Chapitre 5

Les classes P et NP

Définitions 5.1. Pour $w \in \Sigma^*$ et $v \in \Gamma^*$, on dit que la MT $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$

- accepte le mot w si $(\varepsilon, q_0, w) \overset{*}{\vdash} (u, q_a, v)$;
- rejette w si $(\varepsilon, q_0, w) \overset{*}{\vdash} (u, q_r, v)$;
- dans les autres cas on dit que M boucle.

$L(M)$ désigne l'ensemble des mots acceptés par M. ▲

Réductibilité

Définition 5.2. Le langage A se réduit au langage B, noté $A \leq B$, si il existe une fonction calculable

$$f \ : \ \Sigma^* \to \Sigma^*$$

telle que

$$\forall w \in \Sigma^* \ : \ w \in A \Leftrightarrow f(w) \in B.$$

▲

Théorème 5.3 (*transitivité de la réductibilité*). Si $A \leq B$ et $B \leq C$, alors $A \leq C$.

Preuve. La preuve est laissée en exercice. ■

Soit M une MT qui s'arrête sur toutes les entrées possibles.

Comment définir le temps de calcul de M sur un mot w ?

Une définition naturelle est le nombre de transitions avant l'arrêt de M sur entrée w.

Quel sera le temps de calcul de M ?

Définition 5.4. Le temps de calcul de M est la fonction

$$\begin{aligned} f : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto \max_{|w|=n} \{\text{temps de calcul de } M \text{ sur } w\}. \end{aligned}$$

On dira que M fonctionne en temps $f(n)$, ou que sa complexité de temps est $f(n)$. ▲

La classe P

Définition 5.5.

$$\mathsf{TIME}(t(n)) = \{L \mid L \text{ est décidé par une MT en temps } O(t(n))\}.$$

▲

Définition 5.6. La classe de complexité P est définie comme suit :

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k).$$

▲

Remarque 5.7. Si $p(n)$ et $q(n)$ sont des polynômes alors

$$p(n) + q(n),\ p(n)q(n),\ \text{et}\ p(q(n))$$

sont aussi des polynômes. ▲

Remarques 5.8.

- La classe P est robuste par rapport à un changement raisonnable dans le modèle de calcul : les MT avec un ruban, k rubans, k têtes de lecture/écriture, et plusieurs autres modèles définissent la même classe de langages P.
- La classe P correspond aux langages décidables en pratique en un temps raisonnable.

▲

Exemples 5.9. Voici quelques problèmes dans P :

- Décider si un nombre n est premier.
- Décider si $A + B = C$ pour A, B et C des matrices d'entiers.
- Décider si $A \times B = C$ pour A, B et C des matrices d'entiers.
- Décider si il existe un chemin entre deux sommets s et t dans un graphe G dont le coût est moins de c.
- Décider si une liste l est en ordre lexicographique.

- Décider si un graphe G est coloriable avec deux couleurs, de telle sorte que deux sommets adjacents ne seront jamais de la même couleur.
- Décider si une expression booléenne en 2-FNC est satisfaisable.
- Résoudre des problèmes de programmation linéaire.
- Décider si un circuit booléen s'évalue à $\langle$VRAI$\rangle$ pour un input donné.
- Etc.

▲

Dans P ou non ?

Définition 5.10. Un graphe G est k-coloriable s'il est possible d'assigner à chaque sommet de G une couleur choisie parmi k couleurs données, de telle sorte qu'il n'existe aucune paire de sommets adjacents de la même couleur. ▲

Le langage

$$3\text{-}COL = \{\langle G \rangle \mid G \text{ est un graphe 3-coloriable}\}$$

est décidable.

A-t-on $3\text{-}COL \in \mathsf{P}$?

Figure 5.1.

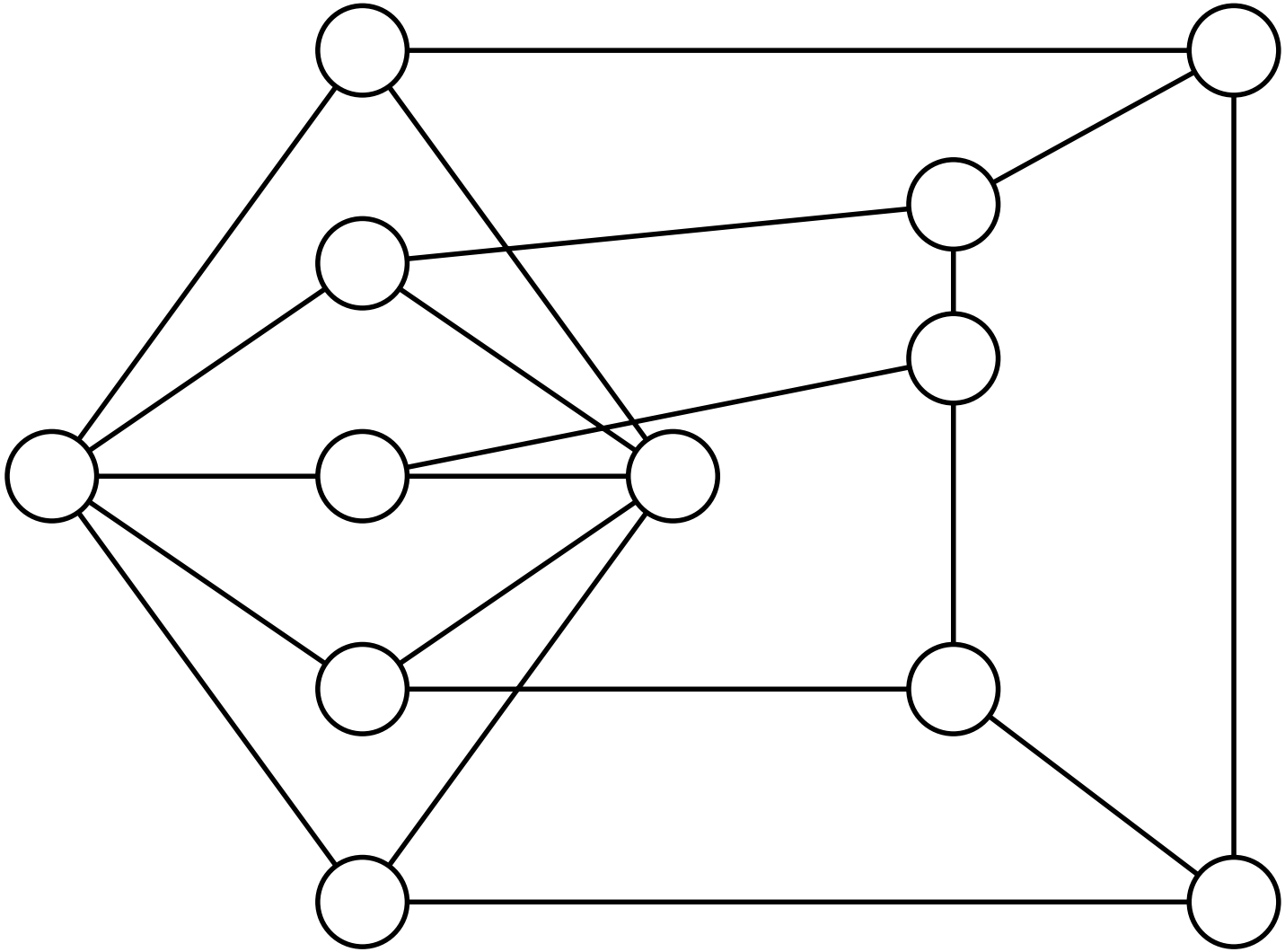


Figure 5.2.

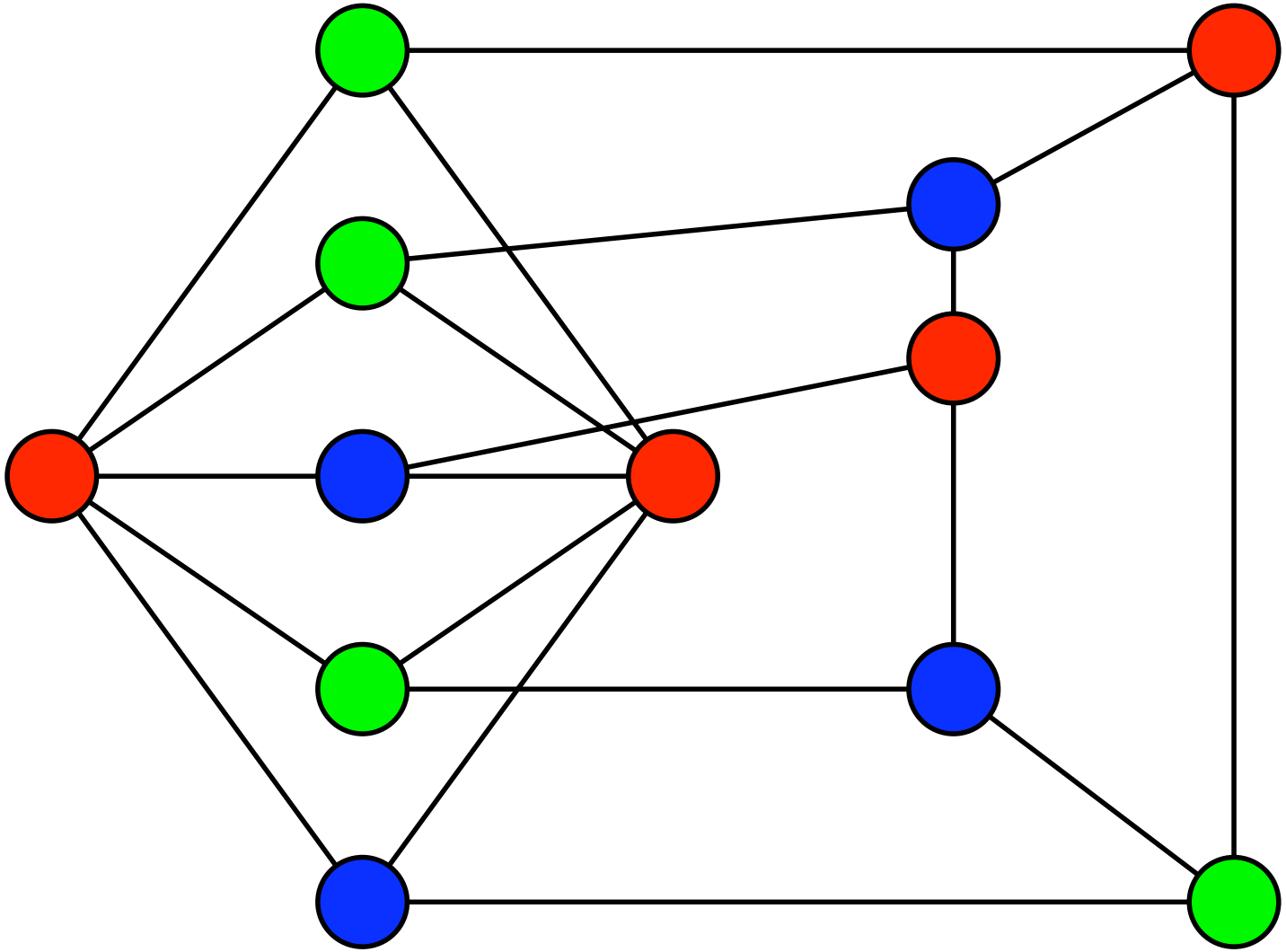


Figure 5.3.

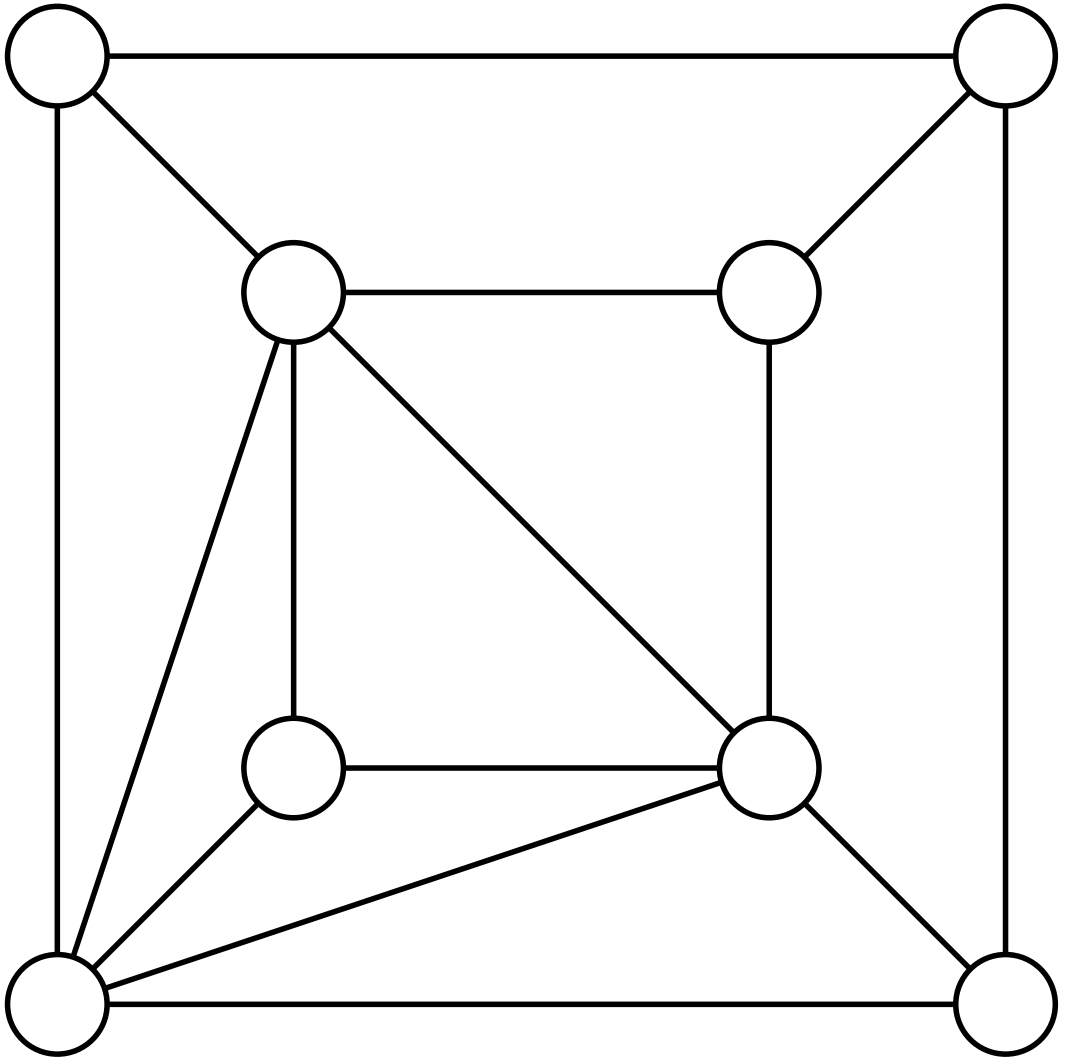
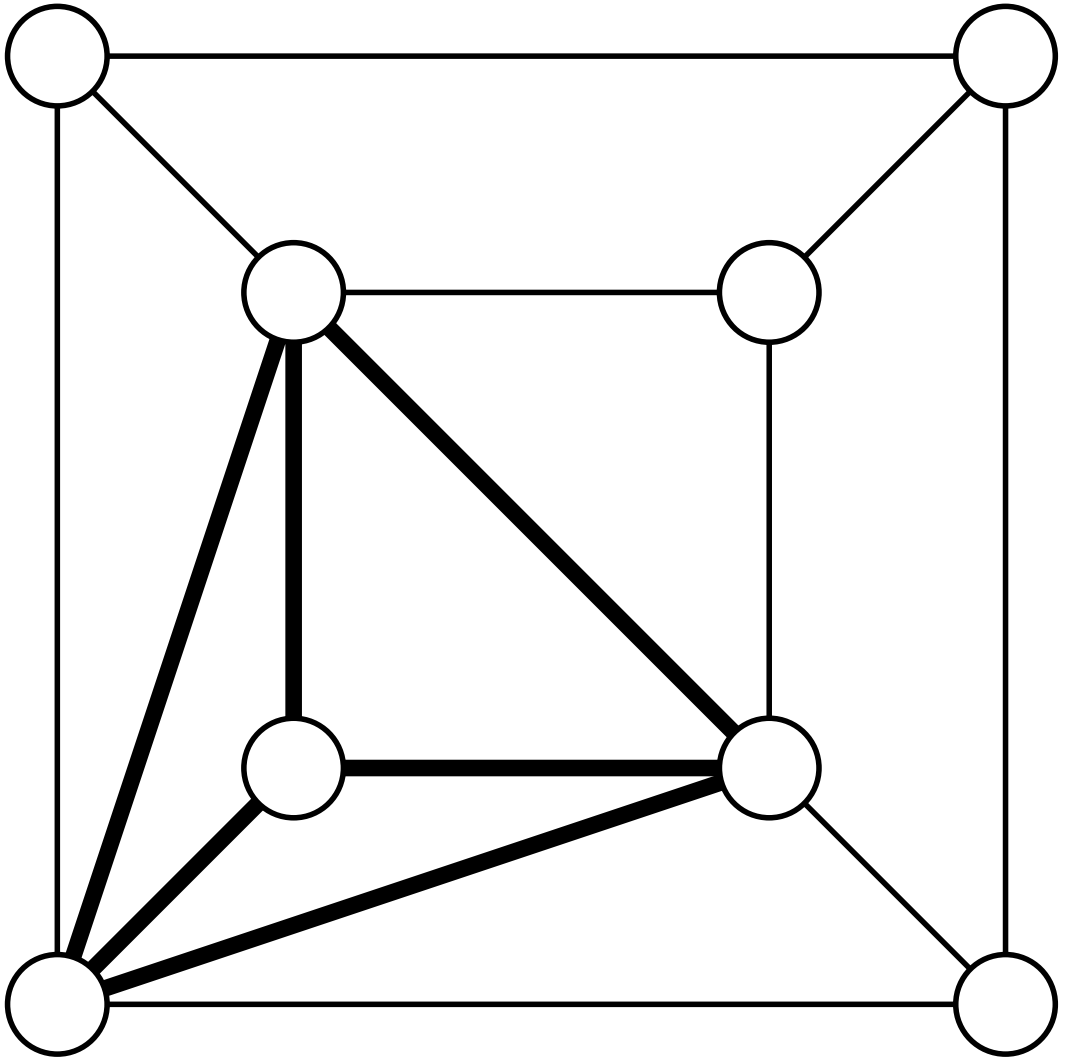


Figure 5.4.



Un algorithme qui décide $3\text{-}COL$ nécessite en pire cas l'examen de 3^s coloriages, où s est le nombre de sommets.

Puisque $n = |\langle G \rangle| = s^2$, le temps de calcul selon cette stratégie est plus grand que $3^{\sqrt{n}}$.

Quel que soit le polynôme $p(n)$, $3^{\sqrt{n}} \notin O(p(n))$, donc cette stratégie n'est pas polynômiale.

Existe-t-il une stratégie différente qui résout $3\text{-}COL$ en temps polynômial ?

La classe NP

Définition 5.11. Un vérificateur polynômial pour un langage L est une MT V telle que pour tout $w \in \Sigma^*$:

- si $w \in L$ alors il existe un mot c tel que $\langle w, c \rangle \in L(V)$;
- si $w \notin L$ alors pour tout c on a $\langle w, c \rangle \notin L(V)$;
- le temps de calcul de V sur $\langle w, c \rangle$ est polynômial en la taille de w.

Un c tel que $\langle w, c \rangle \in L(V)$ est appelé certificat ou preuve ou témoin de l'appartenance de w au langage L. ▲

Définition 5.12. La classe de complexité NP est l'ensemble des langages qui possèdent un vérificateur polynômial. ▲

Définition 5.13.

$$\mathit{COMPOSÉ} = \{\langle n \rangle \mid n \in \mathbb{N} \text{ et } n \text{ est composé}\}.$$

▲

Théorème 5.14.

$$\mathit{COMPOSÉ} \in \mathsf{NP}.$$

Aperçu de la preuve. Pour un nombre composé n, une paire $\langle x, y \rangle$ telle que $1 < x, y < n$ et $xy = n$ peut tenir lieu de certificat. Le vérificateur n'a qu'à multiplier les facteurs et comparer le résultat à n.

Un tel certificat n'existe pas pour un nombre qui n'est pas composé. ■

Théorème 5.15 (*Manindra Agrawal, Neeraj Kayal et Nitin Saxena, 2004*).

$$COMPOSÉ \in \mathsf{P}.$$

Aperçu de la preuve. L'algorithme suivant décide le langage $COMPOSÉ$ dans un temps polynômial en la taille de l'entrée :

Prendre un entier $\langle n \rangle$ en entrée;

si $n \leq 1$, alors rejeter;

si $n = a^b$ pour $a \in \mathbb{N}$ et $b > 1$, alors accepter;

soit r le plus petit entier tel que $\mathrm{ord}_r(n) > \log^2(n)$;

si $1 < \mathrm{PGCD}(a, n) < n$ pour $a \leq r$, alors accepter;

si $n \leq r$, alors rejeter;

pour a de 1 à $\lfloor \sqrt{\phi(r)} \log(n) \rfloor$ faire :

si $(X + a)^n \neq X^n + a$ dans l'anneau $\mathbb{Z}_n[x]/(X^r - 1)$, alors accepter;

rejeter.

■

Théorème 5.16. Le langage $3\text{-}COL$ est dans NP.

Aperçu de la preuve. Le témoin c sera un 3-coloriage des sommets de G, c'est-à-dire qui donne à des sommets adjacents des couleurs différentes.

Comme il existe $n(n+1)/2$ paires de sommets pour un graphe à n sommets, vérifier que le coloriage est correct se fera en temps polynômial. Le vérificateur rejette si deux sommets adjacents ont la même couleur.

Si $w \in 3\text{-}COL$, alors il existe un coloriage c qui fera accepter le vérificateur.

Si $w \notin 3\text{-}COL$ alors tout coloriage c fera rejeter le vérificateur. ■

Définition 5.17. Soient G un graphe orienté et s, t des sommets de G.

Un chemin hamiltonien entre s et t est un chemin orienté qui passe exactement une fois par chaque sommet de G. ▲

Définition 5.18.

$$\begin{aligned} HAMPATH &= \{\langle G, s, t\rangle \mid \text{un chemin hamiltonien existe} \\ &\qquad \text{du sommet } s \text{ au sommet } t \text{ dans le graphe orienté } G\} \end{aligned}$$

▲

Théorème 5.19.

$$HAMPATH \in \mathsf{NP}.$$

Aperçu de la preuve. Le certificat c sera un chemin hamiltonien entre s et t. ■

$\mathsf{P} = \mathsf{NP}\,?$

P : les langages décidables efficacement.

NP : les langages vérifiables efficacement.

Le *Clay Mathematical Institute* offre un prix de un million de dollars à quiconque répondra à la question suivante :

$$\mathsf{P} = \mathsf{NP}?$$

Définition 5.20.

$$\mathsf{EXPTIME} = \bigcup_{k \geq 0} \mathsf{TIME}(2^{n^k}).$$

▲

Théorème 5.21.

$$\mathsf{NP} \subseteq \mathsf{EXPTIME}.$$

Aperçu de la preuve. Soit $L \in \mathsf{NP}$ et V le vérificateur pour L qui fonctionne en temps polynômial $p(n)$.

Un décideur D pour L, sur entrée w, avec $n = |w|$, teste tous les certificats de longueur $\leq p(n)$ en invoquant V à chaque fois. La machine D accepte w si et seulement si un certificat a été trouvé.

Voici une estimation du temps de calcul $t(n)$ pour D où $d = \#\Sigma$:

$$\begin{aligned} t(n) &\approx d^{p(n)}p(n) \\ &\approx 2^{\log_2(d)p(n)+\log_2(p(n))} \\ &\in O(2^{n^k}) \end{aligned}$$

pour un certain k. ■

Définition 5.22.

$$CLIQUE = \{\langle G, k\rangle \mid \text{le graphe } G \text{ contient un sous-graphe complet de taille } k\}$$

▲

Théorème 5.23.

$$CLIQUE \in \mathsf{NP}.$$

Preuve. La preuve est laissée en exercice. ■

Définition 5.24.

$$\begin{aligned} \textit{SUBSET-SUM} \quad = \quad & \{\langle \{x_1, \ldots, x_m\}, t\rangle \mid \ x_1, \ldots, x_m, t \in \mathbb{N}, \\ & \exists \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_m\} \ : \ \sum_{i=1}^{l} y_i = t\} \end{aligned}$$

▲

Théorème 5.25.

$$\textit{SUBSET-SUM} \in \mathsf{NP}.$$

Preuve. La preuve est laissée en exercice. ■

Réductions polynômiales

Définition 5.26. On dit que le langage L_1 se réduit polynômialement à L_2, noté $L_1 \leq_{\mathrm{P}} L_2$ si

$$L_1 \leq L_2$$

à l'aide d'une fonction

$$f : \Sigma^* \to \Sigma^*$$

calculable en temps polynômial. ▲

Exemples 5.27. Nous avons déjà :

- $A_{\mathrm{MT}} \leq_{\mathrm{P}} \overline{VIDE_{\mathrm{MT}}}$,
- $A_{\mathrm{MT}} \leq_{\mathrm{P}} \overline{REG_{\mathrm{MT}}}$,
- $A_{\mathrm{MT}} \leq_{\mathrm{P}} REG_{\mathrm{MT}}$,
- $A_{\mathrm{MT}} \leq_{\mathrm{P}} \overline{TOUT_{\mathrm{GHC}}}$.

▲

Dans chaque cas la réduction est simple et son temps d'exécution est polynômial.

Comme ces problèmes sont indécidables, le fait que les réductions se fassent en temps polynômial est de peu d'intérêt.

Théorème 5.28. Si $L_1 \leq_{\mathrm{P}} L_2$ et $L_2 \in \mathsf{P}$, alors $L_1 \in \mathsf{P}$.

Preuve. Soit la fonction $f : \Sigma^* \to \Sigma^*$ calculable en temps polynômial $p(n)$ telle que $\forall w \in \Sigma^* \ : \ w \in L_1 \Leftrightarrow f(w) \in L_2$.

Soit M_2 une MT qui décide L_2 en temps polynômial $q(n)$.

Considérons la MT M_1 suivante :

Prendre un mot w en entrée;

calculer $w' = f(w)$;

simuler M_2 sur w'.

La machine M_1 décide L_1 et fonctionne en temps $O(q(p(n)))$, donc en temps polynômial. ■

Théorème 5.29. Si $L_1 \leq_{\mathrm{P}} L_2$ et $L_2 \in \mathsf{NP}$, alors $L_1 \in \mathsf{NP}$.

Preuve. Soit la fonction $f : \Sigma^* \to \Sigma^*$ calculable en temps polynômial $p(n)$ telle que $\forall w \in \Sigma^* \ : \ w \in L_1 \Leftrightarrow f(w) \in L_2$.

Soit V_2 un vérificateur pour L_2 de temps polynômial $q(n)$.

Considérons le vérificateur suivant pour L_1 :

Prendre $\langle w, c \rangle$ en entrée;

calculer $w' = f(w)$;

simuler V_2 sur $\langle w', c \rangle$.

Clairement, V_1 fonctionne en temps polynômial.

On a :

$$\begin{aligned} w \in L_1 \quad &\Rightarrow \quad w' = f(w) \in L_2 \\ &\Rightarrow \quad \exists c \ : \ V_2 \text{ accepte } \langle w', c \rangle \\ &\Rightarrow \quad \exists c \ : \ V_1 \text{ accepte } \langle w, c \rangle. \end{aligned}$$

D'autre part :

$$\begin{aligned} w \notin L_1 \quad &\Rightarrow \quad w' = f(w) \notin L_2 \\ &\Rightarrow \quad \forall c \ : \ V_2 \text{ rejette } \langle w', c \rangle \\ &\Rightarrow \quad \forall c \ : \ V_1 \text{ rejette } \langle w, c \rangle. \end{aligned}$$

■

Les langages NP-complets

Définition 5.30. Le langage L est NP-difficile si pour tout $L' \in \mathsf{NP}$ on a $L' \leq_{\mathrm{P}} L$. ▲

Remarque 5.31. Intuitivement, un langage NP-difficile est aussi difficile à décider, ou plus difficile, que n'importe quel langage de NP. ▲

Définition 5.32. Le langage L est NP-complet si

- L est NP-difficile ;
- $L \in \mathsf{NP}$.

La classe des langages NP-complets est notée NPC. ▲

Remarque 5.33. Intuitivement, si L est NP-complet, alors

- L est au moins aussi difficile à décider que n'importe quel langage de NP ;
- L n'est pas trop difficile car il est dans NP ;

▲

Le coeur de la NP-complétude

Théorème 5.34. Si L est un langage NP-complet et $L \in \mathsf{P}$, alors $\mathsf{P} = \mathsf{NP}$.

Preuve. On a trivialement $\mathsf{P} \subseteq \mathsf{NP}$.

Soit $A \in \mathsf{NP}$.

Puisque L est NP-difficile, on a $A \leq_{\mathrm{P}} L$; et $A \in \mathsf{P}$ car $L \in \mathsf{P}$.

D'où : $\mathsf{NP} \subseteq \mathsf{P}$.

On a donc $\mathsf{P} = \mathsf{NP}$. ■

Si l'on peut résoudre un seul problème NP-complet efficacement, alors on aura résolu efficacement tous les problèmes de NP.

La réponse à la question

$$\text{est-ce que } \mathsf{P} = \mathsf{NP} \,?$$

est toujours inconnue, et il s'agit d'un problème central de la théorie de la complexité.

Définition 5.35. Les opérateurs $\neg$, $\wedge$, et $\vee$ qui apparaissent dans un circuit booléen sont appelées des portes. ▲

Définition 5.36. La taille d'un circuit booléen est le nombre de portes qu'il contient. ▲

Le modèle de calcul des circuits booléens est raisonnable et, selon la thèse de Church-Turing, ce modèle doit être équivalent au modèle des MT.

Théorème 5.37. Pour une MT $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ qui fonctionne en temps polynômial $p(n)$ où n est la longueur de son entrée, il existe une famille de circuits booléens $\{C_n \mid n \geq 0\}$ de tailles polynômiales par rapport à n, telle que sur entrée $(w_1, \ldots, w_n)$, C_n retourne $\langle \text{VRAI} \rangle$ si, et seulement si M accepte $w = w_1 \ldots w_n$.

De plus, ces circuits peuvent être construits en temps polynômial.

Preuve. Soit $n \geq 0$.

Nous allons montrer qu'on peut efficacement construire un circuit booléen C_n qui retourne $\langle \text{VRAI} \rangle$ lorsque M accepte, et qui retourne $\langle \text{FAUX} \rangle$ lorsque M rejette.

La taille de C_n sera polynômiale par rapport à n.

Nous supposons, sans perte de généralité, que

$$\Sigma = \{\mathtt{0}, \mathtt{1}\}, \quad \Gamma = \{\mathtt{0}, \mathtt{1}, \sqcup\}, \quad \#Q = l.$$

Rappelons que le temps de calcul de M est $p(n)$.

Considérons un exemple de trace d'un calcul fait par M :

$q_0$0 0 1 1 ⊔ ⊔

1 $q_4$0 1 1 ⊔ ⊔

1 0 $q_6$1 1 ⊔ ⊔

1 $q_3$0 1 1 ⊔ ⊔

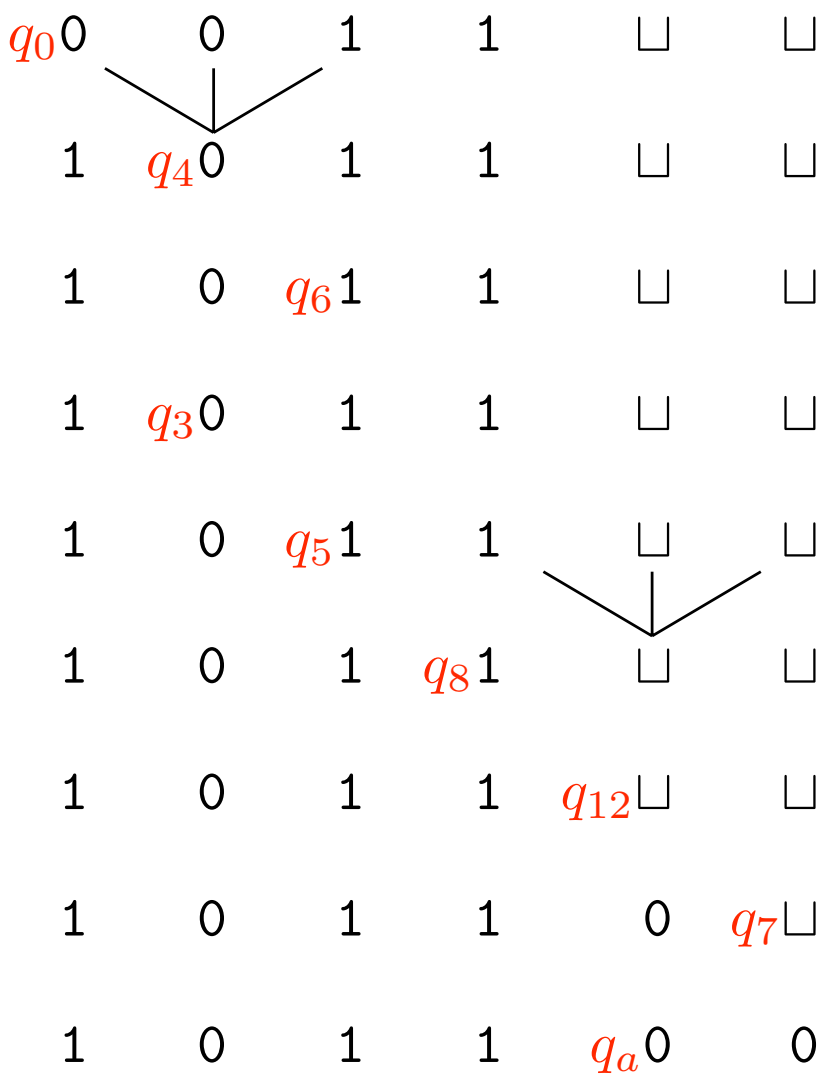
1 0 $q_5$1 1 ⊔ ⊔

1 0 1 $q_8$1 ⊔ ⊔

1 0 1 1 q_{12}⊔ ⊔

1 0 1 1 0 q_7⊔

1 0 1 1 q_a0 0


q_0 0 0 1 1 ⊔ ⊔
1 q_4 0 1 1 ⊔ ⊔
1 0 q_6 1 1 ⊔ ⊔
1 q_3 0 1 1 ⊔ ⊔
1 0 q_5 1 1 ⊔ ⊔
1 0 1 q_8 1 ⊔ ⊔
1 0 1 1 q_{12} ⊔ ⊔
1 0 1 1 0 q_7 ⊔
1 0 1 1 q_a 0 0

Nous allons définir plusieurs tableaux de variables booléennes dont les valeurs vont correspondre à une trace d'exécution de M.

Chaque tableau de variables aura un indice $1 \leq i \leq p(n)$ et $1 \leq j \leq p(n)$.

L'indice i représente le temps, c'est à dire le numéro de la ligne dans la trace de M.

L'indice j correspond à la position horizontale dans la trace.

Les tableaux de variables sont les suivants :

- $ZÉRO(i, j) = \langle \text{VRAI} \rangle$ si, et seulement si la case j au temps i a comme valeur `0`.
- $UN(i, j) = \langle \text{VRAI} \rangle$ si, et seulement si la case j au temps i a comme valeur `1`.
- $BLANC(i, j) = \langle \text{VRAI} \rangle$ si, et seulement si la case j au temps i a comme valeur $\sqcup$.
- $ÉTAT(q, i, j) = \langle \text{VRAI} \rangle$ si, et seulement si l'état de M est q au temps i et la tête est à la case j.

Le nombre de variables est donc $3p(n)^2 + p(n)^2 l$.

Au temps $i = 1$ du calcul, pour $j = 1$ à n, on a :

$$\begin{aligned} ZÉRO(1,j) &= \neg w_j, \\ UN(1,j) &= w_j, \\ BLANC(1,j) &= \langle \textsc{Faux} \rangle; \end{aligned}$$

et pour $j = n + 1$ à $p(n)$ on a :

$$\begin{aligned} ZÉRO(1,j) &= \langle \textsc{Faux} \rangle, \\ UN(1,j) &= \langle \textsc{Faux} \rangle, \\ BLANC(1,j) &= \langle \textsc{Vrai} \rangle; \end{aligned}$$

enfin on a :

$$ÉTAT(q,1,j) = \begin{cases} \langle \textsc{Vrai} \rangle & \text{si} \quad q = q_0 \text{ et } j = 1, \\ \langle \textsc{Faux} \rangle & \text{si} \quad \text{sinon.} \end{cases}$$

Au temps $2 \leq i \leq p(n)$, et à la position j, chacune des valeurs des $l+3$ variables

$$ZÉRO(i,j), \quad UN(i,j), \quad BLANC(i,j), \quad ÉTAT(q,i,j) \text{ pour } q \in Q$$

est une fonction booléenne des $3l+9$ variables

$$ZÉRO(i-1,j-1), \quad ZÉRO(i-1,j), \quad ZÉRO(i-1,j+1),$$
$$UN(i-1,j-1), \quad UN(i-1,j), \quad UN(i-1,j+1),$$
$$BLANC(i-1,j-1), \quad BLANC(i-1,j), \quad BLANC(i-1,j+1),$$
$$ÉTAT(q',i-1,j-1), \quad ÉTAT(q',i-1,j),$$
$$ÉTAT(q',i-1,j+1) \text{ pour } q' \in Q$$

déterminée par la fonction de transition δ de M.

Cette fonction booléenne correspond à une équation du circuit.

Comme chaque variable ne dépend que d'un nombre constant de variables, la taille de ces équations est bornée par une constante.

Le nombre total d'équations est donc dans $O(p(n)^2)$.

Il ne reste qu'à ajouter une dernière variable qui sera $\langle$VRAI$\rangle$ si, et seulement si M accepte.

D'où :

$$ACCEPTE = \bigvee_{\substack{i \geq 1 \\ j \geq 1}} \acute{E}TAT(q_a, i, j).$$

■

Définition 5.38. Un circuit booléen est satisfaisable si il existe une affectation de ses variables d'input qui le rende vrai. ▲

Exemple 5.39. Le circuit

$$\begin{aligned} z_1 &= x_1 \vee x_2 \\ z_2 &= \neg x_1 \wedge x_3 \\ z_3 &= z_1 \wedge z_2 \end{aligned}$$

est satisfaisable par

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 1.$$

▲

Exemple 5.40. Le circuit

$$\begin{aligned} z_1 &= x_1 \wedge x_2 \\ z_2 &= \neg x_1 \wedge x_3 \\ z_3 &= z_1 \wedge z_2 \end{aligned}$$

n'est pas satisfaisable car

$$\begin{aligned} z_3 &= z_1 \wedge z_2 \\ &= (x_1 \wedge x_2) \wedge (\neg x_1 \wedge x_3) \\ &= x_1 \wedge \neg x_1 \wedge x_2 \wedge x_3 \\ &= 0 \wedge x_2 \wedge x_3 \\ &= 0 \end{aligned}$$

▲

Il semble difficile de décider si un circuit est satisfaisable.

On ne connaît pas de meilleure approche que d'essayer toutes les affectations possibles, ce qui se fait en temps exponentiel.

Définition 5.41.

$$SAT\text{-}C = \{\langle C \rangle \mid C \text{ est un circuit booléen satisfaisable}\}.$$

▲

Théorème 5.42.

$$SAT\text{-}C \in \mathsf{NP}.$$

Preuve. Un certificat est une affectation des variables qui rend le circuit vrai.

L'évaluation du circuit se fait dans un temps linéaire par rapport à sa taille. ■

Théorème 5.43 (*Cook-Levin*). $SAT\text{-}C$ est NP-complet.

Preuve. On sait que $SAT\text{-}C \in \mathsf{NP}$.

Il reste à montrer :

$$\forall L \in \mathsf{NP} \ : \ L \leq_{\mathrm{P}} SAT\text{-}C.$$

Soit $L \in \mathsf{NP}$ et soit V son vérificateur de temps polynômial.

Soit la fonction

$$f \ : \ \Sigma^* \to \Sigma^*$$

calculée par la MT suivante :

Prendre un mot w de taille n en entrée;

construire le circuit booléen C associé à V pour la taille n;

construire le circuit C' à partir de C en remplaçant les variables d'entrée

correspondant aux n premiers bits par les valeurs $w_1, \ldots, w_n$,

il ne reste dans C' que les variables d'entrée correspondant

aux bits du certificat;

produire $\langle C' \rangle$ en sortie.

La fonction f se calcule en temps polynômial.

De plus, on a :

$$\begin{aligned} w \in L \quad &\Leftrightarrow \quad \exists \text{ un certificat } c \ : \ V \text{ accepte } \langle w, c \rangle \\ &\Leftrightarrow \quad C' \text{ est satisfaisable} \\ &\Leftrightarrow \quad \langle C' \rangle \in SAT\text{-}C. \end{aligned}$$

■

Il est donc aussi difficile de trouver un algorithme polynômial pour $SAT\text{-}C$ que de trouver un algorithme polynômial pour n'importe quel langage de NP !

Un algorithme polynômial pour $SAT\text{-}C$ fournirait automatiquement :

- un algorithme polynômial pour $3\text{-}COL$,
- un algorithme polynômial pour $HAMPATH$,
- un algorithme polynômial pour $CLIQUE$,
- etc.

$SAT\text{-}C$ est-il le seul problème ayant cette propriété ?

Le langage $SAT\text{-}C$ est particulier puisqu'un circuit peut simuler n'importe quel calcul.

Existe-t-il d'autres problèmes dans NP qui soient NP-complets ?

Théorème 5.44. Soit L_1 un langage NP-difficile. Si $L_2 \in \mathsf{NP}$ et $L_1 \leq_{\mathrm{P}} L_2$, alors L_2 est NP-complet.

Preuve. Soit $L \in \mathsf{NP}$.

$$
\begin{aligned}
L \in \mathsf{NP} \text{ et } L_1 \text{ est NP-difficile} &\Rightarrow L \leq_{\mathrm{P}} L_1 \\
L \leq_{\mathrm{P}} L_1 \text{ et } L_1 \leq_{\mathrm{P}} L_2 &\Rightarrow L \leq_{\mathrm{P}} L_2 \\
L \leq_{\mathrm{P}} L_2 &\Rightarrow L_2 \text{ est NP-difficile} \\
L_2 \in \mathsf{NP} \text{ et } L_2 \text{ est NP-difficile} &\Rightarrow L_2 \text{ est NP-complet.}
\end{aligned}
$$

■

Définition 5.45.

$$SAT = \{\langle E \rangle \mid E \text{ est une expression booléenne satisfaisable}\}.$$

▲

Théorème 5.46.

$$SAT \in \mathsf{NP}.$$

Preuve. Un certificat est une affectation des variables qui rend l'expression vraie.

La vérification se fait dans un temps linéaire par rapport à la taille de l'expression. ■

Théorème 5.47. SAT est NP-complet.

Preuve. Nous allons montrer que $SAT\text{-}C \leq_{\mathrm{P}} SAT$.

Soit la fonction

$$f \ : \ \Sigma^* \to \Sigma^*$$

telle que

- si y n'est pas de la forme $\langle C \rangle$ où C est un circuit booléen, alors $f(y) = \varepsilon$;
- si $y = \langle C \rangle$, alors $f(y) = \langle E \rangle$ où E est l'expression booléenne décrite ci-dessous.

Le circuit C est une liste d'équations de la forme

$$x_i = E_i$$

pour i allant de 1 à l, et x_l est la dernière variable, c'est-à-dire celle qui donne la valeur du circuit.

Pour chaque i, soit :

$$E'_i = (x_i \wedge E_i) \vee (\neg x_i \wedge \neg E_i),$$

et

$$E = x_l \wedge E'_1 \wedge E'_2 \wedge \ldots \wedge E'_l.$$

Il est facile de voir que la fonction f se calcule en temps polynômial.

Supposons que $\langle C \rangle \in SAT\text{-}C$ et considérons une affectation des variables d'entrée qui rend le circuit vrai. On a :

$$\begin{aligned}
E &= x_l \wedge E'_1 \wedge \ldots \wedge E'_l \\
&= \langle \textsc{Vrai} \rangle \wedge \bigwedge_{1 \leq i \leq l} ((x_i \wedge E_i) \vee (\neg x_i \wedge \neg E_i)) \\
&= \langle \textsc{Vrai} \rangle \wedge \bigwedge_{1 \leq i \leq l} ((x_i \wedge x_i) \vee (\neg x_i \wedge \neg x_i)) \\
&= \langle \textsc{Vrai} \rangle \wedge \bigwedge_{1 \leq i \leq l} (x_i \vee \neg x_i) \\
&= \langle \textsc{Vrai} \rangle \wedge \bigwedge_{1 \leq i \leq l} \langle \textsc{Vrai} \rangle \\
&= \langle \textsc{Vrai} \rangle,
\end{aligned}$$

donc $\langle E \rangle \in SAT$.

Supposons que $\langle E \rangle \in SAT$ et considérons une afectation pour E qui donne $E = \langle \text{VRAI} \rangle$. On a :

$$\begin{aligned} E = x_l \wedge E'_1 \wedge \ldots \wedge E'_l \quad &\Rightarrow \quad \langle \text{VRAI} \rangle = x_l \wedge E'_1 \wedge \ldots \wedge E'_l \\ &\Rightarrow \quad \langle \text{VRAI} \rangle = x_l \wedge \bigwedge_{1 \leq i \leq l} ((x_i \wedge E_i) \vee (\neg x_i \wedge \neg E_i)) \\ &\Rightarrow \quad x_l = \langle \text{VRAI} \rangle,\ x_1 = E_1,\ \ldots,\ x_l = E_l, \end{aligned}$$

donc l'affectation rend le circuit C vrai et $\langle C \rangle \in SAT\text{-}C$.

D'où :

$$\langle C \rangle \in SAT\text{-}C \ \Leftrightarrow \ f(\langle C \rangle) = \langle E \rangle \in SAT.$$

■

Définitions 5.48. Un terme est soit une variable booléenne ou la négation d'une variable booléenne.

Une clause est une somme booléenne de termes.

Une expression booléenne est en forme normale conjonctive (FNC) si il s'agit d'un produit booléen de clauses. ▲

Exemple 5.49. L'expression booléenne suivante est en FNC :

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

▲

Définition 5.50.

$$SAT\text{-}FNC = \{\langle E \rangle \mid E \text{ est une expression booléenne en FNC satisfaisable}\}.$$

▲

Théorème 5.51.

$$SAT\text{-}FNC \in \mathsf{NP}.$$

Preuve. La preuve est la même que celle du théorème 5.46. ■

Théorème 5.52. $SAT\text{-}FNC$ est NP-complet.

Preuve. Nous allons montrer que $SAT \leq_{\mathrm{P}} SAT\text{-}FNC$.

Soit la fonction

$$f \ : \ \Sigma^* \to \Sigma^*$$

telle que

- si y n'est pas de la forme $\langle E \rangle$ où E est une expression booléenne, alors $f(y) = \varepsilon$;
- si $y = \langle E \rangle$, alors $f(y) = \langle E' \rangle$ où E' est l'expression booléenne en FNC dont la construction, à partir de E, est décrite ci-après.

On peut considérer l'expression E comme un arbre syntaxique dont les noeuds contiennent des opérateurs booléens et les feuilles sont des termes, comme dans l'exemple 5.53.

On appliquant la loi de de Morgan autant de fois que nécessaire, du haut de l'arbre vers le bas, on peut faire descendre les négations ($\neg$) jusqu'aux feuilles de l'arbre.

Ensuite, en procédant du bas de l'arbre vers le haut, et en supposant que les sous-expressions E_1 et E_2 sont déjà en FNC, on transforme toute disjonction

$$E_1 \vee E_2$$

en une conjonction

$$E_1^y \wedge E_2^{\neg y},$$

où E_i^α est l'expression obtenue de E_i en ajoutant le terme α à toutes les clauses de E_i.

On peut remarquer que si on choisit $y = \langle \text{VRAI} \rangle$, alors $E_1^y \wedge E_2^{\neg y}$ est logiquement équivalent à E_2 ; et si on choisit $y = \langle \text{FAUX} \rangle$, alors $E_1^y \wedge E_2^{\neg y}$ est logiquement équivalent à E_1 ; donc $E_1 \vee E_2$ est satisfaisable si, et seulement si $E_1^y \wedge E_2^{\neg y}$ est satisfaisable.

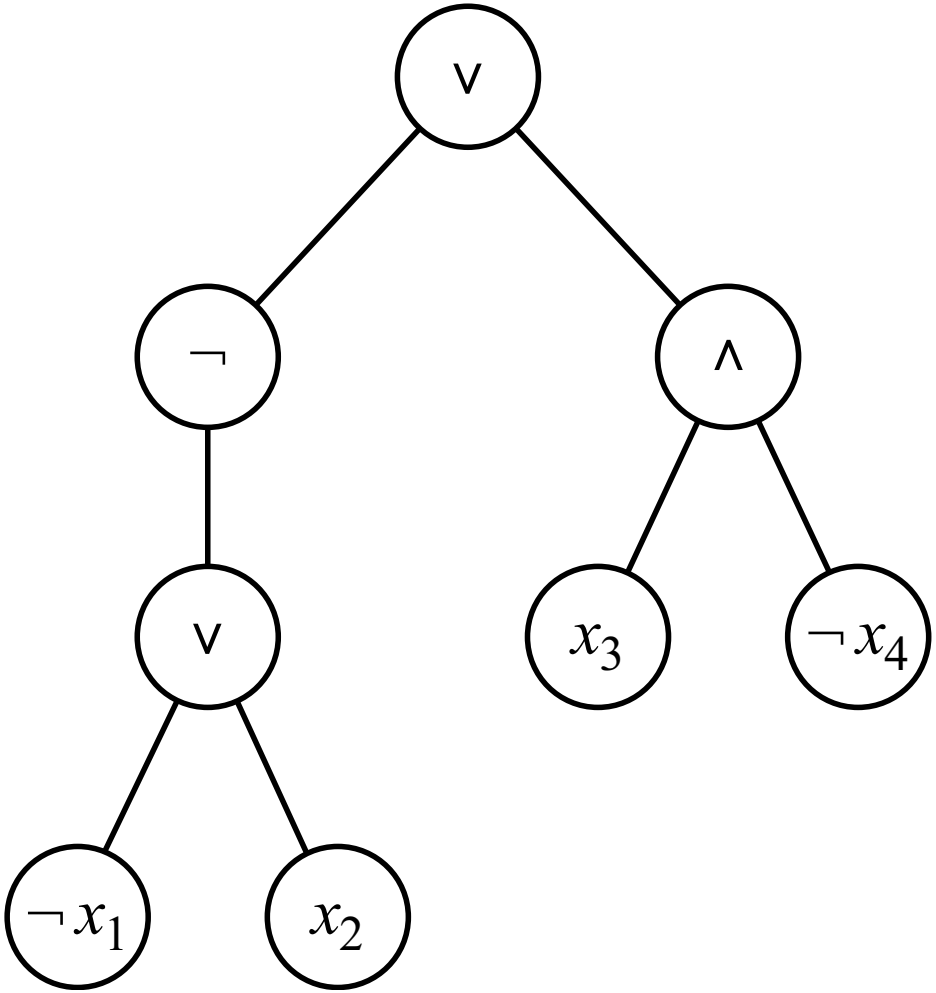
De plus, l'expression résultante est en FNC.

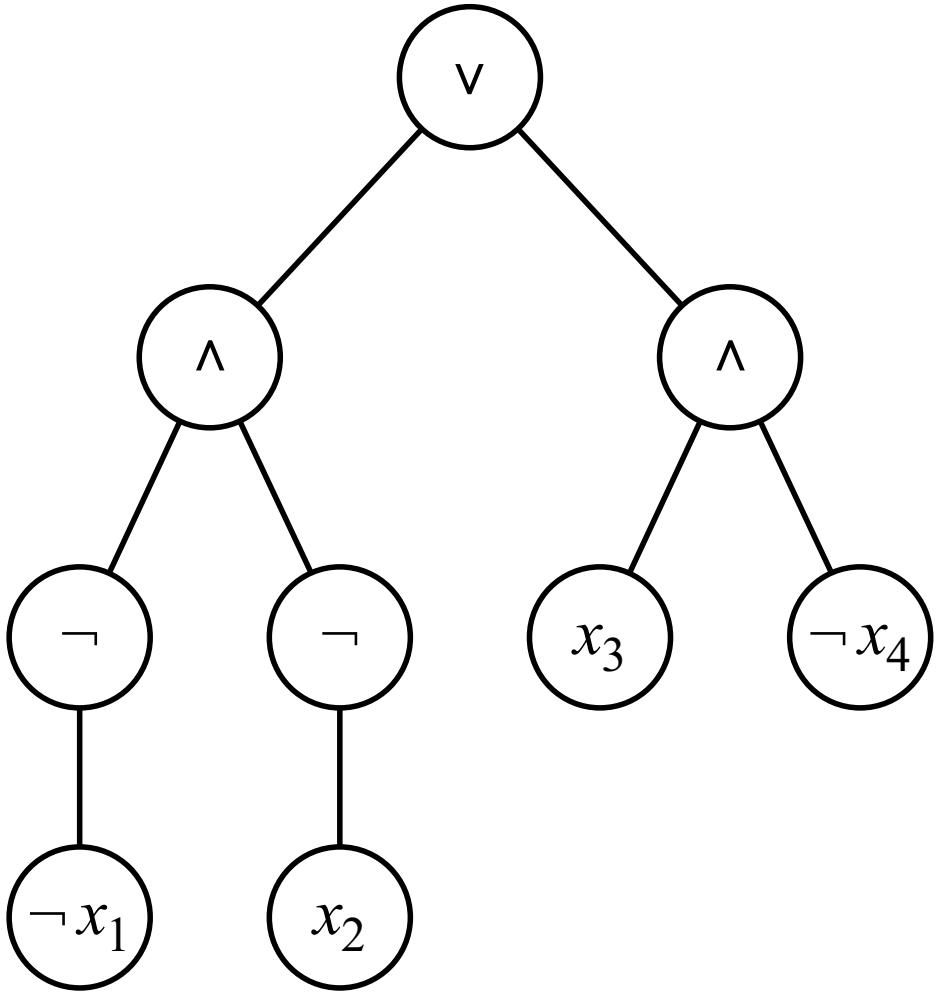
Lorsque le sommet de l'arbre a été traité, l'expression résultante est en FNC.

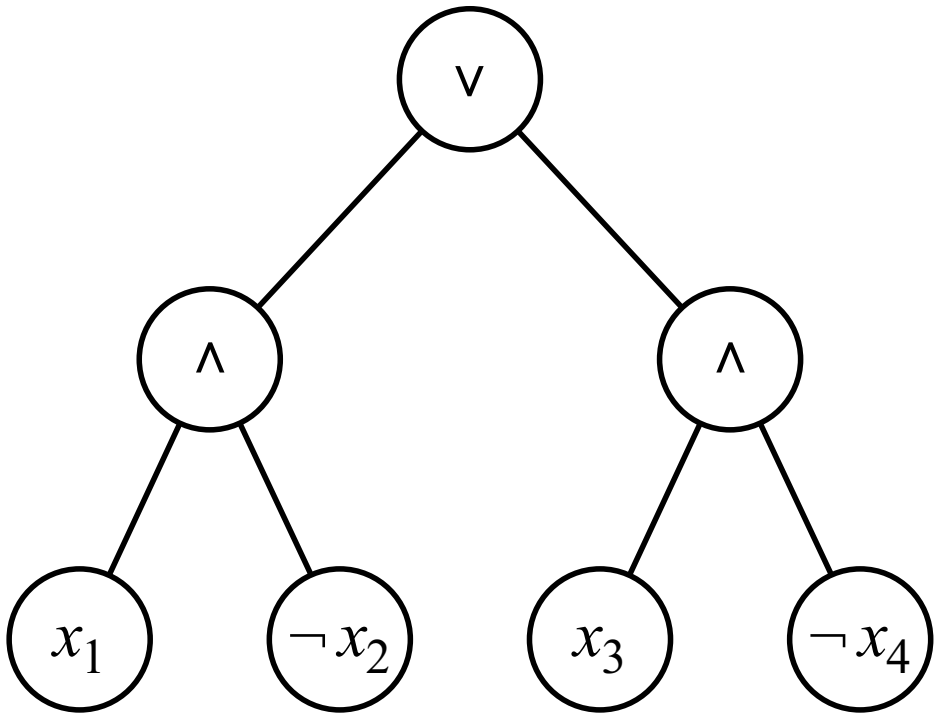
La procédure est de temps polynômial en la taille de l'expression donnée. ■

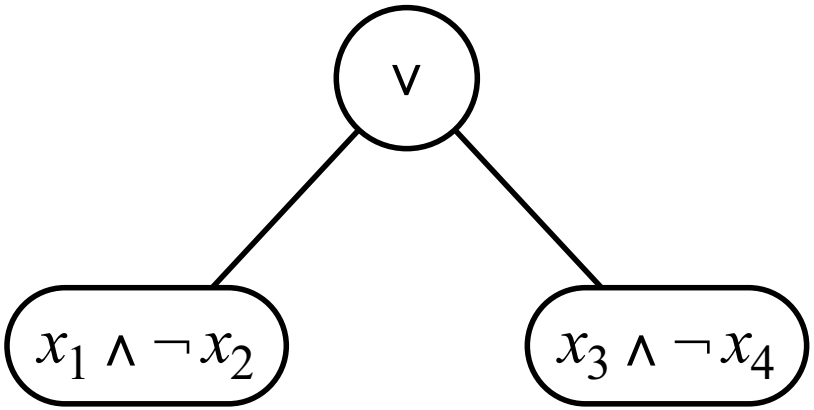
Exemple 5.53.

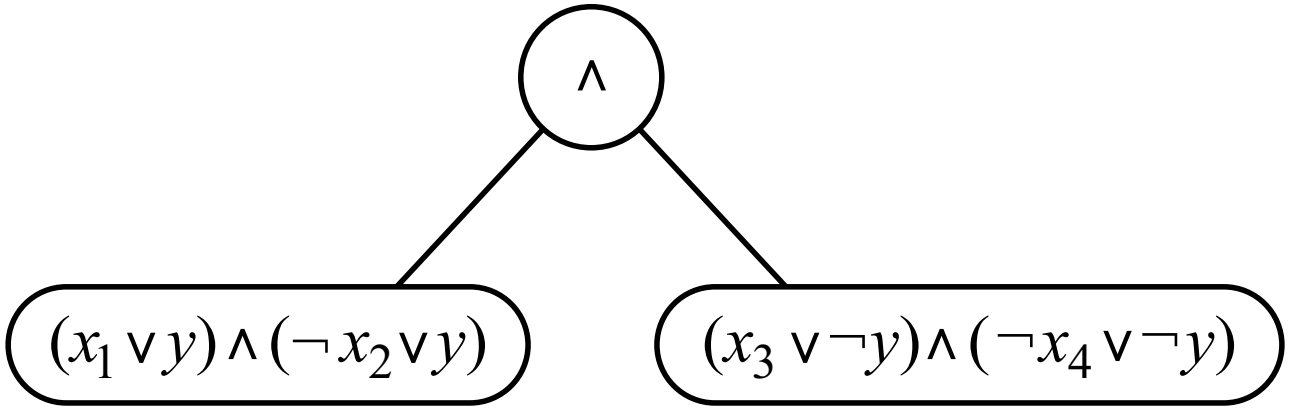
$$\neg(\neg x_1 \vee x_2) \vee (x_3 \wedge \neg x_4)$$




∨
∧
∧
¬
¬
x_3
¬ x_4
¬ x_1
x_2


∨
∧
∧
x_1
$\neg x_2$
x_3
$\neg x_4$


∨
$x_1 \wedge \neg x_2$
$x_3 \wedge \neg x_4$



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

▲

Définition 5.54. Une expression booléenne est en 3-FNC si elle est en FNC et si chaque clause est une somme booléenne d'exactement 3 termes qui comprennent 3 variables distinctes. ▲

Exemple 5.55. L'expression booléenne suivante est en 3-FNC :

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

▲

Définition 5.56.

$$3\text{-}SAT = \{\langle \phi \rangle \mid \phi \text{ est une expression en 3-FNC qui est satisfaisable}\}.$$

▲

Théorème 5.57.

$$3\text{-}SAT \in \mathsf{NP}.$$

Preuve. La preuve est la même que celle du théorème 5.46. ■

Théorème 5.58. $3\text{-}SAT$ est NP-complet.

Aperçu de la preuve. On peut montrer que $SAT\text{-}FNC \leq_{\mathrm{P}} 3\text{-}SAT$.

La fonction de réduction transforme chaque clause d'un exemplaire de $SAT\text{-}FNC$ comme suit :

$$
\begin{aligned}
(t) &\mapsto (t \vee y \vee z) \wedge (t \vee y \vee \neg z) \wedge (t \vee \neg y \vee z) \wedge (t \vee \neg y \vee \neg z) \\
(t_1 \vee t_2) &\mapsto (t_1 \vee t_2 \vee y) \wedge (t_1 \vee t_2 \vee \neg y) \\
(t_1 \vee t_2 \vee t_3) &\mapsto (t_1 \vee t_2 \vee t_3) \\
(t_1 \vee t_2 \vee \ldots \vee t_k) &\mapsto (t_1 \vee t_2 \vee y_1) \wedge (\neg y_1 \vee t_3 \vee y_2) \wedge (\neg y_2 \vee t_4 \vee y_3) \\
&\quad \wedge \ldots \wedge \\
&\quad (\neg y_{k-4} \vee t_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee t_{k-1} \vee t_k)
\end{aligned}
$$

■

Le langage $CLIQUE$

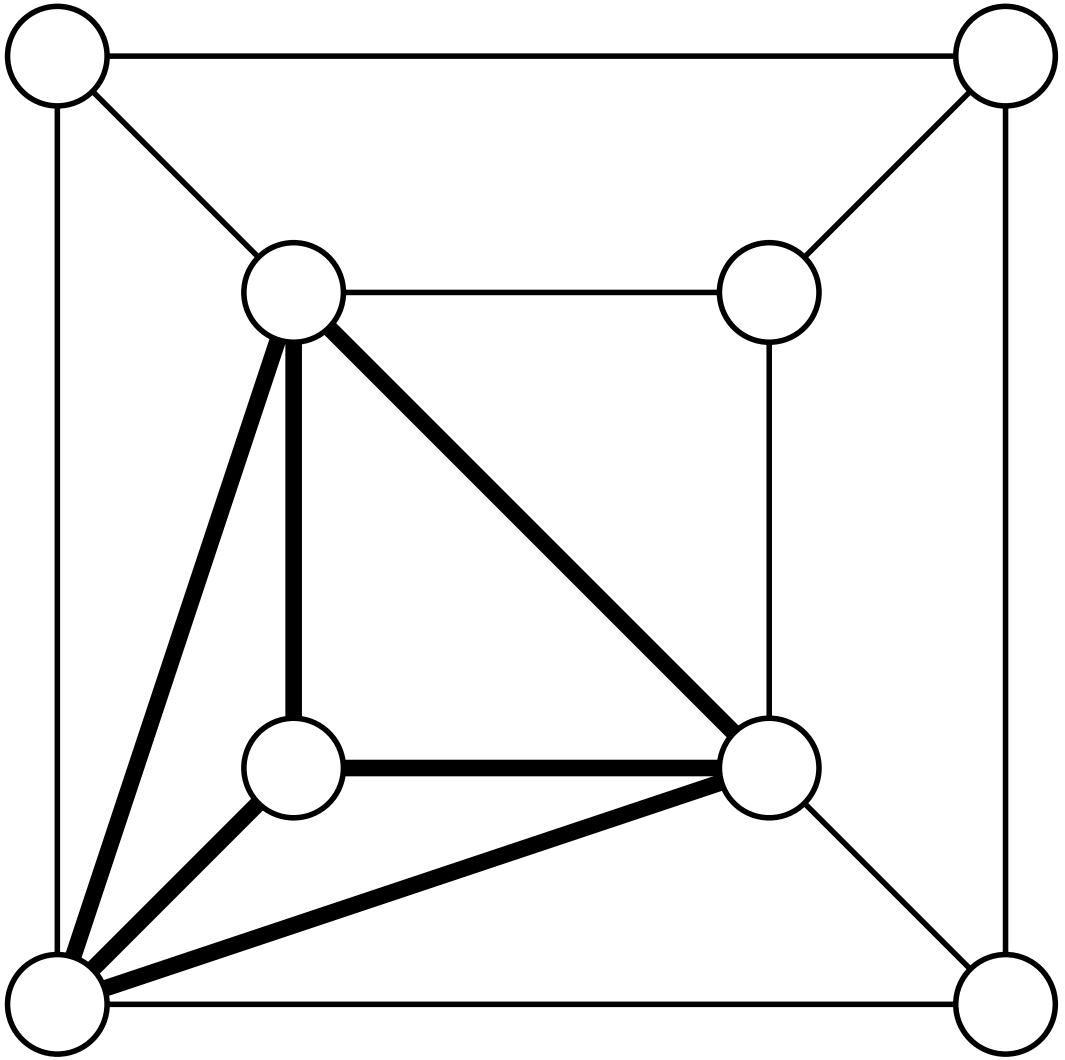
Définition 5.59.

$CLIQUE = \{\langle G, k\rangle \mid k \geq 1$ et

G est un graphe qui contient un sous-graphe complet de taille $k\}$.

▲

Figure 5.5.



Théorème 5.60. $CLIQUE$ est NP-complet.

Preuve. Il est facile de voir que $CLIQUE \in \mathsf{NP}$.

Nous allons montrer que $3\text{-}SAT \leq_{\mathrm{P}} CLIQUE$.

Pour une expression booléenne en 3-FNC

$$(t_{1,1} \vee t_{1,2} \vee t_{1,3}) \wedge (t_{2,1} \vee t_{2,2} \vee t_{2,3}) \wedge \ldots \wedge (t_{k,1} \vee t_{k,2} \vee t_{k,3}),$$

la fonction de réduction retourne $\langle G, k \rangle$ tel que

- les $3k$ sommets de G correspondent aux termes $t_{i,j}$;
- il y a une arête entre le sommet t_{i_1,j_1} et le sommet t_{i_2,j_2} si, et seulement si

$$i_1 \neq i_2 \quad \text{et} \quad t_{i_1,j_1} \neq \neg t_{i_2,j_2}.$$

Voir l'exemple 5.61.

Soit

$$\phi = (t_{1,1} \vee t_{1,2} \vee t_{1,3}) \wedge \ldots \wedge (t_{k,1} \vee t_{k,2} \vee t_{k,3})$$

où $t_{i,j}$ est de la forme $x_{i,j}$ ou $\neg x_{i,j}$ pour une variable booléenne $x_{i,j}$.

Supposons que

$$\langle \phi \rangle \in \mathit{3\text{-}SAT}$$

et considérons une affectation des variables qui rende ϕ vrai.

Dans chaque clause $t_{i,1} \vee t_{i,2} \vee t_{i,3}$ on choisit un terme t_{i,j_i} qui est vrai.

L'ensemble des sommets correspondant à

$$\{t_{1,j_1},\ t_{2,j_2},\ \ldots,\ t_{k,j_k}\}$$

forme une clique parce qu'on ne peut pas trouver dans cet ensemble une variable et sa négation.

Donc $f(\langle \phi \rangle) \in \mathit{CLIQUE}$.

D'autre part, supposons que

$$f(\langle\phi\rangle) = \langle G, k\rangle \in CLIQUE.$$

et considérons une clique de k sommets $C = \{s_1, s_2, \ldots, s_k\}$ dans G.

Comme des sommets correspondant à des termes d'une même clause ne sont jamais reliés par une arête, alors on a exactement un sommet s_i par clause.

De plus, deux sommets correspondant à une variable et à sa négation ne sont jamais reliés dans G, et donc une variable et sa négation ne peuvent pas être représentées dans C.

En choisissant comme affectation $\langle$VRAI$\rangle$ si s_i correspond à une variable, et $\langle$FAUX$\rangle$ si s_i correspond à une variable niée, on rend tous les s_i vrais, et ϕ aussi devient vrai.

Donc $\langle\phi\rangle \in 3\text{-}SAT$.

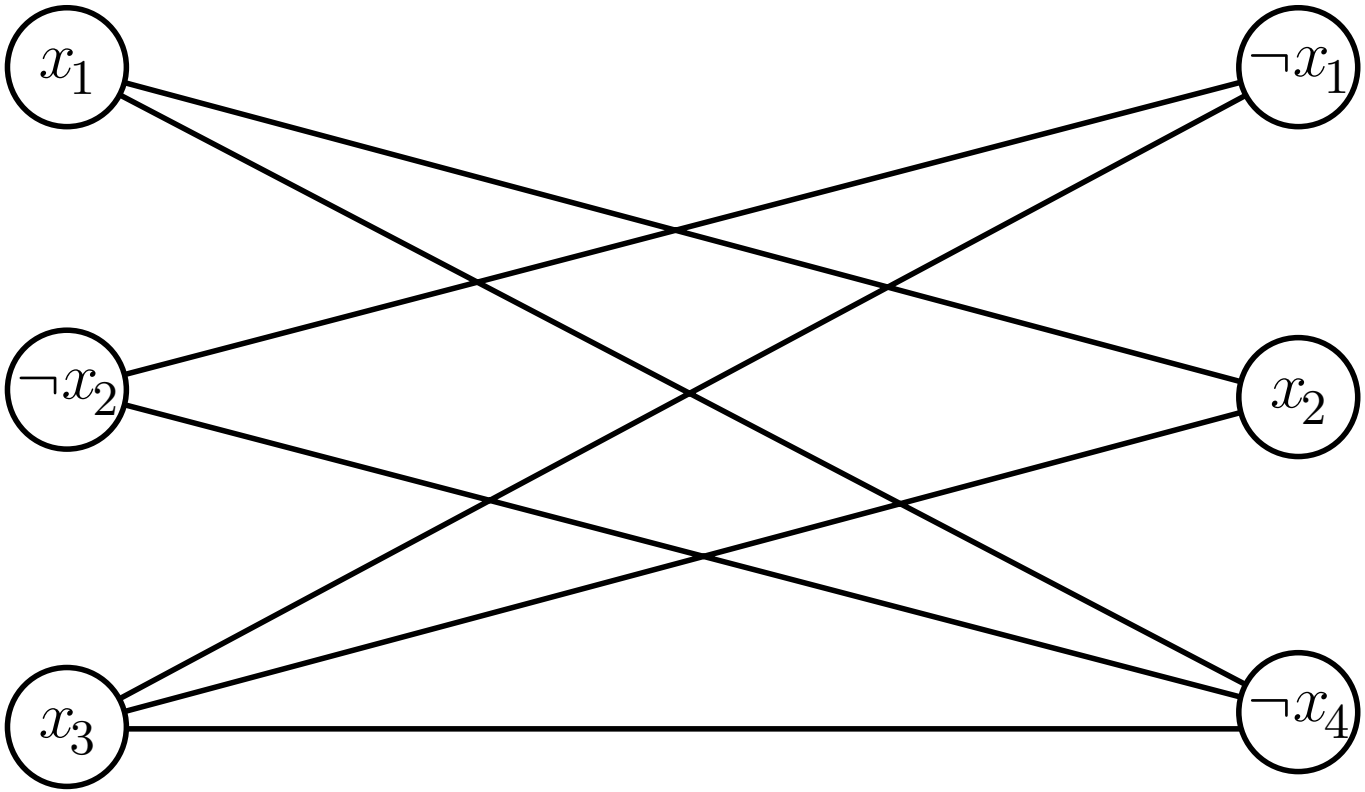
D'où :

$$\langle \phi \rangle \in \textit{3-SAT} \;\Leftrightarrow\; f(\langle \phi \rangle) \in CLIQUE.$$

De plus, la transformation se fait en temps polynômial. ■

Exemple 5.61.

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



▲

Le langage $COUVERTURE$

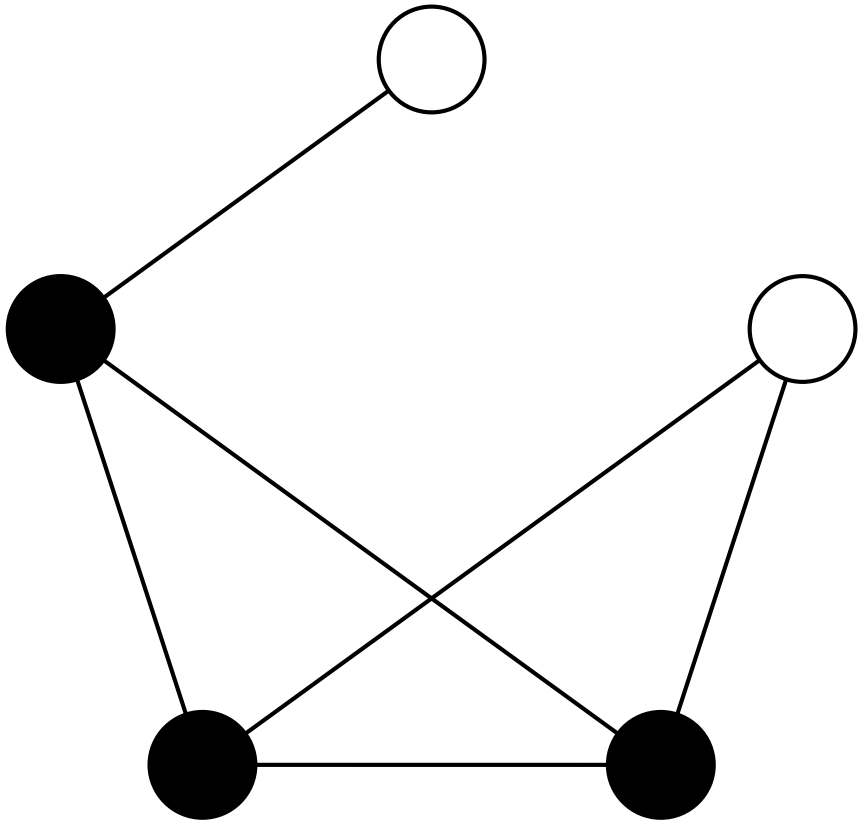
Définition 5.62. Dans un graphe G, une couverture de sommets est un sous-ensemble des sommets de G tel que toute arête touche un sommet de la couverture. ▲

Définition 5.63.

$$COUVERTURE = \{\langle G, s\rangle \mid \text{le graphe } G \text{ possède une couverture de sommets de taille } s\}.$$

▲

Exemple 5.64.



$\langle G, 3\rangle \in COUVERTURE.$

▲

Définition 5.65. Le complément d'un graphe G est le graphe dont les sommets sont ceux de G et dont l'ensemble des arêtes est le complément de l'ensemble des arêtes de G. ▲

Théorème 5.66. $COUVERTURE$ est NP-complet.

Aperçu de la preuve. On obtient $CLIQUE \leq_{\mathrm{P}} COUVERTURE$ en montrant qu'il existe une clique de taille k dans un graphe G à n sommets si, et seulement si il existe une couverture de taille $n - k$ dans le complément de G.

Les détails de cette preuve sont laissés en exercice. ■

Remarque 5.67. La preuve du théorème 5.66 montre aussi que $COUVERTURE \leq_{\mathrm{P}} CLIQUE$. ▲

Le langage $ANTICLIQUE$

Définition 5.68. Dans un graphe G, une anticlique ou un ensemble indépendant est un sous-ensemble des sommets de G tel qu'il n'existe aucune arête joignant deux sommets de ce sous-ensemble. ▲

Définition 5.69.

$$ANTICLIQUE = \{\langle G, k\rangle \mid k \geq 1 \text{ et}$$
$$G \text{ est un graphe qui contient une anticlique de taille } k\}.$$

▲

Théorème 5.70. $ANTICLIQUE$ est NP-complet.

Aperçu de la preuve. On peut montrer que $CLIQUE \leq_{\mathrm{P}} ANTICLIQUE$, car il existe une clique de taille k dans G si, et seulement si il existe une anticlique de taille k dans le complément de G. ■

Le langage $3\text{-}COL$

Définition 5.71.

$$3\text{-}COL = \{\langle G \rangle \mid G \text{ est un graphe 3-coloriable}\}$$

▲

Théorème 5.72. $3\text{-}COL$ est NP-complet.

Preuve. Il est facile de voir que $3\text{-}COL \in \mathsf{NP}$.

Nous allons montrer que $3\text{-}SAT \leq_{\mathrm{P}} 3\text{-}COL$.

Considérons une expression booléenne ϕ en 3-FNC :

$$\phi = (t_{1,1} \vee t_{1,2} \vee t_{1,3}) \wedge \ldots \wedge (t_{k,1} \vee t_{k,2} \vee t_{k,3})$$

où $t_{i,1}$, $t_{i,2}$ et $t_{i,3}$ sont de la forme x ou $\neg x$ pour une variable booléenne $x \in \{x_1, \ldots, x_l\}$.

Nous allons d'abord décrire le graphe G tel que $f(\langle\phi\rangle) = \langle G\rangle$ où f est la fonction de réduction calculable en temps polynômial. Ensuite on montrera que ϕ est satisfaisable si, et seulement si G est 3-coloriable.

L'ensemble des $5k + 2l + 2$ sommets de G est défini par la réunion des ensembles suivants :

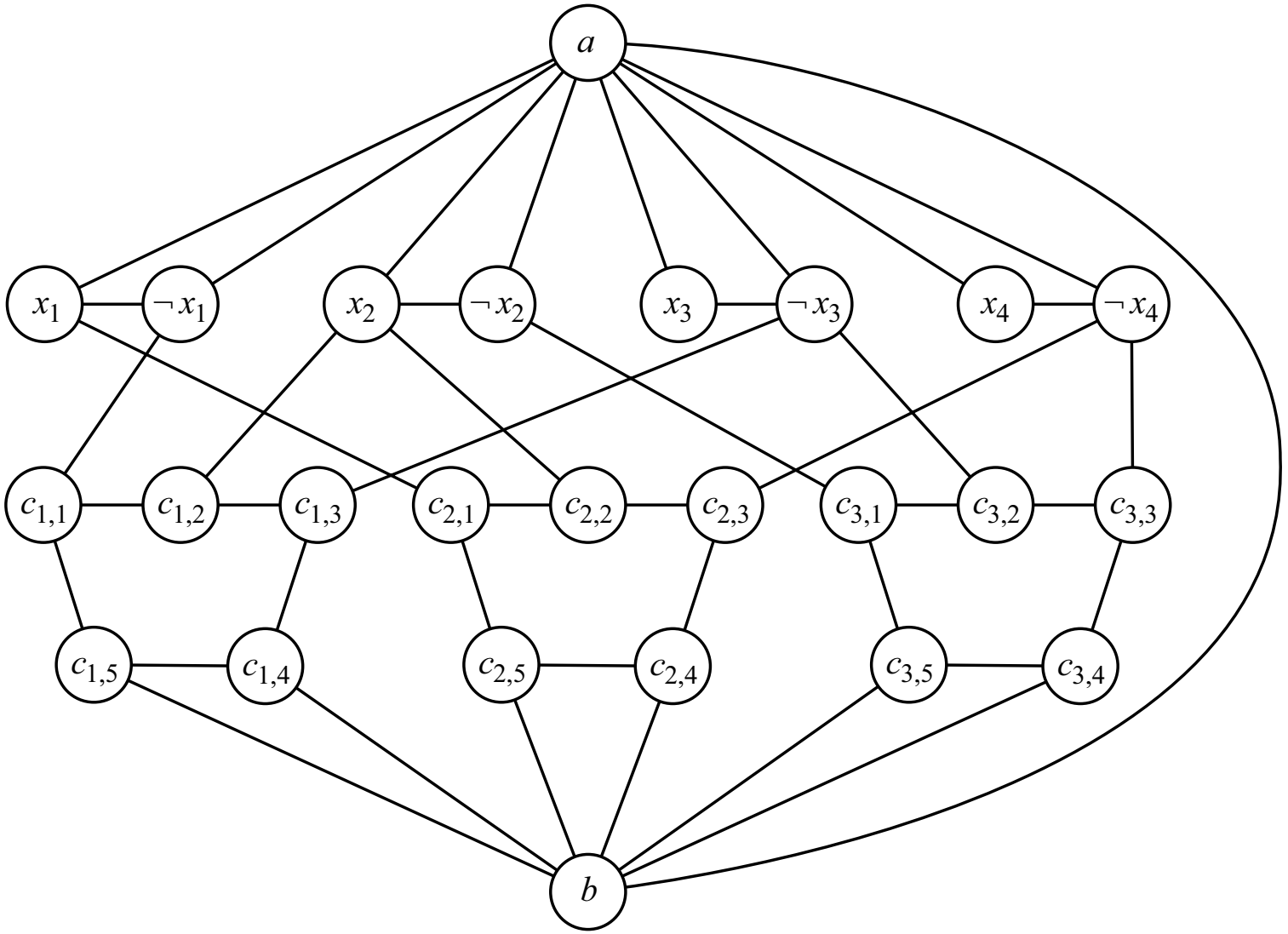
$$\begin{aligned}
\text{5 sommets par clause} \quad &: \quad \{c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}, c_{1,5}, \ldots, c_{k,1}, c_{k,2}, c_{k,3}, c_{k,4}, c_{k,5}\} \\
\text{2 sommets par variable} \quad &: \quad \{x_1, \neg x_1, \ldots, x_l, \neg x_l\}, \\
\text{et 2 sommets accessoires} \quad &: \quad \{a, b\},
\end{aligned}$$

et les $10k + 3l + 1$ arêtes de G sont les suivantes :

$$\begin{aligned}
& \bigcup_{1 \le i \le k} \{(c_{i,1}, c_{i,2}), (c_{i,2}, c_{i,3}), (c_{i,3}, c_{i,4}), (c_{i,4}, c_{i,5}), (c_{i,5}, c_{i,1})\} \\
\cup \quad & \bigcup_{1 \le i \le k} \{(c_{i,1}, t_{i,1}), (c_{i,2}, t_{i,2}), (c_{i,3}, t_{i,3}), (c_{i,4}, b), (c_{i,5}, b)\} \\
\cup \quad & \bigcup_{1 \le j \le l} \{(x_j, \neg x_j), (a, x_j), (a, \neg x_j)\} \\
\cup \quad & \{(a, b)\},
\end{aligned}$$

comme dans l'exemple illustré ci-dessous :

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



Supposons que ϕ est satisfaisable, considérons une assignation des variables booléennes $x_1, \ldots, x_l$ qui rend l'expression ϕ vraie, et montrons que G est 3-coloriable.

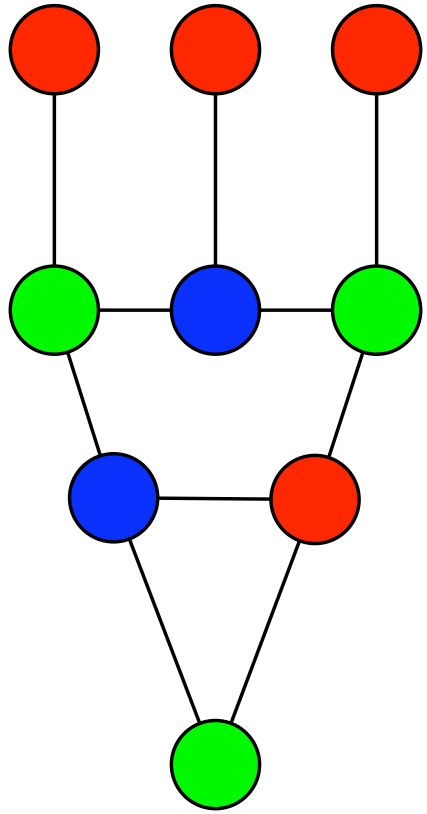
On colorie a en bleu et b en vert.

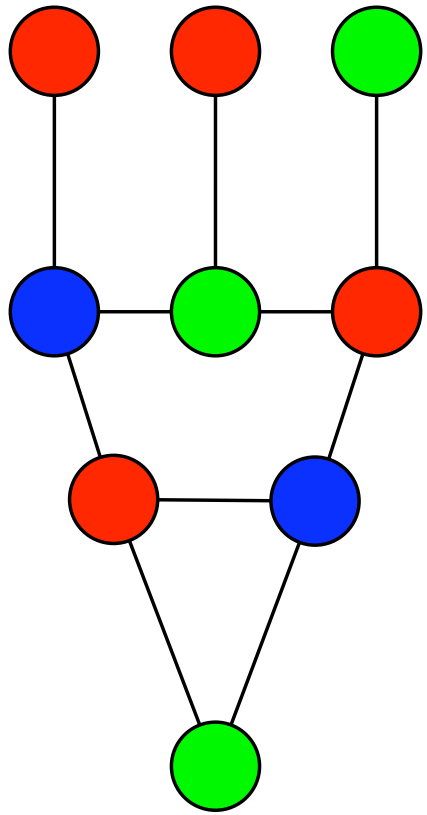
Pour $1 \leq j \leq l$, si x_j est assignée à $\langle$VRAI$\rangle$, alors on colorie x_j en rouge et $\neg x_j$ en vert. Inversement, si x_j est assignée à $\langle$FAUX$\rangle$, alors on colorie x_j en vert et $\neg x_j$ en rouge.

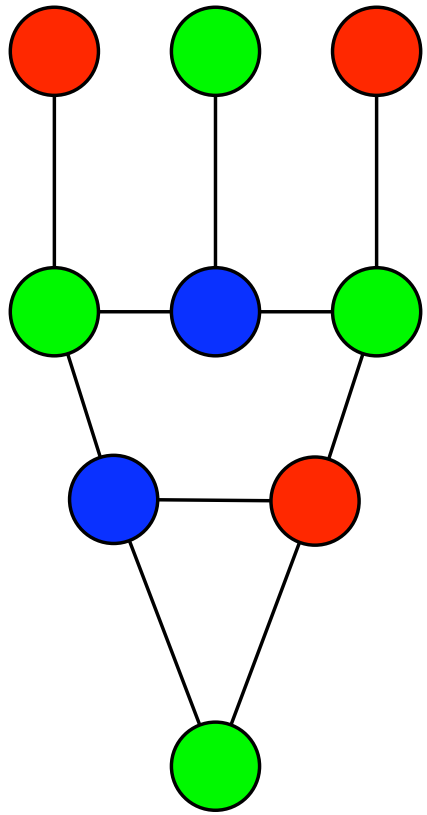
Pour $1 \leq i \leq k$, les couleurs des noeuds $c_{i,1}$, $c_{i,2}$, $c_{i,3}$, $c_{i,4}$ et $c_{i,5}$ dépendent des couleurs des noeuds b, $t_{i,1}$, $t_{i,2}$ et $t_{i,3}$ selon le tableau ci-dessous. Notons que les t_i ne peuvent être tous trois verts car dans ce cas ϕ serait fausse.

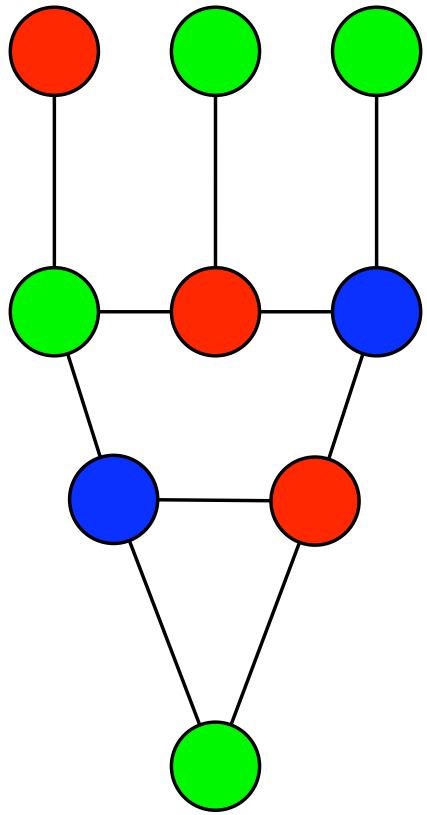
b	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$	$c_{i,1}$	$c_{i,2}$	$c_{i,3}$	$c_{i,4}$	$c_{i,5}$
vert	rouge	rouge	rouge	vert	bleu	vert	rouge	bleu
vert	rouge	rouge	vert	bleu	vert	rouge	bleu	rouge
vert	rouge	vert	rouge	vert	bleu	vert	rouge	bleu
vert	rouge	vert	vert	vert	rouge	bleu	rouge	bleu
vert	vert	rouge	rouge	rouge	bleu	vert	rouge	bleu
vert	vert	rouge	vert	rouge	vert	bleu	rouge	bleu
vert	vert	vert	rouge	rouge	bleu	vert	rouge	bleu

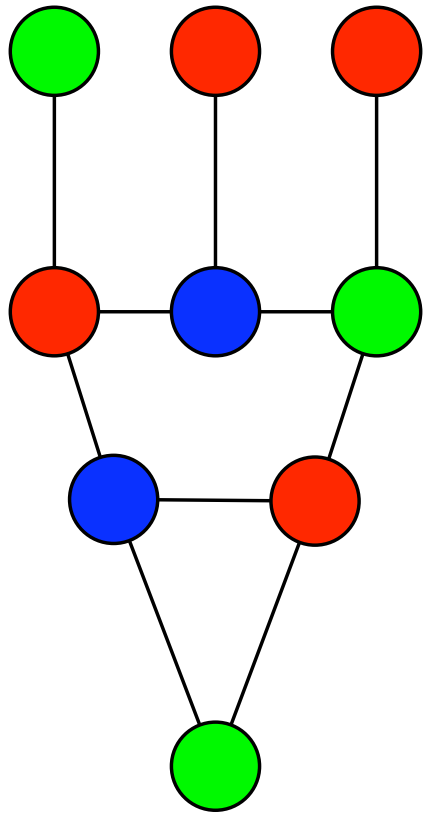
Les 7 cas de ce tableau sont illustrés dans les diagrammes suivants.

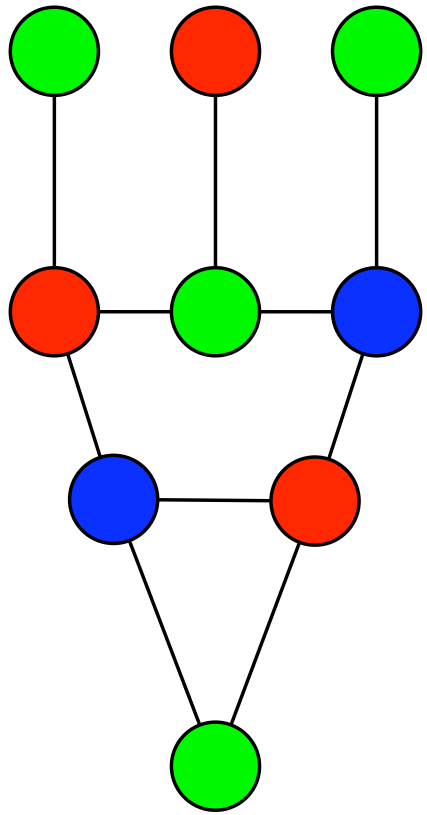


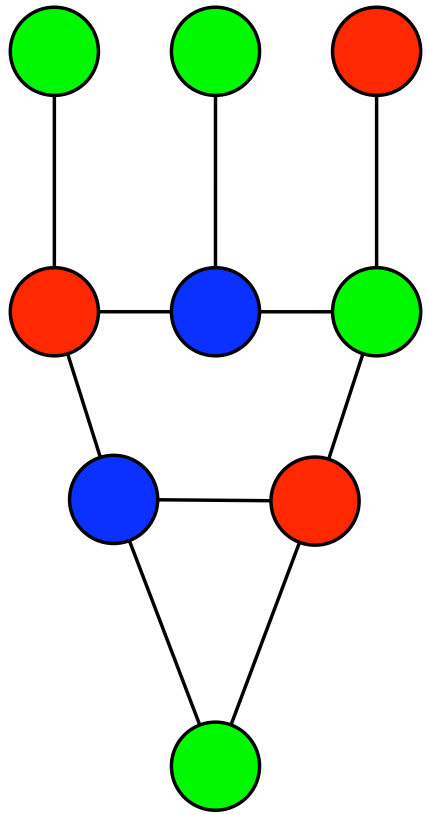






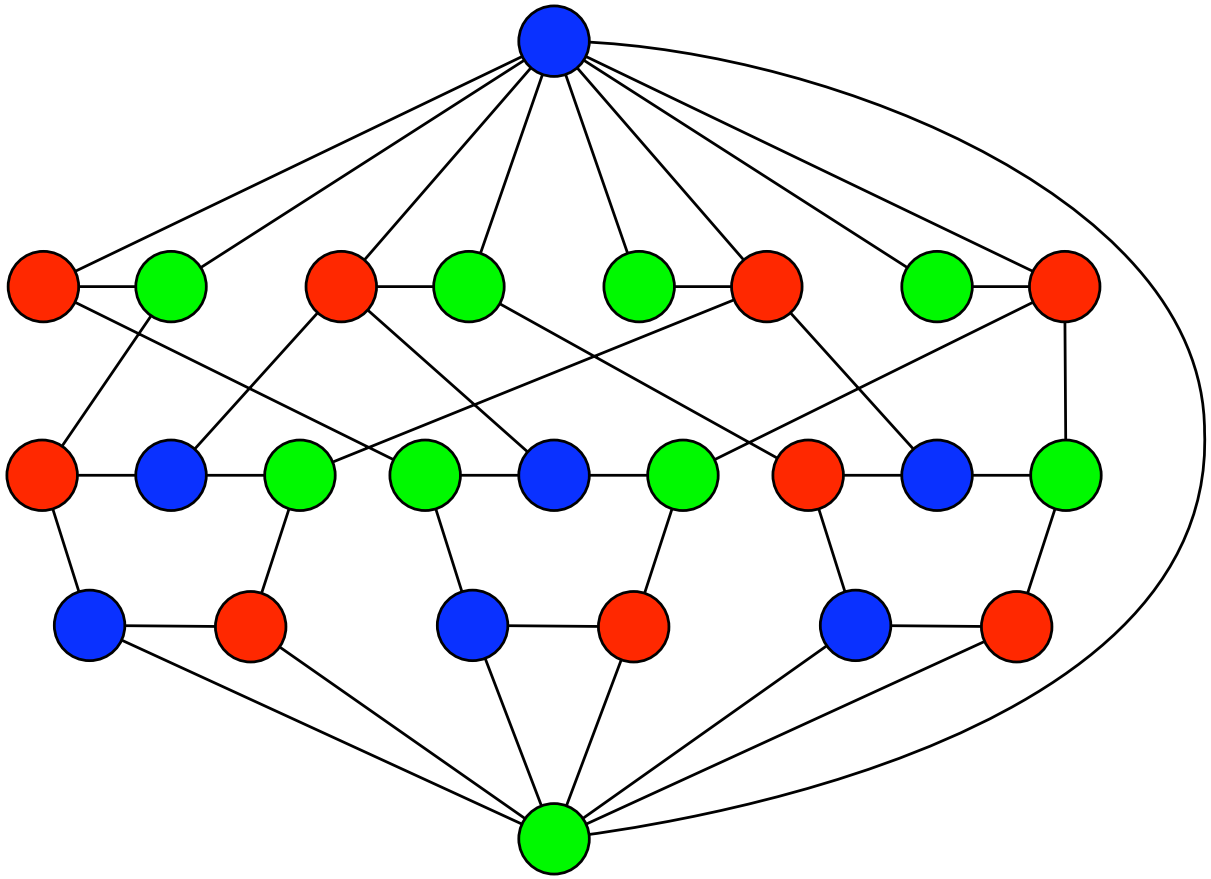






Le diagramme suivant montre le coloriage pour l'exemple introduit plus haut.

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



D'autre part, supposons que G est 3-coloriable, considérons un 3-coloriage valide, et montrons que ϕ est satisfaisable.

Sans perte de généralité, le noeud a est bleu et le noeud b est vert.

Pour $1 \leq j \leq l$, la paire de noeuds $(x_j, \neg x_j)$ est $(\text{rouge}, \text{vert})$ ou $(\text{vert}, \text{rouge})$. Choisissons l'assignation $x_j = \langle \textsc{Vrai} \rangle$ dans le premier cas, et l'assignation $x_j = \langle \textsc{Faux} \rangle$ dans le deuxième cas.

Pour $1 \leq i \leq k$, il est facile de vérifier que les noeuds $t_{i,1}$, $t_{i,2}$ et t_{i_3} ne peuvent pas être tous verts parce que dans ce cas le pentagone $(c_{i,1}, c_{i,2}, c_{i,3}, c_{i,4}, c_{i,5})$ ne pourrait pas être colorié.

Il y donc un terme vrai dans chaque clause et ϕ est vraie et satisfaisable. ■

Le langage k-COL

Définition 5.73.

$$k\text{-}COL = \{\langle G, k\rangle \mid G \text{ est } k\text{-coloriable}\}$$

▲

Théorème 5.74. k-COL est NP-complet.

Preuve. Un coloriage valide sert de témoin pour vérifier en temps polynômial qu'un graphe est k-coloriable. Donc k-$COL \in$ NP.

De plus la relation 3-$COL \leq_{\mathrm{P}} k$-COL est triviale. ■

Remarque 5.75. Le langage $k\text{-}COL$ est NP-complet, mais les langages définis semblablement $k\text{-}COUVERTURE$, $k\text{-}CLIQUE$ et $k\text{-}ANTICLIQUE$ ne le sont pas. ▲

Le langage $HORAIRE$

Définition 5.76. Un exemplaire du langage $HORAIRE$ est un encodage d'une liste d'activités, une liste d'individus, une liste de toutes les activités auxquelles participe un individu, et un entier s.

Une telle structure de données fait partie du langage si il existe un horaire de s plages pour lesquelles on peut attribuer une activité par plage sans qu'un individu soit dans deux activités différentes durant la même plage horaire. ▲

Théorème 5.77. $HORAIRE$ est NP-complet.

Aperçu de la preuve. On peut montrer que $3\text{-}COL \leq_{\mathrm{P}} HORAIRE$. ■

Le langage $PROGR\text{-}ENT$ (programmation en nombres entiers)

Définition 5.78. Les éléments du langage $PROGR\text{-}ENT$ sont de la forme $\langle A, b\rangle$ où :

- A est une matrice de nombres entiers ;
- b est un vecteur de nombre entiers ;
- le système d'inéquations suivant possède une solution en nombres entiers :

$$\begin{aligned} A_{1,1}x_1 + A_{1,2}x_2 + \ldots + A_{1,n}x_n &\leq b_1 \\ A_{2,1}x_1 + A_{2,2}x_2 + \ldots + A_{2,n}x_n &\leq b_2 \\ &\vdots \\ A_{m,1}x_1 + A_{m,2}x_2 + \ldots + A_{m,n}x_n &\leq b_m. \end{aligned}$$

▲

Théorème 5.79. $PROGR\text{-}ENT$ est NP-complet.

Aperçu de la preuve. Une solution au système sert de témoin, et la validité de cette solution peut être vérifiée en temps polynômial, donc $PROGR\text{-}ENT \in \mathsf{NP}$.

De plus, on peut montrer que $3\text{-}SAT \leq_{\mathrm{P}} PROGR\text{-}ENT$.

Voici la correspondance entre des clauses d'une expression booléenne en 3-FNC et des inéquations :

$$
\begin{aligned}
\forall x_i &: \quad 0 \leq x_i \leq 1 \\
x_{i_1} \vee x_{i_2} \vee x_{i_3} &: \quad x_{i_1} + x_{i_2} + x_{i_3} \geq 1 \\
x_{i_1} \vee x_{i_2} \vee \neg x_{i_3} &: \quad x_{i_1} + x_{i_2} + (1 - x_{i_3}) \geq 1 \\
x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3} &: \quad x_{i_1} + (1 - x_{i_2}) + (1 - x_{i_3}) \geq 1 \\
\neg x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3} &: \quad (1 - x_{i_1}) + (1 - x_{i_2}) + (1 - x_{i_3}) \geq 1
\end{aligned}
$$

■

Liste de langages NP-complets

Voir :

`http ://en.wikipedia.org/wiki/NP-complete#Example_problems`

et de nombreux autres sites webs.