ALGO1 – NP-complétude

François Schwarzentruber

1er décembre 2020

1 Problème de décision

Définition 1 (problème de décision) Un *problème de décision* est une fonction qui à une entrée associe oui ou non.

Exemple 2 **Accessibilité**
entrée : un graphe orienté, un sommet source s et un sommet destination t
sortie : oui, s'il existe un chemin de s à t dans le graphe G ; non sinon.

Exemple 3 **Arbre couvrant de poids minimum**
entrée : un graphe non orienté connexe pondéré $G = (S, A, poids)$, un seuil k
sortie : oui, s'il existe un arbre couvrant de poids $\leq k$; non sinon.

Exemple 4 **Distance d'édition**
entrée : deux mots x et y, un seuil k
sortie : oui, si la distance d'édition entre x à y est $\leq k$; non sinon.

Exemple 5 **Couplage maximum biparti**
entrée : un graphe G biparti, un seuil k
sortie : oui, s'il existe un couplage dans G de cardinal $\geq k$; non sinon.

Exemple 6 **Flot maximal**
entrée : un réseau G, un seuil k
sortie : oui, s'il existe un flot dans G de valeur $\geq k$; non sinon.

Exemple 7 **Flot maximal à valeurs entières**
entrée : un réseau G, un seuil k
sortie : oui, s'il existe un flot à valeurs entières dans G de valeur $\geq k$; non sinon.

Exemple 8 **CLIQUE**
entrée : Un graphe non orienté $G = (S, A)$, un entier k ;
sortie : oui si il y a une clique (un sous-ensemble de sommets tous reliés entre eux) de taille k dans G ; non sinon.

Exemple 9 **2-COLORATION**
entrée : Un graphe non orienté $G = (S, A)$;
sortie : oui si G est 2-coloriable, i.e. il existe une coloration $c : S \to \{1, 2\}$ avec pour tout $(s, t) \in A$, $c(s) \neq c(t)$; non sinon.

Exemple 10 **3-COLORATION**
entrée : Un graphe non orienté $G = (S, A)$;
sortie : oui si G est 3-coloriable, i.e. il existe une coloration $c : S \to \{1, 2, 3\}$ avec pour tout $(s, t) \in A$, $c(s) \neq c(t)$; non sinon.

Exemple 11 **SAT**
entrée : une formule de la logique propositionnelle φ
sortie : oui, si la formule φ est satisfiable, non sinon.

Exemple 12 **Sac à dos avec répétition (sans répétition)**
entrée : une collection d'objets $(p_i, v_i)_{i=1..n}$, un poids maximal P, un seuil k
sortie : oui, s'il existe n_i dans $\mathbb{N}$ (dans $\{0, 1\}$) avec $\sum_{i=1}^{n} n_i v_i \geq k$ avec $\sum_{i=1}^{n} n_i p_i \leq P$; non sinon.

Définition 13 (instance) Une instance d'un problème **A** est une entrée pour ce problème.

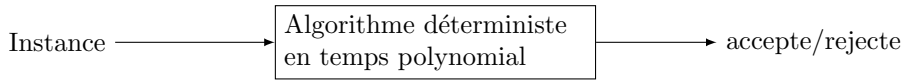
Définition 14 (taille d'une instance) La taille d'une instance/entrée est le nombre de bits qu'il faut pour la représenter.

2 Classe P

Définition 15 (notation de Landau) On note $poly(n)$ la classe des fonctions polynomiales en n.

Exemple 16 $n^3 + n \in poly(n)$. Par abus, on écrit $n^3 + n = poly(n)$.

Définition 17 (classe P) La classe P est la classe des problèmes de décision qui admettent un algorithme déterministe en temps polynomial en la taille de l'entrée.



Exemple 18 **Accessibilité** est dans P.

Exemple 19 **Arbre couvrant de poids minimum** est dans P.

Exemple 20 **Distance d'édition** est dans P.

Exemple 21 **Flot maximal** et **Flot maximal à valeurs entières** sont dans P.

Exemple 22 **Couplage maximum** est dans P.

Thèse 23 (thèse d'Edmonds-Cobham) Un problème dans P est facile.

- 👍 La plupart des algorithmes en temps polynomial sont en $O(n)$, $O(n^2)$, $O(n \log n)$, donc efficaces.
- 👍 Des fois, on utilise des algorithmes en temps exponentiels mais efficaces en pratique bien que le problème soit dans P (e.g.programmation linéaire réelle)
- 👍 Stabilité par boucle for

```
pour i := 1 à n faire
|   algoefficace(i)
```

- 👍 Stabilité par modèle de calcul (programmes Java, machines de Turing, machines RAM, etc.)
- 👎 Un algorithme qui réalise n^{100} opérations est inefficace en pratique.
- 👎 Complexité pire cas.
- 👎 Quid des aspects quantiques, probabilistes ?

3 Classe NP

NP est l'acronyme de **non-déterministe** en **temps polynomial**.

Définition 24 (problème de recherche) Un problème de décision est dit de recherche s'il peut se mettre sous la forme suivante :

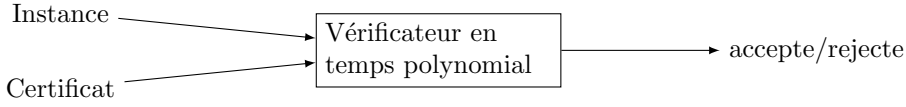
— entrée : une instance I ;
— sortie : oui s'il existe un élément S de taille $poly(|I|)$ avec $V(I, S) = 1$; non sinon.

où V est un algorithme en temps $poly(|I|)$.

Définition 25 (vérifieur ou vérificateur) L'algorithme V s'appelle un vérifieur ou un vérificateur.

Définition 26 (certificat) L'élément S s'appelle certificat.

Définition 27 (classe NP) NP est la classe des problèmes de recherche.



Exemple 28 Tous les problèmes cités plus haut sont des problèmes de recherche.

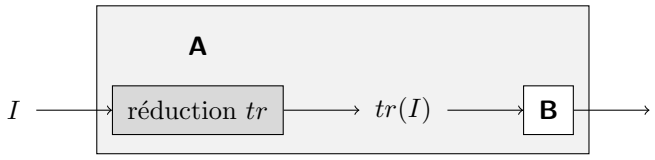
Proposition 29 P $\subseteq$ NP.

Démonstration. Juste ignorer S. ■

? Est-ce que P = NP ? Est-ce que P $\neq$ NP ?

4 Réduction polynomiale

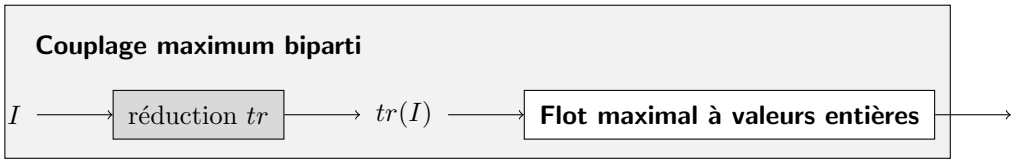
A soit plus facile que **B**.



Définition 30 (Réduction polynomiale) Une *réduction polynomiale* d'un problème **A** à un problème **B** est une fonction tr, qui, à toute instance I de **A**, associe une instance $tr(I)$ de **B**, telle que

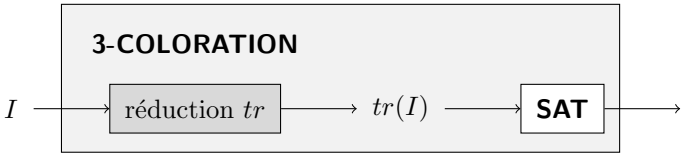
1. pour tout I de **A**, I est une instance positive de **A** ssi $tr(I)$ est une instance positive de **B** ;
2. $tr(I)$ calculable en temps $poly(|I|)$.

Exemple 31



Proposition 32 Si **A** se réduit à **B** et **B** dans P, alors **A** est dans P.

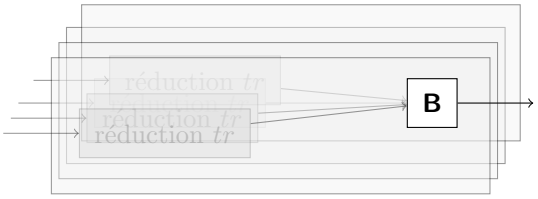
Exemple 33



Proposition 34 Si **A** se réduit à **B** et **B** dans NP, alors **A** est dans NP.

5 NP-dureté

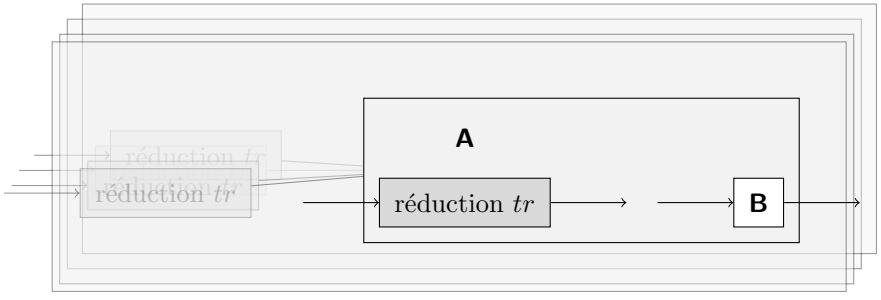
Définition 35 (NP-dur) Un problème est NP-*dur* si tout problème de NP s'y réduit en temps polynomial.



Proposition 36 Si un problème NP-dur est dans P alors P = NP.

Proposition 37 Si **A** se réduit polynomialement à **B** et que **A** est NP-dur alors **B** est NP-dur.

Démonstration. Soit un problème **C** dans NP. Il existe une réduction polynomiale tr_1 de **C** dans **A**. Par ailleurs, il existe une réduction polynomiale tr_2 de **A** dans **B**. Ainsi, $tr_2 \circ tr_1$ est une réduction polynomiale de **C** dans **B**.



■

Définition 38 (NP-complet) Un problème est NP-*complet* s'il est dans NP et NP-dur.

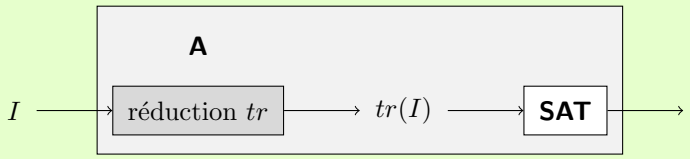
6 Théorème de Cook (piste verte)

Cette section est une version édulcorée de l'idée de la démonstration du théorème de Cook, sans avoir besoin de comprendre les algorithmes non-déterministes et les machines de Turing.

Théorème 39 **SAT** est NP-complet.

IDÉE DE LA DÉMONSTRATION. D'abord **SAT** est dans NP : étant donné une formule φ et une valuation ν, on peut vérifier en temps polynomial en $|\varphi|$ si la formule φ est rendue vraie par ν.

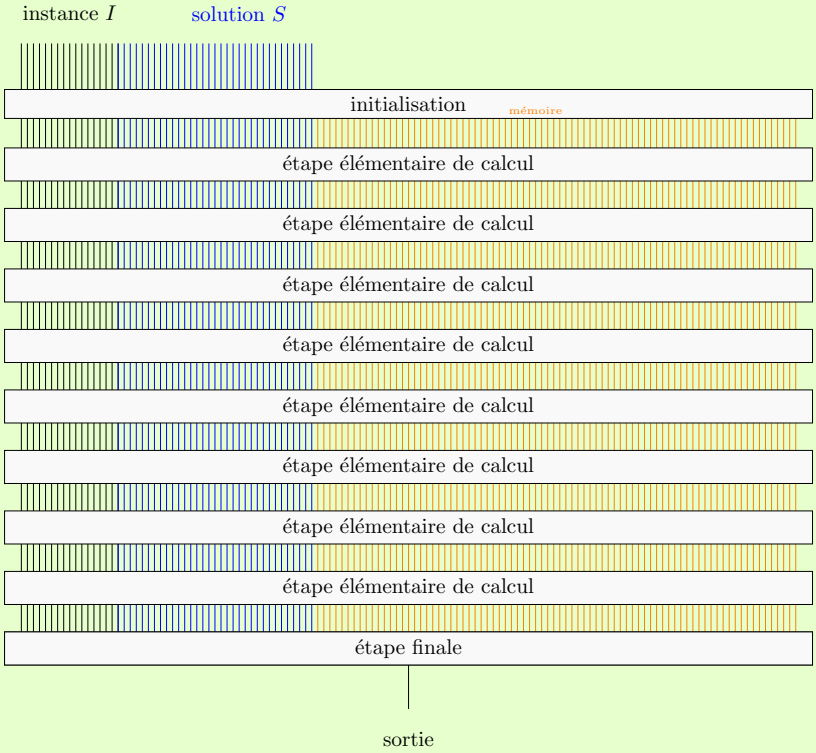
Montrons que **SAT** soit NP-dur. Soit **A** un problème dans NP et donnons une réduction de **A** dans **SAT**.



Comme **A** est dans NP, il existe un vérificateur V qui s'exécute en temps $poly(|I|)$ tel que

$$I \text{ est une instance positive de } \textbf{A} \text{ ssi il existe } S \text{ de taille } poly(|I|) \text{ tel que } V(I,S) = 1.$$

Circuit pour le vérifieur. Voici un schéma d'un circuit logique C_n pour l'algorithme V sur des instances de taille n. Le circuit prend en entrée le codage binaire d'une instance I de taille n, et le codage binaire d'une solution S de taille $poly(n)$. En sortie, il renvoie $V(I,S)$. Il existe un algorithme qui, étant donné une taille n, génère un circuit logique pour V qui traitent les instances de taille n :



La réduction est donc :

fonction $tr(I)$

1. calculer le circuit C_n qui traite les instances de taille n, pour $n = |I|$
2. calculer le circuit C'_n qui est le circuit C_n où les entrées de l'instance sont fixées pour qu'elles codent I

retourner une formule équisatisfiable au circuit C'_n

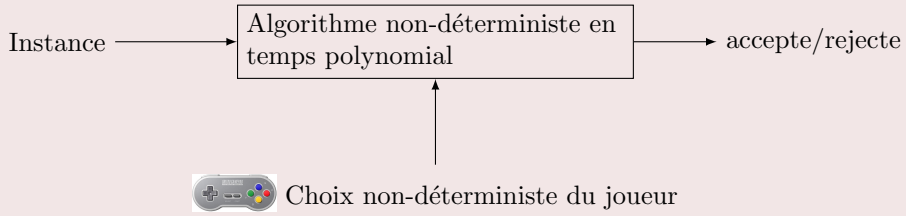
7 Algorithmes non-déterministes (piste noire)

Algorithme **non-déterministe** $\sim$ jeu à un joueur

On considère des algorithmes qui terminent sur **accept** ou sur **reject** .

Définition 40 (algorithme non-déterministe) Un algorithme de non-déterministe estsimilaire à un algorithme déterministe sauf qu'il peut contenir des instructions de choix par un joueur, typiquement le choix d'un bit :

choose $i \in \{0, 1\}$.



Exemple 41

procédure 3-coloration(G)
  pour $s \in S$ **faire**
    choose $c[s]$ dans $\{0, 1, 2\}$;
  si c est une 3-coloriation **alors**
    accept (gagné)
  sinon
    reject (perdu)

Exemple 42

procédure sat(φ)
  pour toute proposition atomique p dans φ **faire**
    choose $\nu[p]$ dans $\{faux, vrai\}$;
  si $\nu \models \varphi$ **alors**
    accept (gagné)
  sinon
    reject (perdu)

Remarque 43 Un algorithme déterministe est un algorithme non-déterministe qui ne contient pas de choix pour le joueur.

Définition 44 (acceptation d'une instance) Un algorithme non-déterministe accepte une instance ω s'il existe une exécution (i.e. une suite de choix) de l'algorithme sur ω qui mène à **accept** .

Définition 45 (décision d'un problème de décision) Un algorithme non-déterministe décide un problème de décision si :

— toute exécution depuis toute configuration initiale est fini ;
— une instance est positive ssi l'algorithme l'accepte.

Définition 46 (alternative pour la classe NP**)** La classe NP est la classe des problèmes de décision qui admettent un algorithme non-déterministe en temps polynomial en la taille de l'entrée.

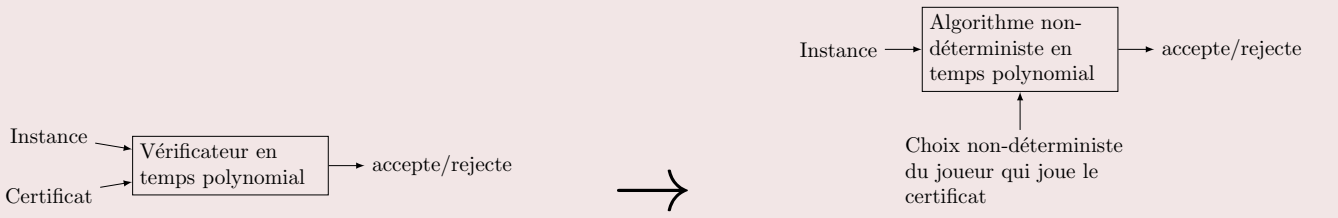
Théorème 47 **SAT** est dans NP.

Démonstration. L'algorithme donné plus haut est non-déterministe et décide **SAT** en temps polynomial. ■

Proposition 48 Les définitions 27 et 46 sont équivalentes.

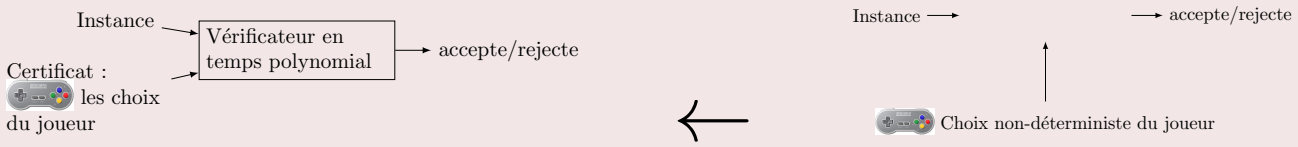
Démonstration.

⇐ Supposons qu'un problème **A** admette un vérificateur V en temps polynomial. On construit l'algorithme non déterministe qui décide **A** en temps polynomial.



procédure $M(w)$
  choose c de longueur $f(|w|)$
  si $V(w, c)$ accepte **alors**
    accept
  sinon
    reject

⇒ Supposons que **A** admette un algorithme non-déterministe qui le décide en temps polynomial. On construit le vérificateur suivant.



procédure $V(w, c)$
  simuler l'exécution $M(w)$ en prenant les choix non-déterministes conseillés par c
  si l'exécute accepte **alors**
    accept
  sinon
    reject

■

8 Machines de Turing (piste noire)

Définition 49 Un problème de décision est un langage.

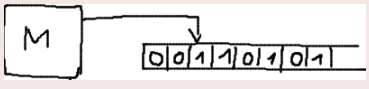
Exemple 50 **CLIQUE** $= \{\langle G, k\rangle \in \{0,1\}^* \mid G$ est un graphe non orienté, k entier tels que G admet une clique de taille $k\}$

`http://people.irisa.fr/Francois.Schwarzentruber/turing_machine_simulator/`

Définition 51 (machine de Turing déterministe) Une *machine de Turing déterministe* est un tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

où :
- Q est un ensemble fini non vide d'états ;
- Σ est l'alphabet fini des mots d'entrée tel que $\sqcup \notin \Sigma$;
- Γ est l'alphabet fini du ruban avec $\Sigma \subseteq \Gamma$ et $\sqcup \in \Gamma$;
- δ est une fonction partielle $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow, \bullet\}$;
- $q_0 \in Q$ est l'*état initial* ;
- $q_{acc} \in Q$ est l'*état d'acceptation* ;
- $q_{rej} \in Q$ est l'*état de rejet* avec $q_{acc} \neq q_{rej}$.



La fonction de transition δ se lit comme suit :

δ(état courant, lettre lue sous la tête) = (nouvel état, lettre écrite, déplacement de la tête).

Pour inclure les choix du joueur dans un algorithme non-déterministe, on se propose des machines de Turing non-déterministe où la fonction de transition est remplacée par une relation de transition.

Définition 52 (machine de Turing non-déterministe) Une machine de Turing *non-déterministe* est un tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc})$ comme précédemment sauf que δ est maintenant une relation de transition

$$\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \bullet, \rightarrow\})$$

Définition 53 (configuration) Une *configuration* est un mot uqv où $u \in \Gamma^*$, $v \in \Gamma^\omega$ et $q \in Q$.

Définition 54 (configuration initiale) Une *configuration initiale* depuis l'instance $\omega \in \Sigma^*$ est le mot $q_0 w \sqcup\sqcup \ldots$ où q_0 est l'état initial.

Définition 55 (configuration acceptante) Une configuration est *acceptante* si l'état de la configuration est q_{acc}.

Définition 56 (un pas de calcul) Pour tout $a, b, c \in \Gamma$, $u \in \Gamma^*, v \in \Gamma^\omega$,
- Si $(q, a, q', b, \leftarrow) \in \delta$ alors $ucqav$ devient $uq'cbv$;
- Si $(q, a, q', b, \bullet) \in \delta$, alors $uqav$ devient $uq'bv$;
- Si $(q, a, q', b, \rightarrow) \in \delta$, alors $uqav$ devient $ubq'v$.

Définition 57 (exécution) Une *exécution* est une suite maximale de configurations $C_1, \ldots, C_k$ telle que C_i devient C_{i+1} pour tout $i \in \{1, \ldots, k-1\}$.

Définition 58 (exécution acceptante) Une exécution est $C_1, \ldots, C_k$ *acceptante* si C_k est une configuration acceptante.

Définition 59 (acceptation d'un mot) Une machine M *accepte* un mot w s'il existe une exécution acceptante $C_1, \ldots, C_k$ où C_1 est la configuration initiale $q_0 w \sqcup\sqcup \ldots$.

Définition 60 (arbre de calcul) L'*arbre de calcul* depuis $q_0 w \sqcup \ldots$ est l'arbre de racine $q_0 w \sqcup \ldots$ et t.q. les fils de tout nœud C sont les configurations C' où C devient C'.

Définition 61 (langage accepté) Le *langage accepté* par M, noté $L(M)$, est l'ensemble des mots w acceptés par M.

Définition 62 (décideur, langage décidé) Une machine de Turing M qui s'arrête depuis $q_0 w$ pour tout mot w est appelé *décideur*. On dit que le langage accepté par M est *décidé* par M.

Définition 63 (M décide L en temps f) Soit $f : \mathbb{N} \to \mathbb{N}$. M décide L en temps f si
- M décide L ;
- pour tout w, la hauteur de l'arbre de calcul depuis $q_0 w$ est $\leq f(|w|)$.

Définition 64 (classe P) P est la classe des langages décidés par une machine de Turing déterministe en temps polynomial.

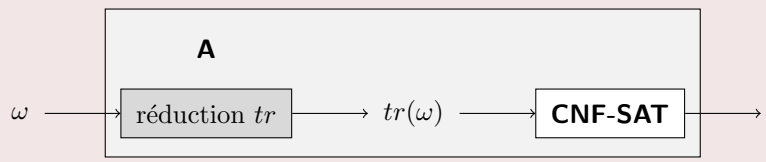
Définition 65 (classe NP) NP est la classe des langages décidés par une machine de Turing non-déterministe en temps polynomial.

9 Théorème de Cook (piste noire)

Théorème 66 (de Cook) **SAT** est NP-complet.

Théorème 67

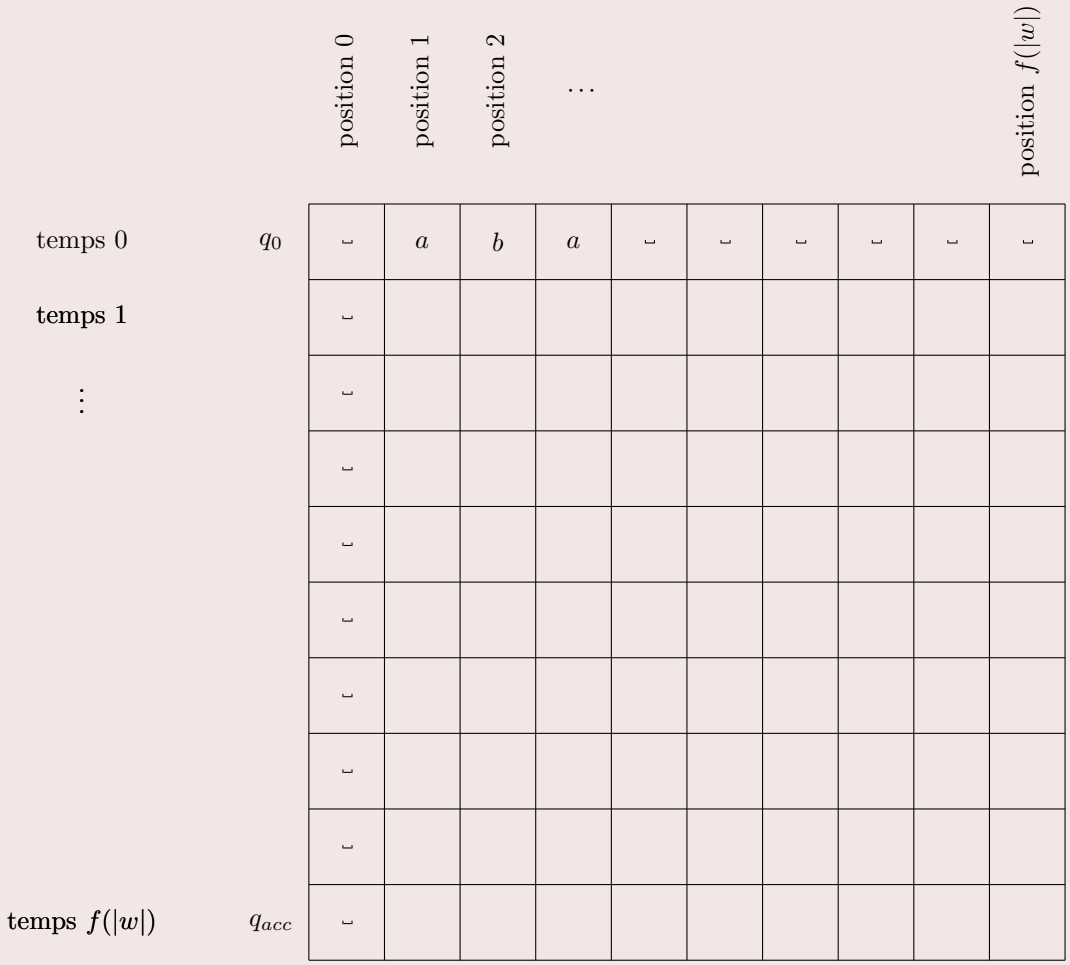
Démonstration. Nous allons démontrer que **SAT** est NP-dur. Définissons une réduction polynomiale tr tel que $\omega \in \mathbf{A}$ ssi $tr(\omega)$ est une formule satisfiable.



Soit $M = (\Sigma, \Gamma, Q, \delta, q_0, q_{acc})$ une machine de Turing non-déterministe qui décide **A** en temps polynomial. Il existe un polynôme f tel que, pour toute entrée ω, toutes les exécutions de M soient de longueur au plus $f(|\omega|)$. On modifie M pour que l'état final acceptant q_{acc} est un état sur lequel on boucle.

La formule $tr(\omega)$ exprime

"il existe une exécution acceptante de M sur le mot ω en temps $f(|\omega|)$".



Définition de $tr(\omega)$. Soit $\mathcal{C} = \{0, \ldots, f(|\omega|)\}$. On introduit les propositions atomiques :

au temps t, l'état est q — au temps t, la position du curseur est i — au temps t, la case n° i contient a — au temps t, on tire la transition τ

où $t, i \in \mathcal{C}$, $q \in Q$, $a \in \Gamma$ et $\tau \in \delta$.

Seules les $f(|\omega|)$ cases du ruban sont pertinentes car la tête de lecture ne va jamais au-delà.

Définissons $tr(\omega)$ comme la conjonction les formules 1-8 suivantes.

Configuration initiale

1. $\boxed{\text{au temps } 0, \text{ la case n° } 0 \text{ contient } \sqcup} \wedge \boxed{\text{au temps } 0, \text{ la case n° } 1 \text{ contient } x_1} \wedge \ldots \boxed{\text{au temps } 0, \text{ la case n° } n \text{ contient } x_n} \wedge \boxed{\text{au temps } 0, \text{ la case n° } n+1 \text{ contient } \sqcup} \wedge \ldots \boxed{\text{au temps } 0, \text{ la case n° } f(|\omega|) \text{ contient } \sqcup}$

   Au début, le ruban contient $\sqcup x_1 \ldots x_n \sqcup \ldots \sqcup$

2. $\boxed{\text{au temps } 0, \text{ l'état est } q_0} \wedge \boxed{\text{au temps } 0, \text{ la position du curseur est } 1}$

   Au début, la machine est dans l'état initial q_0 à l'instant 0 et le curseur est à la position 1.

Exécution acceptante

3. $\boxed{\text{au temps } f(|w|), \text{ l'état est } q_{acc}}$

   La machine atteint l'état d'acceptation q_{acc}

Unicité et existence des valeurs.

4. $\bigwedge_{t\in\mathcal{C}} \bigvee_{q\in Q} \boxed{\text{au temps } t, \text{ l'état est } q}$

   La machine est dans un état à tout instant t

5. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{q,q'\in Q|q\neq q'} \left(\neg \boxed{\text{au temps } t, \text{ l'état est } q} \vee \neg \boxed{\text{au temps } t, \text{ l'état est } q'} \right)$

   La machine n'est jamais dans deux états à la fois

6. $\bigwedge_{t\in\mathcal{C}} \bigvee_{i\in\mathcal{C}} \boxed{\text{au temps } t, \text{ la position du curseur est } i}$

   Le curseur est positionné quelque part à tout instant t

7. $\bigwedge_{t\in\mathcal{C}} \neg \boxed{\text{au temps } t, \text{ la position du curseur est } 0}$

   Le curseur n'est jamais en dehors sur la gauche

8. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{i,i'\in\mathcal{C}|i\neq i'} \left(\neg \boxed{\text{au temps } t, \text{ la position du curseur est } i} \vee \neg \boxed{\text{au temps } t, \text{ la position du curseur est } i'} \right)$

   Le curseur n'a jamais deux positions différentes

9. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{i\in\mathcal{C}} \bigvee_{a\in\Sigma} \boxed{\text{au temps } t, \text{ la case n° } i \text{ contient } a}$

   À tout instant, toute case du ruban contient une lettre

10. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{a,b\in\Sigma|a\neq b} \left(\neg \boxed{\text{au temps } t, \text{ la case n° } i \text{ contient } a} \vee \neg \boxed{\text{au temps } t, \text{ la case n° } i \text{ contient } b} \right)$

    Une case ne contient au plus qu'une lettre

11. $\bigwedge_{t\in\mathcal{C}} \bigvee_{\tau\in\delta} \boxed{\text{au temps } t, \text{ on tire la transition } \tau}$

    À tout instant t, on tire une transition pour aller vers l'instant $t+1$

12. $\bigwedge_{\tau,\tau'\in\delta|\tau\neq\tau'} \left(\neg \boxed{\text{au temps } t, \text{ on tire la transition } \tau} \vee \neg \boxed{\text{au temps } t, \text{ on tire la transition } \tau'} \right)$

    On ne tire jamais plus d'une transition

Exécution des transitions

13. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{i\in\mathcal{C}} \bigwedge_{a\in\Sigma} \left(\left(\neg \boxed{\text{au temps } t, \text{ la position du curseur est } i} \wedge \boxed{\text{au temps } t, \text{ la case n° } i \text{ contient } a} \right) \rightarrow \boxed{\text{au temps } t+1, \text{ la case n° } i \text{ contient } a} \right)$

    on ne change pas le contenu du ruban si le curseur n'y est pas

14. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|\omega|)\}} \bigwedge_{(q,a,q',b,d)\in\delta} \left(\boxed{\text{au temps } t, \text{ on tire la transition } (q,a,q',b,d)} \rightarrow \boxed{\text{au temps } t, \text{ l'état est } q} \right)$

15. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|\omega|)\}} \bigwedge_{(q,a,q',b,d)\in\delta} \bigwedge_{i\in\mathcal{C}} \left[\left(\boxed{\text{au temps } t, \text{ on tire la transition } (q,a,q',b,d)} \wedge \boxed{\text{au temps } t, \text{ la position du curseur est } i} \right) \rightarrow \boxed{\text{au temps } t, \text{ la case n° } i \text{ contient } a} \right]$

16. $$\bigwedge_{t\in\mathcal{C}\setminus\{f(|\omega|)\}} \bigwedge_{(q,a,q',b,d)\in\delta} \left(\text{au temps } t\text{, on tire la transition } (q,a,q',b,d) \rightarrow \text{au temps } t+1\text{, l'état est } q' \right)$$

17. $$\bigwedge_{t\in\mathcal{C}\setminus\{f(|\omega|)\}} \bigwedge_{(q,a,q',b,d)\in\delta} \bigwedge_{i\in\mathcal{C}} \left[\left(\text{au temps } t\text{, on tire la transition } (q,a,q',b,d) \wedge \text{au temps } t\text{, la position du curseur est } i \right) \rightarrow \text{au temps } t+1\text{, la case n}^\circ\ i \text{ contient } b \right]$$

18. $$\bigwedge_{t\in\mathcal{C}\setminus\{f(|\omega|)\}} \bigwedge_{(q,a,q',b,d)\in\delta} \bigwedge_{i\in\mathcal{C}|i+d\in\mathcal{C}} \left(\text{au temps } t\text{, on tire la transition } (q,a,q',b,d) \wedge \text{au temps } t\text{, la position du curseur est } i \right) \rightarrow \text{au temps } t+1\text{, la position du curseur est } i+d$$

La formule $tr(\omega)$ est bien en forme normale conjonctive.

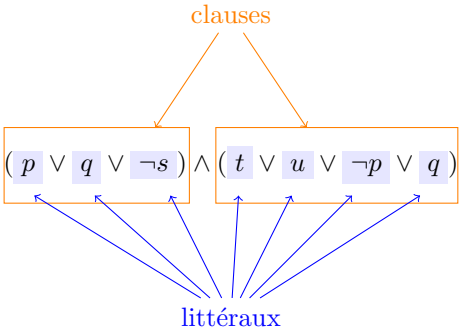
1. Le mot ω est une instance positive de **A** ssi $tr(\omega)$ est une formule satisfiable.
   $\boxed{\Rightarrow}$ Soit ω est une instance positive de **A**. Alors il existe une exécution de M acceptante sur le mot ω en temps $f(|\omega|)$ et sur une longueur de ruban de $f(|\omega|)$. On construit une valuation qui satisfait $tr(\omega)$.
   $\boxed{\Leftarrow}$ Si $tr(\omega)$ est satisfiable. Soit ν une valuation qui satisfait $tr(\omega)$. On construit à partir de ν une exécution de M acceptante sur le mot ω. Donc le mot ω est une instance positive de **A**.
2. $tr(\omega)$ est calculable en temps polynomial en $|\omega|$ à partir de ω.

■

10 Quelques problèmes NP-complets

10.1 CNF-SAT

Définition 68 (littéral) Un *littéral* est une proposition atomique ou la négation d'une proposition atomique.



Exemple 69 p $\neg p$ q $\neg q$

Définition 70 (clause) Une *clause* est une disjonction de littéraux.

Exemple 71 $(\neg p \vee q \vee s \vee \neg t)$.

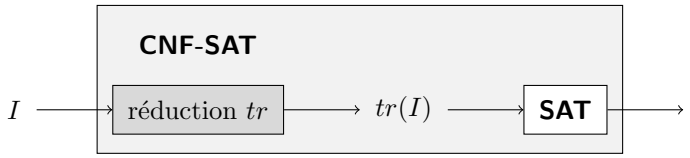
Définition 72 (forme normale conjonctive) Une *forme normale conjonctive* (FNC) est une formule de la forme $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \ell_{i,j}$ où les $\ell_{i,j}$ sont des littéraux.

Exemple 73 $(q \vee \neg s \vee p) \wedge (\neg p \vee q \vee s \vee \neg t)$.

Définition 74 **CNF-SAT**

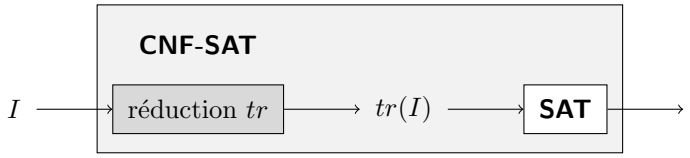
entrée : Une formule φ de la logique propositionnelle en forme normale conjonctive ;
sortie : oui si φ est satisfiable ; non sinon.

Exemple 75



Théorème 76 **CNF-SAT** est dans NP.

Démonstration.



■

Théorème 77 **CNF-SAT** est NP-dur.

Démonstration. En fait, la démonstration de Cook construit une forme normale conjonctive ! ■

10.2 3-SAT

Définition 78 (3-forme normale conjonctive) Une 3-forme normale conjonctive est une formule normale conjonctive dans laquelle il y a au plus 3 littéraux dans une clause.

Exemple 79 $(p \vee q) \wedge (r \vee \neg p \vee q) \wedge (\neg r \vee s \vee \neg p)$.

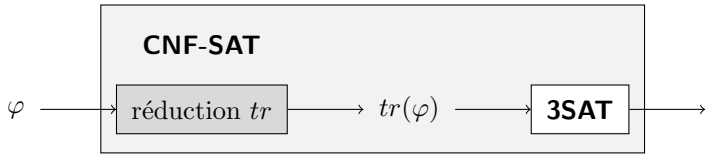
Définition 80 **3-SAT**

entrée : Une formule φ de la logique propositionnelle en 3-forme normale conjonctive ;
sortie : oui si φ est satisfiable ; non sinon.

Remarque 81 On peut toujours supposer qu'il y a exactement 3 littéraux par clause en dupliquant des littéraux.

Proposition 82 **3-SAT** est NP-dur.

Démonstration.
Nous allons réduire polynomialement **CNF-SAT** à **3SAT**.



Si φ est une forme normale conjonctive, $tr(\varphi)$ est obtenue à partir de φ en remplaçant chaque clause par un ensemble de 3-clauses en introduisant des variables supplémentaires.

Exemple 83 $(a \vee b \vee c \vee d \vee e) \quad \rightsquigarrow \quad (a \vee b \vee \alpha) \wedge (\neg\alpha \vee c \vee \beta) \wedge (\neg\beta \vee d \vee e).$

Voici l'algorithme pour tr :

```
entrée : une forme normale conjonctive φ
sortie : une 3-forme normale conjonctive φ' telle que φ satisfiable ssi φ' satisfiable
fonction tr(φ)
    φ' := ⊤
    φ s'écrit sous la forme ⋀_{i=1..n} ⋁_{j=1..n_i} ℓ_ij où ℓ_ij sont des littéraux
    pour i := 1..n faire
        si n_i = 1 alors
            ajouter la clause ℓ_i1 à φ'
        sinon
            ajouter la clause ℓ_i1 ∨ α_i2 à φ'
            pour j := 2..n_i − 1 faire
                ajouter la clause ¬α_{i,j} ∨ ℓ_ij ∨ α_{i,j+1} à φ'
            ajouter la clause ¬α_{i,n_i} ∨ ℓ_{i n_i} à φ'
    retourner φ'
```

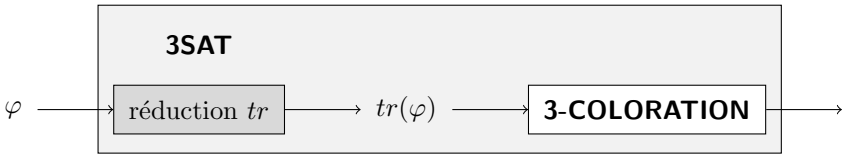
1. φ est une instance positive de **SAT** ssi $tr(\varphi)$ est une instance positive de **3SAT**.
   $\boxed{\Leftarrow}$ Supposons qu'il existe une valuation ν qui rende φ vraie (on écrit $\nu \models \varphi$ pour dire que cela). En particulier, chaque clause $\bigvee_{j=1..n_i} \ell_{ij}$ est vraie. Il existe donc $j \in \{1, \ldots, n_i\}$ tel que $\nu \models \ell_{ij}$. On modifie ν en disant que $\alpha_{ij'}$ pour $j' < j$ est vraie, puis que $\alpha_{ij'}$ pour $j' \geq j$. Ainsi, toutes les clauses de φ' sont rendues vraies par ν.
   $\boxed{\Rightarrow}$ Réciproquement, supposons que $tr(\varphi)$ soit vraie pour une certaine valuation ν. Montrons $\nu \models \varphi$. Par l'absurde, supposons qu'une clause $\bigvee_{j=1..n_i} \ell_{ij}$ soit fausse pour ν. On aurait alors ν qui rend vraie $\alpha_{i2}, \ldots, \alpha_{in_i}$ mais aussi qui rend faut α_{in_i}. Contraduction. On a bien $\nu \models \varphi$.
2. $tr(\varphi)$ est calculable en temps polynomial en la taille de φ.

■

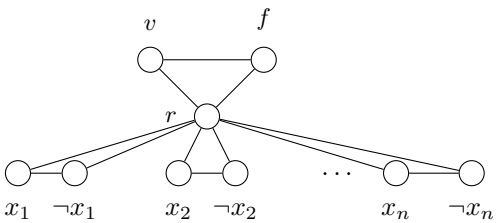
10.3 3-coloration

Théorème 84 **3-COLORATION** est NP-dur.

Démonstration.



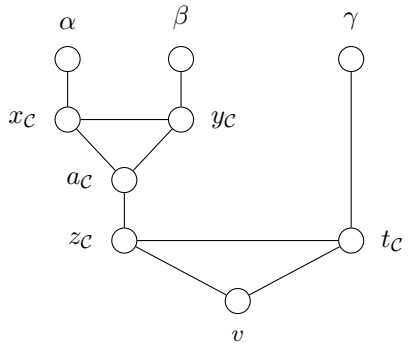
On définit une réduction tr en temps polynomial qui à toute **3SAT**-instance φ associe une **3-COLORATION**-instance $tr(\varphi)$. $tr(\varphi)$ est un graphe basé sur le *gadget* suivant :



où $x_1, x_2, \ldots, x_n$ sont les propositions atomiques qui apparaissent dans φ. On code le fait qu'un littéral est vrai par le fait qu'il a la couleur du sommet v et faux s'il a la couleur du sommet f. La couleur de $\neg x$ est toujours différente de celle de x.

Ensuite, pour chaque 3-clause $\mathcal{C}$ de la forme $(\alpha \vee \beta \vee \gamma)$ de φ, on ajoute le gadget ci-dessous. Pour coder une 2-clause $\mathcal{C}$ de la forme $(\alpha \vee \beta)$, on prend le même gadget mais avec f à la place de γ. La fonction tr est calculable en temps polynomial.

Lemme 85 Dans toute coloration, l'un des sommets α, β, γ a la couleur de v.



Démonstration. Par l'absurde, supposons qu'ils ont tous la couleur de f (on rappelle qu'ils ne peuvent pas avoir la couleur de r car la structure initiale ne le permet pas). Dans ce cas, comme γ est de la couleur de f, pour sûr $t_\mathcal{C}$ est de la couleur de r donc $z_\mathcal{C}$ est de la couleur de f. D'autre part, comme α et β sont de la couleur de f, pour sûr, $x_\mathcal{C}$ et $y_\mathcal{C}$ ne sont pas de couleur de f donc $a_\mathcal{C}$ est de couleur de f. Contradiction. ■

Lemme 86 On peut compléter toute coloriation où les sommets α, β, γ sont de la couleur de f ou v et l'un d'eux est de la couleur de v.

Démonstration. Faire tous les cas. ■

Lemme 87 φ satisfiable ssi $tr(\varphi)$ est 3-coloriable.

Démonstration. $\boxed{\Rightarrow}$ Supposons que φ est satisfiable et soit ν une valuation telle que $\nu \models \varphi$. On colorie les noeuds de la façon suivante :

— $c[r] = 2$; $c[v\ ] = 1$; $c[f\ ] = 0$;
— $c[p] = \nu[p]$;
— $c[\neg p] = 1 - \nu[p]$.

Par le lemme 86, on complète la coloriation pour tous les gadgets. Donc $tr(\varphi)$ est 3-coloriable.

$\boxed{\Leftarrow}$ Supposons que $tr(\varphi)$ est 3-coloriable. On construit la valuation ν :

— $\nu[p_i] = 1$ si p_i est colorié avec la couleur de v : 0 sinon.

Par le lemme 85, $\nu \models \varphi$. ■ ■

11 Notes bibliographiques

La classe NP a été définie initialement par des machines de Turing par Cook en 1971 [Coo71]. Le non-déterminisme est aussi présent dans d'autres modèles de calcul : automates, automates à pile. Karp a donné et montré que plusieurs problèmes sont NP-complets [Kar72]. On trouve une grande liste de problèmes NP-complets dans [GJ79].

Références

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.