idée de la démonstration

- $A \in \mathbf{NP}$ parce qu'on peut **simuler $N(x)$** avec une machine de Turing universelle non déterministe
- Pour tout $B \in \mathbf{NP}$, on a $B \leq A$ parce qu'**on peut prévoir le résultat** de la machine N_B qui reconnaît B
 - Le problème A est, justement, la prédiction du résultat d'une TM non déterministe

Démonstration : $A \in \text{NP}$

- Soit U une machine non déterministe efficace
- Voilà un algorithme pour décider si $(\langle N \rangle, x, 1^t) \in A$:
 - **Simuler** $N(x)$ pendant **t étapes** en exécutant $U(\langle N \rangle, x)$
 - Si $U(\langle N \rangle, x)$ **accepte** en $\leq t$ étapes, **accepter**
 - Si elle **rejette** ou elle **n'a pas terminé** en t étapes, **rejeter**
- Comme U simule N en temps polynomial, cet algorithme fonctionne aussi en **temps polynomial**

Démonstration :

$$\forall B \in \mathbf{NP}, B \leq A$$

- Soit $B \in \mathbf{NP}$ et soit N_B une machine non déterministe qui reconnaît B en temps polynomial $p(n)$
- Donc on connaît N_B et son temps de calcul $p(n)$, puisque c'est nécessaire pour prouver que $B \in \mathbf{NP}$
- Voilà la réduction : $f(x) = (\langle N_B \rangle, x, 1^{p(|x|)})$
- $x \in B$ ssi $N_B(x)$ accepte en temps $\leq p(|x|)$ ssi $f(x) \in A$
- f est calculable en temps polynomial

f est calculable en temps polynomial, par exemple
si N_B fonctionne en temps $p(n) = n^3 - 3n + 7$

fonction $f(x)$

écrire "(" sur le ruban de sortie

écrire le code $\langle N_B \rangle$ sur le ruban de sortie

écrire "," sur le ruban de sortie

écrire x sur le ruban de sortie

écrire "," sur le ruban de sortie

pour $i := 1$ à n faire

 pour $j := 1$ à n faire

 pour $k := 1$ à n faire

 écrire 1 sur le ruban de travail

pour $i := 1$ à 3 faire

 pour $j := 1$ à n faire

 effacer 1 du ruban de travail

pour $i := 1$ à 7 faire

 écrire 1 du ruban de travail

copier le ruban de travail sur le ruban de sortie

écrire ")" sur le ruban de sortie

fin

	{	$O(1)$	{	$O(p(n))$
		$O(n)$		
		$O(1)$		
		$O(n^3)$		
		$O(n)$		
		$O(1)$		
		$O(p(n))$		
		$O(1)$		



Remarque

- Ouais, on est bien d'accord, ce n'est **pas** un problème **très intéressant**... 🙄
- C'est un problème ad-hoc calqué sur la définition de **NP**
- La **NP**-complétude devient pertinente lorsqu'elle concerne des problèmes **naturels**
- On verra qu'il y en a **plein** de naturels !

Les problèmes NP-complets,
ou entrons enfin
dans le vif du sujet

Proposition 3-P (p. 69)

NP-complétude : tout ou rien

Les affirmations suivantes sont équivalentes :

1. **$P = NP$**
2. tout problème **NP-complet** est dans **P**
3. il existe un problème **NP-complet** dans **P**

Démonstration

- Si $\mathbf{P} = \mathbf{NP}$ (1), alors en particulier $\mathbf{NP} \subseteq \mathbf{P}$,
donc tout problème \mathbf{NP} -complet est dans \mathbf{P} (2)
- Si tout problème \mathbf{NP} -complet est dans \mathbf{P} (2),
vu qu'il existe au moins un problème \mathbf{NP} -complet,
alors il existe un problème \mathbf{NP} -complet dans \mathbf{P} (3)
- S'il existe un problème \mathbf{NP} -complet A dans \mathbf{P} (3),
alors on peut le résoudre en temps polynomial.
Mais comme $B \leq A$ pour tout $B \in \mathbf{NP}$,
alors $B \in \mathbf{P}$ aussi, donc $\mathbf{NP} \subseteq \mathbf{P}$, donc $\mathbf{P} = \mathbf{NP}$ (1)

⚠ Proposition 3-W (p. 76) ⚠

NP-complétude par réduction

- Soit C un problème **NP**-complet et $A \in \mathbf{NP}$
- Si $C \leq A$ alors A est aussi **NP**-complet

Démonstration

- Soit $B \in \mathbf{NP}$ n'importe quel problème dans \mathbf{NP}
- Comme C est \mathbf{NP} -complet, on a $B \leq C$
- Si $C \leq A$, alors $B \leq A$ aussi par transitivité de \leq
- Ça vaut pour tout $B \in \mathbf{NP}$, donc A est \mathbf{NP} -difficile
- Et comme $A \in \mathbf{NP}$ aussi, il est \mathbf{NP} -complet



Remarque

- On vient d'utiliser la notion de réduction $C \leq A$, qui montre qu'un **peut résoudre C efficacement si on a une solution efficace pour A ...**
- ...d'une façon **perverse**, pour montrer que **A n'a pas de solution efficace*** !
- Autrement dit, si C est difficile est A est au moins aussi difficile que C , alors A est difficile aussi

* Dans l'hypothèse que $P \neq NP$

⚠ Théorème 3-V (p. 72) ⚠

Théorème de Cook-Levin

- Considérons le problème SAT :
 - **Entrée** : une formule booléenne φ
 - **Question** : φ est-elle satisfaisable ?
- SAT est **NP**-complet
- On verra la démonstration dans un prochain épisode

Proposition 3-Z (p. 77)

3SAT est NP-complet

- Considérons le problème 3SAT :
 - **Entrée** : une formule booléenne φ en forme normale conjonctive avec au plus 3 littéraux par clause, par exemple :

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

- **Question** : φ est-elle satisfaisable ?
- 3SAT est NP-complet

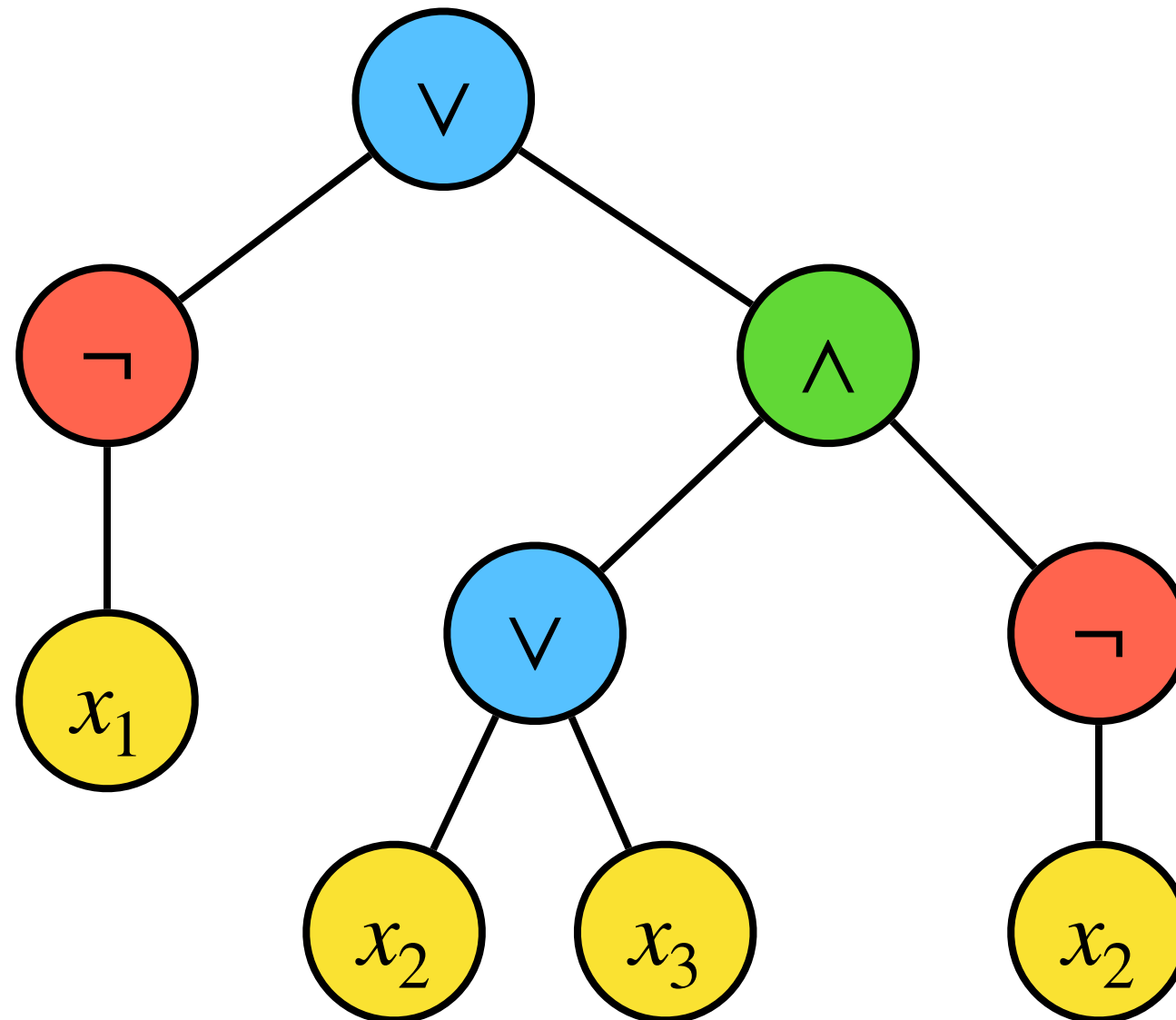


Idée de la démonstration

- On voit une instance $\varphi(x_1, \dots, x_n)$ de SAT sous la forme d'un **arbre syntaxique**
- Tester sa satisfaisabilité revient alors à **deviner les valeurs de x_1, \dots, x_n et celles des nœuds** de l'arbre et **tester** la cohérence de ces choix
- Ce test est **local à chaque nœud** de l'arbre et nécessite seulement des **clauses contenant trois littéraux** car le degré d'un sommet de l'arbre est au plus 3 (un père et au plus deux fils)

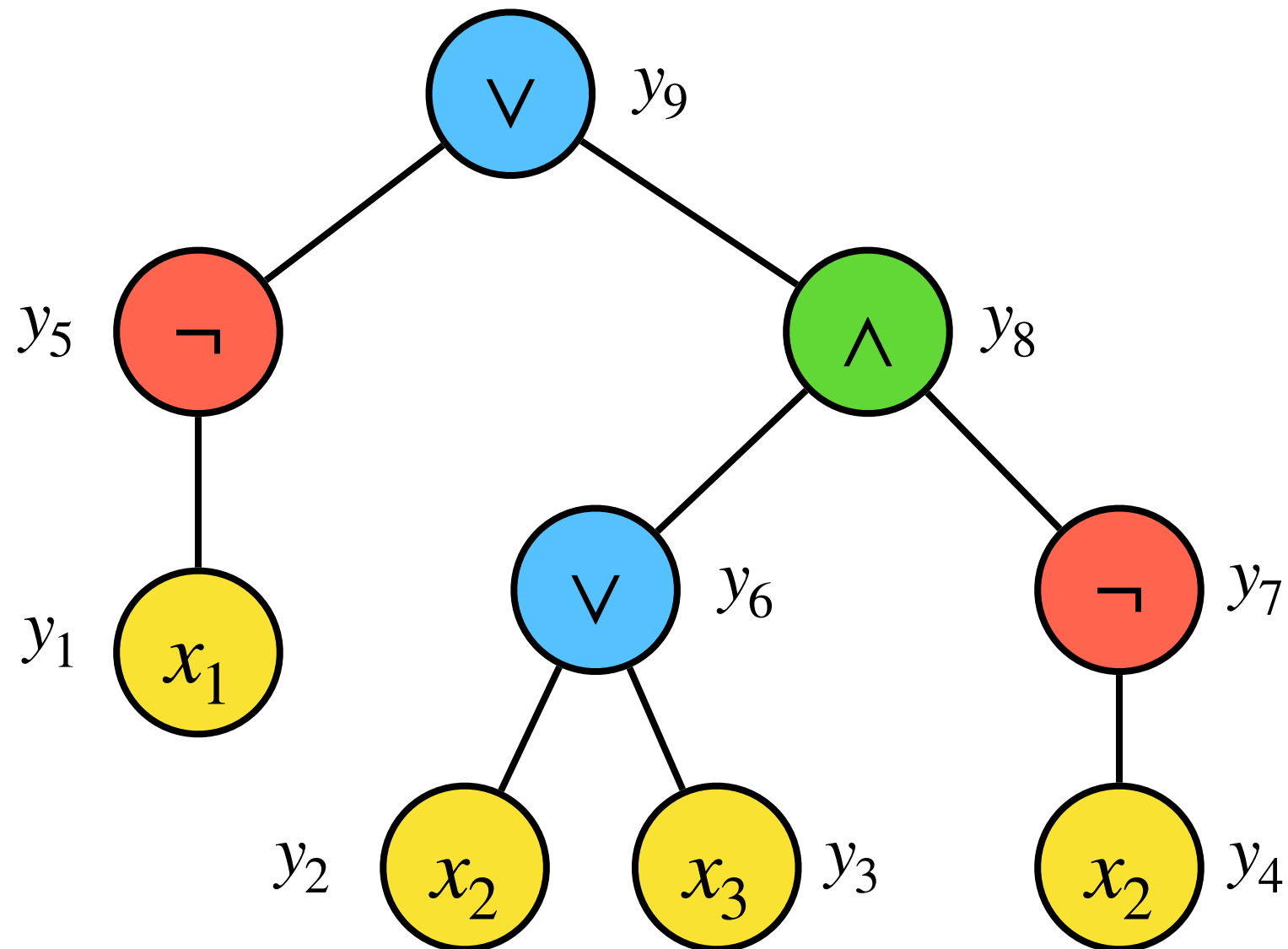
Démonstration

$$\varphi = (\neg x_1) \vee ((x_2 \vee x_3) \wedge (\neg x_2))$$

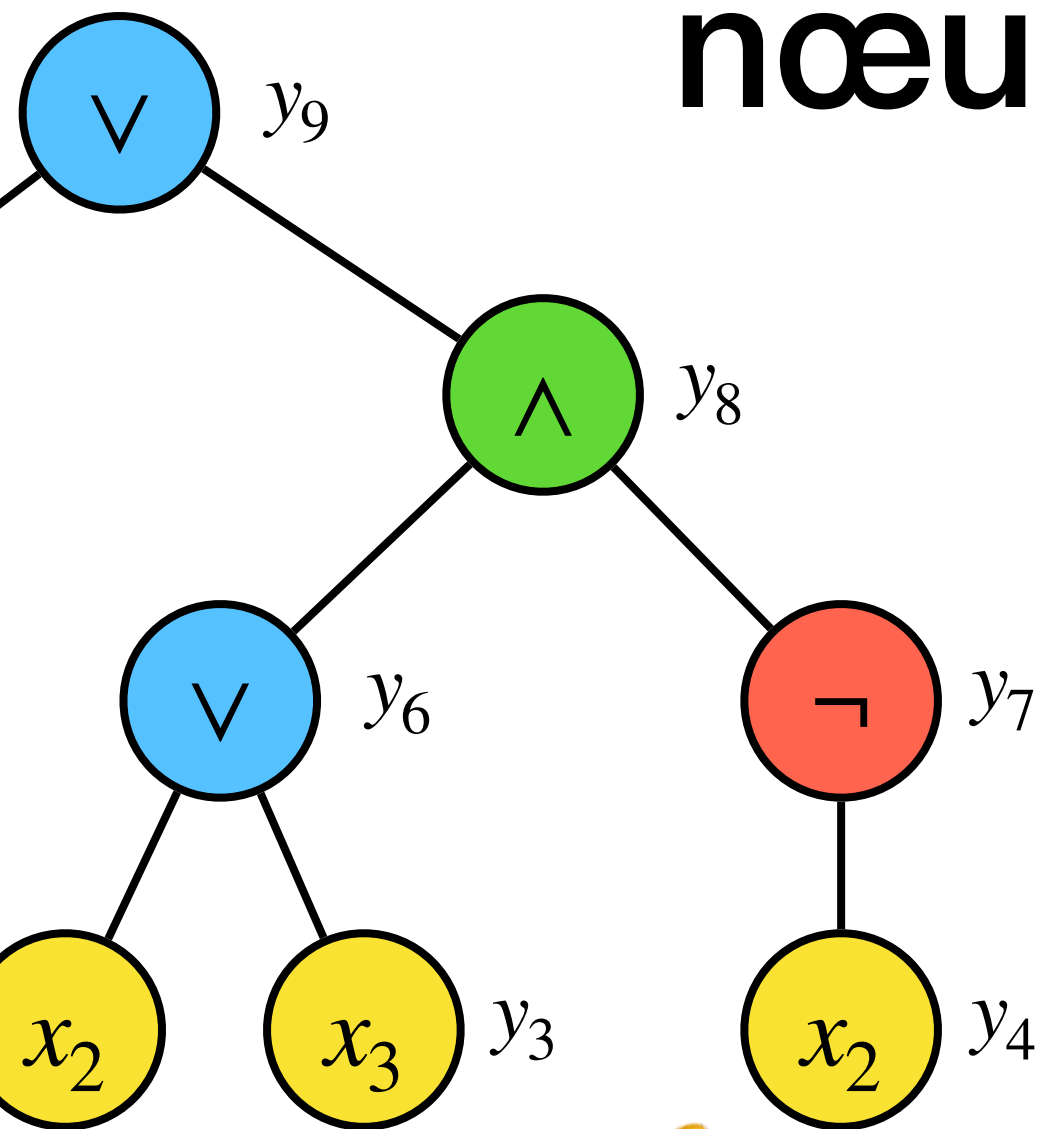


Démonstration

$$\varphi = (\neg x_1) \vee ((x_2 \vee x_3) \wedge (\neg x_2))$$



Description des valeurs des nœuds de l'arbre



$$y_4 \iff x_2$$

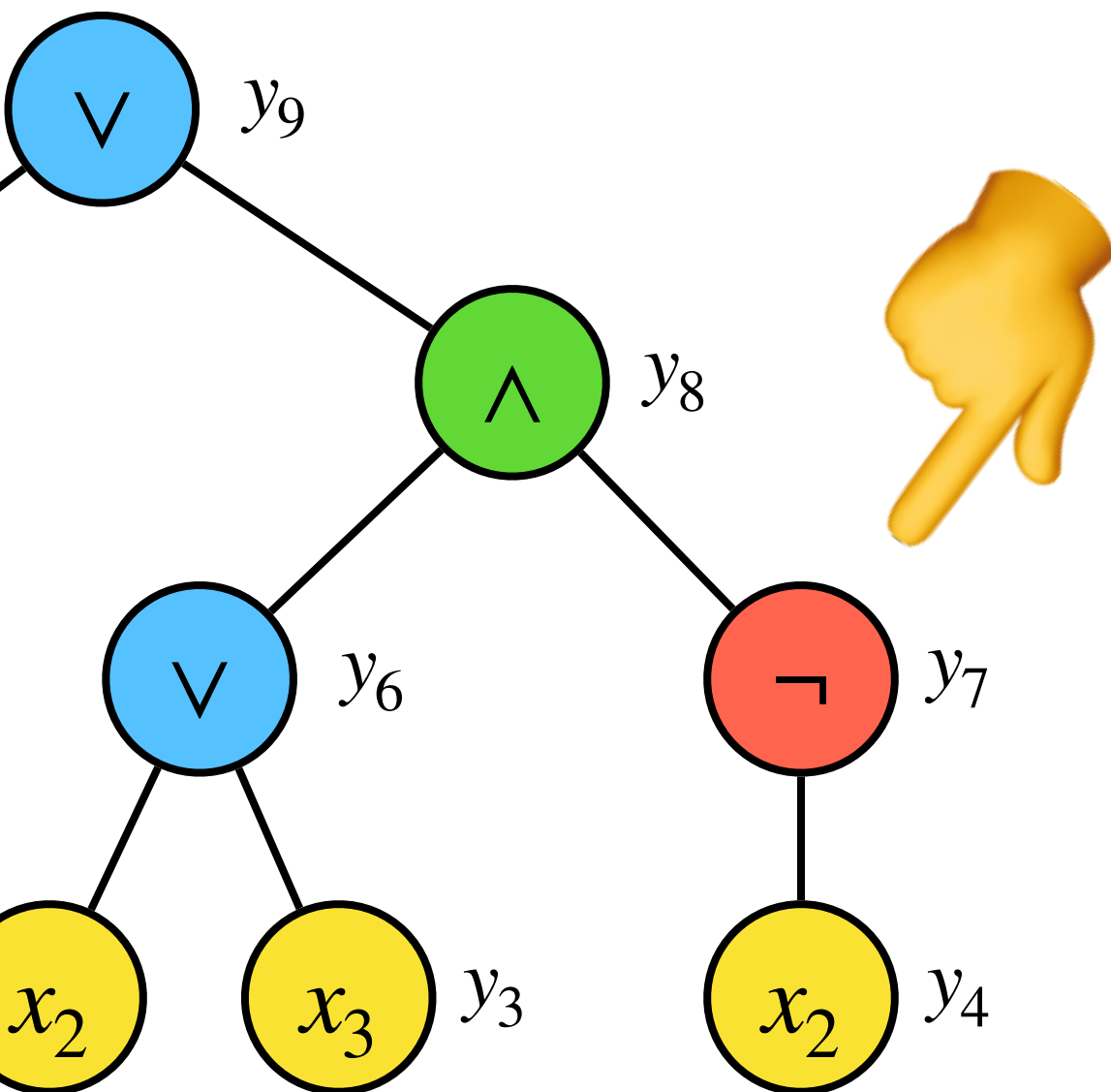
↓

$$(y_4 \Rightarrow x_2) \wedge (x_2 \Rightarrow y_4)$$

↓

$$(\neg y_4 \vee x_2) \wedge (\neg x_2 \vee y_4)$$

Description des valeurs des nœuds de l'arbre



$$y_7 \iff \neg y_4$$

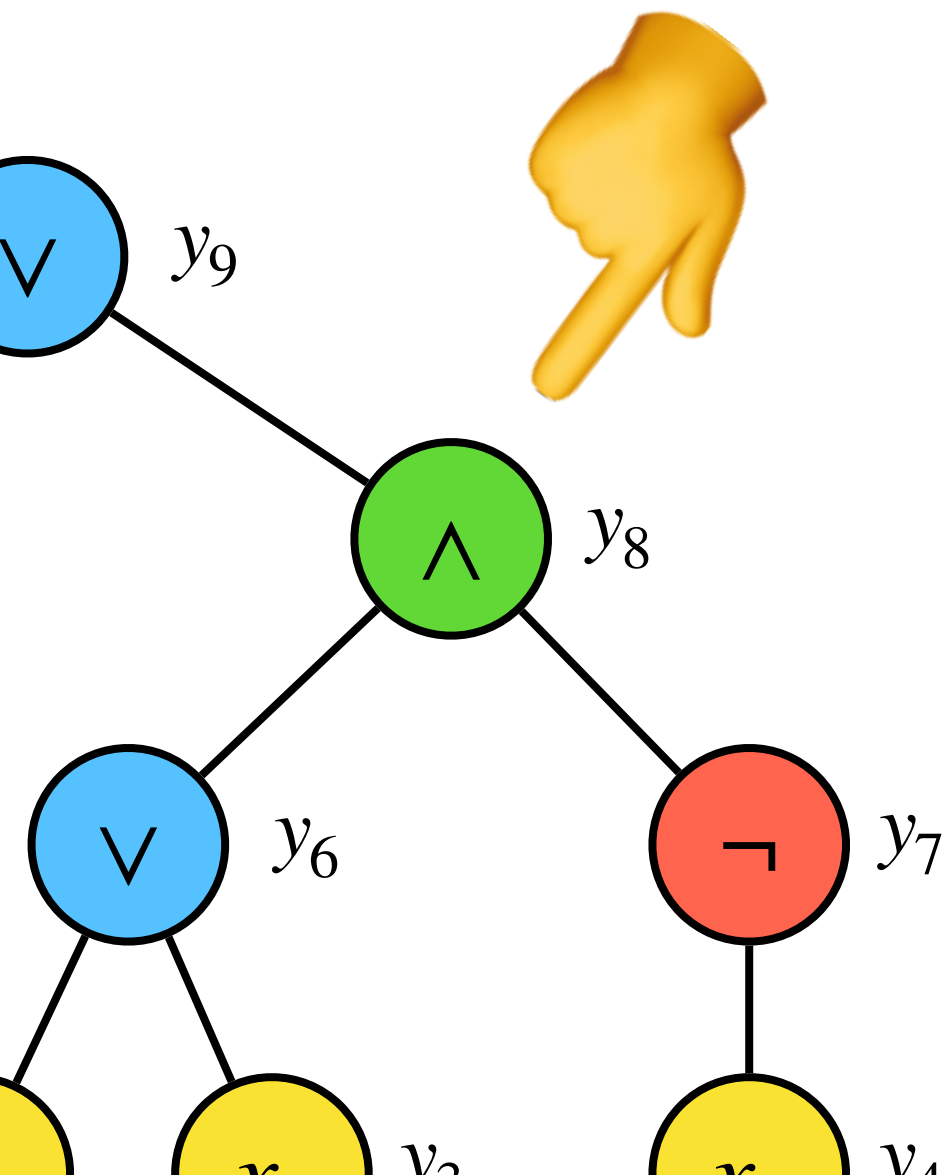
↓

$$(y_7 \Rightarrow \neg y_4) \wedge (\neg y_4 \Rightarrow y_7)$$

↓

$$(\neg y_7 \vee \neg y_4) \wedge (y_4 \vee y_7)$$

Description des valeurs des nœuds de l'arbre



$$y_8 \iff y_6 \wedge y_7$$

↓

$$(y_8 \Rightarrow y_6 \wedge y_7) \wedge (y_6 \wedge y_7 \Rightarrow y_8)$$

↓

$$(\neg y_8 \vee (y_6 \wedge y_7)) \wedge (\neg(y_6 \wedge y_7) \vee y_8)$$

↓

$$(\neg y_8 \vee y_6) \wedge (\neg y_8 \vee y_7) \wedge (\neg y_6 \vee \neg y_7 \vee y_8)$$

Description des valeurs des nœuds de l'arbre

$$y_9 \iff y_5 \vee y_8$$

↓

$$(y_9 \Rightarrow y_5 \vee y_8) \wedge (y_5 \vee y_8 \Rightarrow y_9)$$

↓

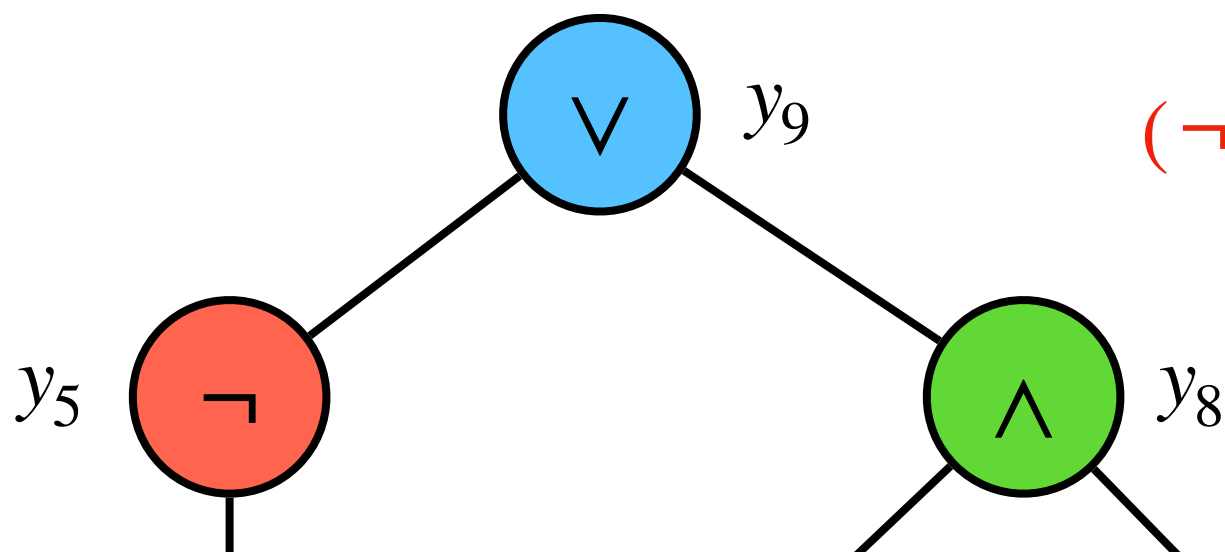
$$(\neg y_9 \vee y_5 \vee y_8) \wedge (\neg(y_5 \vee y_8) \vee y_9)$$

↓

$$(\neg y_9 \vee y_5 \vee y_8) \wedge ((\neg y_5 \wedge \neg y_8) \vee y_9)$$

↓

$$(\neg y_9 \vee y_5 \vee y_8) \wedge (\neg y_5 \vee y_9) \wedge (\neg y_8 \vee y_9)$$



Démonstration

- Pour chaque nœud $s = 1, \dots, m$ on a une conjonction C_s de deux ou trois clauses disjonctives avec au plus 3 variables
- Soit y_m la variable correspondant à la racine de l'arbre
- On construit la formule en forme normale conjonctive

$$\psi(x_1, \dots, x_n, y_1, \dots, y_m) = y_m \wedge \bigwedge_{s=1}^m C_s$$

Démonstration

$$\psi(x_1, \dots, x_n, y_1, \dots, y_m) = y_m \wedge \bigwedge_{s=1}^m C_s$$

- La formule ψ est satisfaisable ssi la formule φ l'est aussi
- La fonction $\varphi \mapsto \psi$ est calculable en temps polynomial (il suffit de construire les clauses C_s indiquées ci-dessus)
- Donc 3SAT est **NP**-difficile
- Comme il est dans **NP**, il est **NP**-complet



Remarque

- On peut résoudre 2SAT **en temps polynomial** !
- Il est **vraiment** nécessaire d'avoir **3 littéraux par clause** pour obtenir un problème **NP**-complet*
- Pourquoi 3SAT est **important** ? C'est très pratique pour faire des réductions $3SAT \leq A$ et montrer que A est **NP**-complet
- La raison est que 3SAT a une structure **plus simple et régulière** que SAT

* Dans l'hypothèse que $P \neq NP$

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 ?

Proposition 3-AE (p. 80)

ENS-INDÉP est NP-complet

- Le problème est dans **NP** car vérifier que k sommets sont indépendants se fait en temps polynomial :

pour $v \in X$ faire

pour $w \in X - \{v\}$ faire

si $(v, w) \in E$ alors

rejeter

accepter

- Pour la **NP**-difficulté, on réduit 3SAT

Proposition 3-AE (p. 80)

ENS-INDÉP est NP-complet

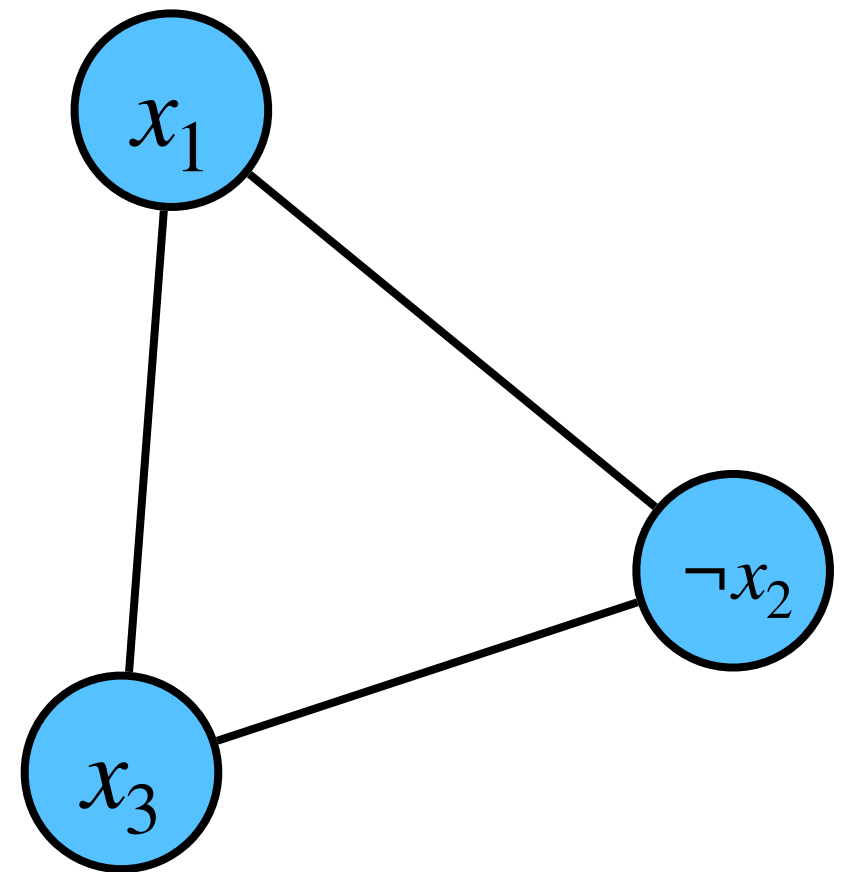
- La réduction transforme une formule φ en 3-CNF à i clauses en un graphe contenant i triangles (un pour chaque clause)
- Chaque sommet d'un triangle correspond à un littéral de la clause
- Entre les triangles, on relie ensuite x et $\neg x$
- Si φ est satisfaisable, alors les littéraux valant 1 dans une solution forment un ensemble indépendant du graphe, et réciproquement

Démonstration

- Soit $\varphi(x_1, \dots, x_n)$ une instance de 3SAT
- La formule a m clauses de au plus 3 littéraux
- Si une formule a moins de trois littéraux,
on répète un littéral pour en avoir exactement 3
- Par exemple $(x_1 \vee \neg x_2) \mapsto (x_1 \vee x_1 \vee \neg x_2)$

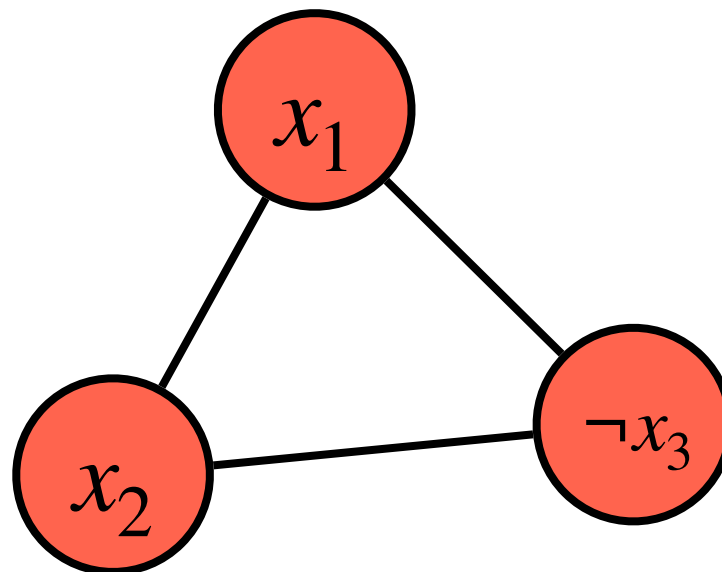
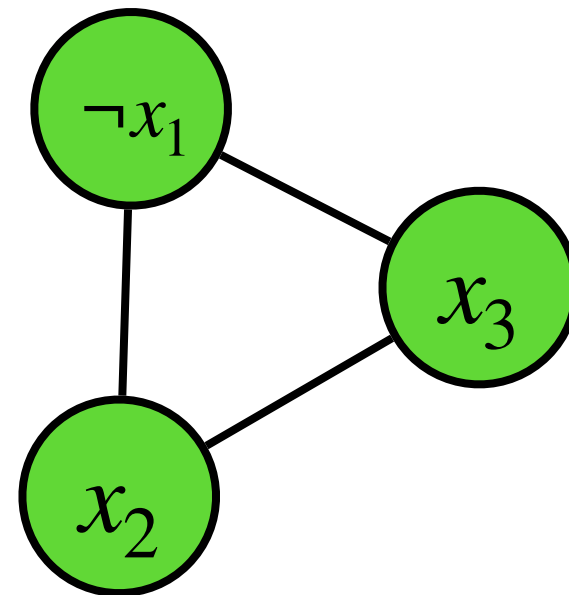
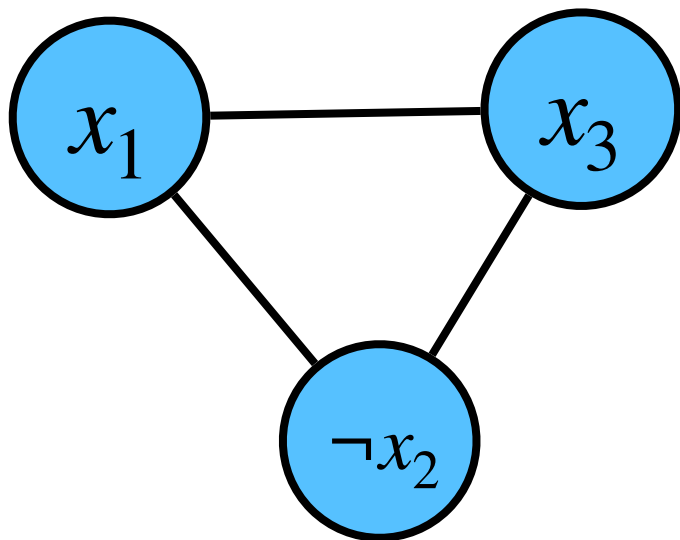
Clause \rightarrow Triangle

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



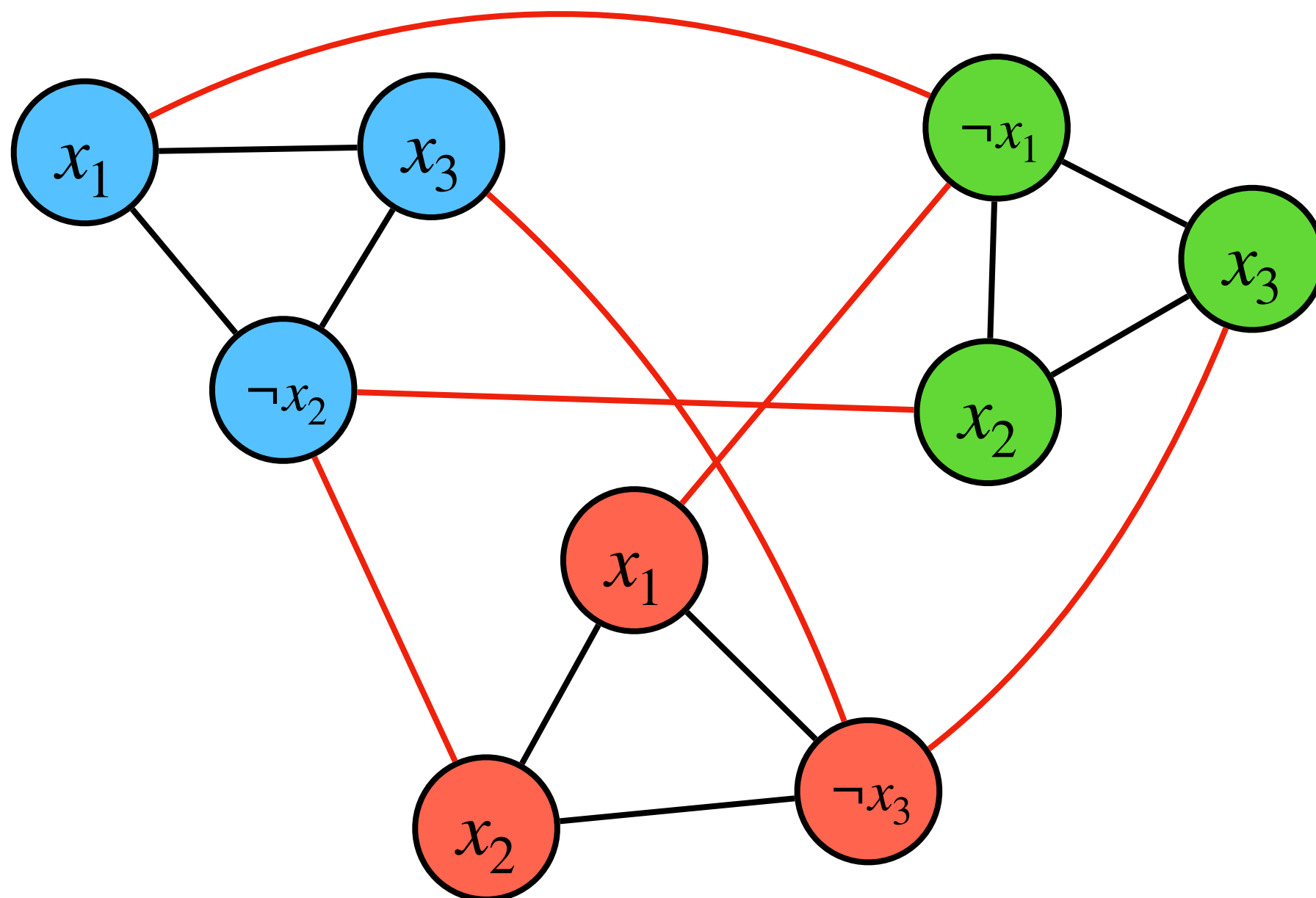
Formule \rightarrow Triangles

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$



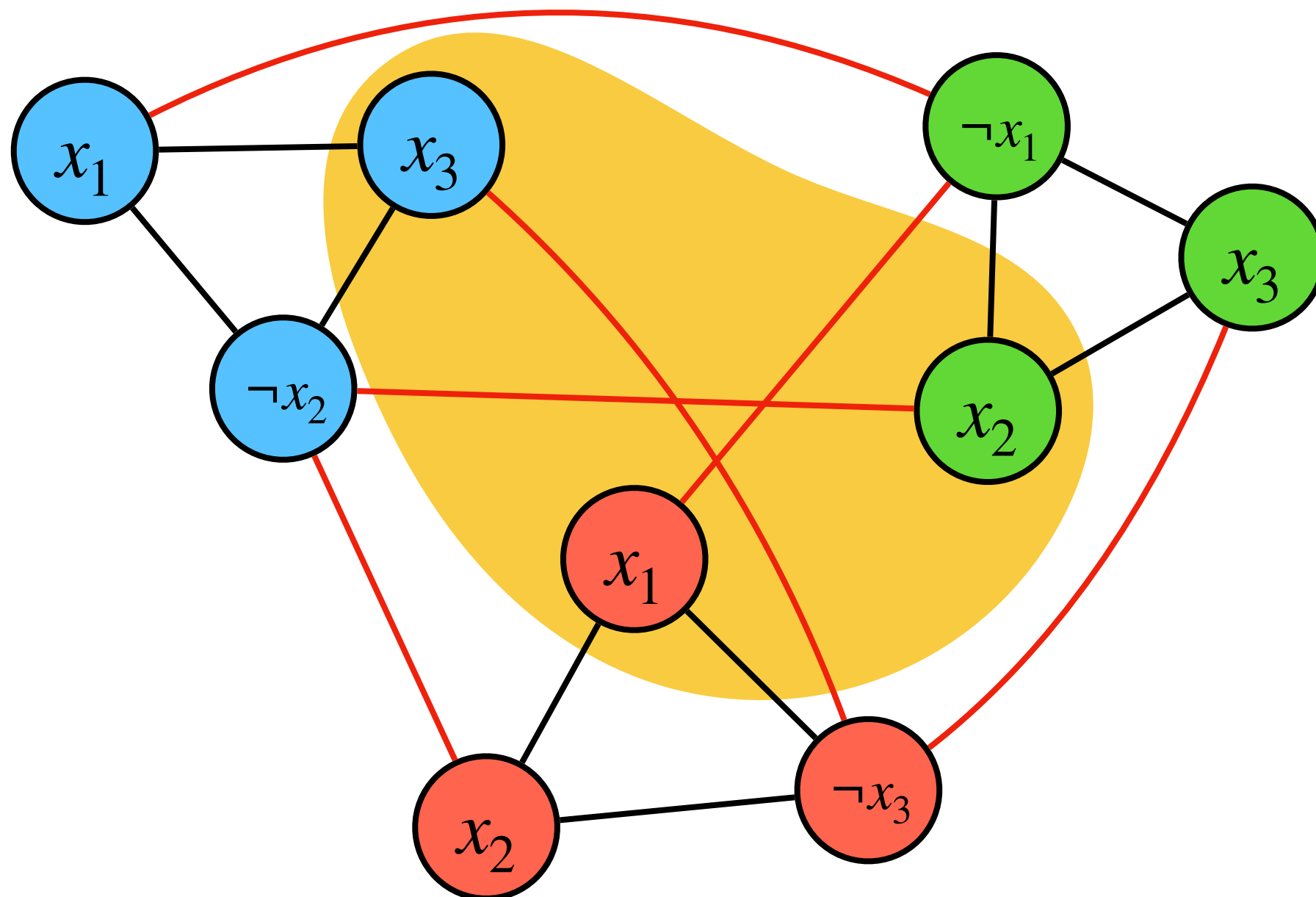
Arcs pour les négations

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$



Arcs pour les négations

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$



Graphe associé à la formule

- Soit $G = (V, E)$ le graphe ainsi construit
- Et soit $k = m$, le nombre de clauses
- On peut construire G et k à partir de φ
en temps polynomial
- Il reste à montrer que φ est satisfaisable ssi
 G a un ensemble indépendant de taille $k = m$

$$\varphi \in \text{SAT} \Rightarrow (G, k) \in \text{ENS-INDÉP}$$

- Si φ est satisfaisable par une certaine affectation des variables, **soit ℓ_1 un littéral vrai** dans la première clause (ℓ_1 est soit une variable, soit sa négation)
- Au moins l'un des sommets de G correspond à ce littéral ℓ_1
- Soit ℓ_2 un littéral vrai dans la deuxième clause ;
donc ℓ_1 et ℓ_2 ne sont **pas dans le même triangle** de G
- Aussi $\ell_1 \neq \neg \ell_2$, car ils sont tous les deux vrais ;
donc il n'y a **pas d'arête entre les deux** dans G

$$\varphi \in \text{SAT} \Rightarrow (G, k) \in \text{ENS-INDÉP}$$

- En répétant le raisonnement, on obtient **un ensemble $\{\ell_1, \dots, \ell_m\}$ de littéraux vrais** dans φ ...
- ...qui correspond à un ensemble de sommets dans G qui ne sont jamais connectés par un arête
- Donc il s'agit d'un **ensemble indépendant** de taille $m = k$
- Donc **$(G, k) \in \text{ENS-INDÉP}$**

$$(G, k) \in \text{ENS-INDÉP} \Rightarrow \varphi \in \text{SAT}$$

- Supposons que G ait un ensemble indépendant $\{\ell_1, \dots, \ell_m\}$ de taille $k = m$
- Si $i \neq j$, alors $\ell_i \neq \neg \ell_j$, sinon ils seraient reliés dans G
- Alors il existe une affectations qui rend vrais tous les ℓ_i
- Mais ℓ_i et ℓ_j ne sont jamais littéraux dans la même clause, sinon ils feraient partie du même triangle
- Donc l'affectation rend vraies toutes les clauses
- Donc $\varphi \in \text{SAT}$



Corollaire 3-AG (p. 82)

CLIQUE est NP-complet

- ENS-INDÉP est NP-complet
- $\text{ENS-INDÉP} \leq \text{CLIQUE} \in \text{NP}$
- Donc CLIQUE est NP-complet aussi

