Théorème de Cook

Leçons : 913 ; 915 ; 916 ; 928

Références : CARTON, *Langages formels, calculabilité et complexité* (p.203) [?] et WOLPER, *Introduction à la calculabilité* (p.185) [?] et LESESVRE–MONTAGNON–LE BARBENCHON–PIERRON, *131 développements pour l'oral* (p. 791) [?]

Le théorème de Cook est un résultat fondamental en théorie de la complexité car il montre la NP-complétude du problème Sat *en logique propositionnelle. Il est d'autant plus important que c'est la brique élémentaire qui sert à montrer la NP-difficulté de nombreux problèmes, comme celui de la 3-coloration d'un graphe, le problème du voyageur de commerce ou le problème du sac à dos. On effectue une réduction en simulant une machine de Turing, ce qui justifie le placement de ce développement dans la leçon 913. Le résultat en lui-même est un pilier de la classe de complexité des problèmes NP-complets, ce qui en fait une bonne illustration des leçons 915 et 928. De plus, le problème de satisfiabilité d'une formule en calcul propositionnel est une question récurrente ; le fait de savoir qu'il s'agit d'un problème NP-complet permet de comprendre sa difficulté, ce qui justifie son placement dans la leçon 916.*

Introduction :

On rappelle qu'un problème est dit NP -complet, s'il est NP [1] et NP-dur [2]. Le problème Sat est le premier problème que l'on prouve être NP-complet. Ce qui nous permet ensuite de pouvoir réduire Sat d'autres problèmes et ainsi prouver qu'ils sont aussi NP-dur. (voir Appendice)

Théorème 1. Le problème Sat $\begin{cases} \text{entrée : une formule } \varphi \text{ en calcul propositionnel} \\ \text{sortie : oui si } \varphi \text{ est satisfiable, non sinon} \end{cases}$ est NP-complet.

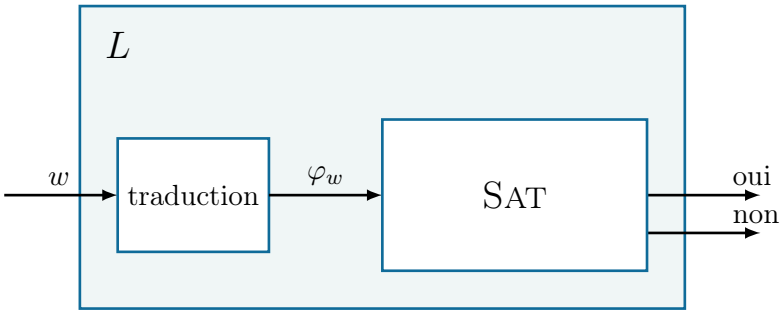
Démonstration.

Étape 1 : Sat $\in$ *NP*

Une valuation est un certificat polynomial, car on peut vérifier en temps polynomial qu'elle satisfait ou non la formule φ. Donc Sat $\in$ NP.

Étape 2 : Sat *est NP-dur*

Soit L un problème NP, on veut réduire L à Sat.



1. décidé par une machine de Turing non déterministe en temps polynomial
2. tout problème NP se réduit à lui en temps polynomial

On sait que L est décidé par une machine de Turing $M = (Q, \Sigma, \Gamma, q_0, q_f, \#, \delta)$ non déterministe. On suppose qu'elle a un seul état final q_f et qu'elle boucle sur cet état final. [3] De plus, M s'arrête en temps polynomial, donc il existe $P \in \mathbb{R}[X]$ tel que, pour tout $w \in L$ de taille n, le temps d'exécution de M sur w est inférieur à $P(n)$.

Pour tout $w = w_1 w_2 \dots w_n \in \Sigma^*$ tel que $|w| = n$,

$$w \in L \iff M \text{ s'arrête sur } w \iff M \text{ est dans l'état } q_f \text{ à l'étape } P(n)+1$$

On représente l'exécution de la machine M sur w par un tableau des configurations que l'on peut borner par $P(n)+1$ lignes (par définition de P) mais aussi par $P(n)+1$ colonnes car la complexité spatiale est inférieure à la complexité temporelle. [4] Chaque configuration est représentée dans le tableau par gqd où $q \in Q$ et $g, d \in \Gamma^*$, la tête de lecture de M est sur le premier caractère de d. Ainsi, on a un tableau de taille $(P(n)+1) \times (P(n)+1)$ avec des caractères de $A := \Gamma \cup Q$ dans chaque case. Par exemple :

q_0	w_1	w_2	$\cdots$	w_n	$\#$	$\cdots$	$\#$
γ_1	q_1	w_2	$\cdots$	w_n	$\#$	$\cdots$	$\#$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
γ_k	$\cdots$	γ_l	q_f	γ_m	$\cdots$	$\cdots$	γ_p

Tableau typique représentant la suite des configurations de la machine M

On veut construire une formule φ_w de calcul propositionnel de taille polynomiale telle que

$$\varphi_w \text{ est satisfiable} \iff w \in L \iff M \text{ est dans l'état } q_f \text{ à l'étape } P(n)+1$$

On introduit les variables popositionnelles $x_{i,j,a}$ pour $i, j \in [\![0, P(n)]\!]$ et $a \in A$ qui signifie "la case (i,j) du tableau contient le symbole a".

On pose la formule

$$\varphi_w = \underbrace{E}_{\text{Existence}} \wedge \underbrace{U}_{\text{Unicité}} \wedge \underbrace{I}_{\text{Initialisation}} \wedge \underbrace{F}_{\text{État final}} \wedge \underbrace{T}_{\text{Transition}}$$

avec

$$E \wedge U = \bigwedge_{i,j} \Big(\bigvee_{a \in A} x_{i,j,a} \wedge \bigwedge_{\substack{a,a' \in A \\ a \neq a'}} \neg(x_{i,j,a} \wedge x_{i,j,a'}) \Big)$$

qui est de taille $O((P(n)+1)^2(|A| + |A|^2))$

$$I = x_{0,0,q_0} \wedge \bigwedge_{j=1}^{n} x_{0,j,w_j} \wedge \bigwedge_{j=n+1}^{P(n)} x_{0,j,\#}$$

qui est de taille $O(P(n)+1)$

$$F = \bigvee_{j=0}^{P(n)} x_{P(n),j,q_f}$$

qui est de taille $O(P(n)+1)$

$$T = \bigwedge_{i=0}^{P(n)-1} \Big(\bigwedge_{j=0}^{P(n)} \big([(i,j) \notin C_{i,j}] \wedge [x_{i,j} = x_{i+1,j}] \big) \vee x_{i,j-1 \in Q}$$

3. il suffit de rajouter des transitions d'un ensemble d'états finals vers un état q_f pour chaque élément de Γ et de rajouter des transitions de q_f vers q_f pour chaque lettre de Γ pour faire boucler sur q_f

4. en effet, on ne pourrait pas atteindre la case $P(n)+2$ du ruban en $P(n)$ étapes

$$\vee \left(x_{i,j\in Q} \wedge [\delta^{\rightarrow}(i,j) \vee \delta^{\leftarrow}(i,j)]\right) \vee x_{i,j+1\in Q}\Big)$$

où $C_{i,j}$ correspond aux 6 cases $\begin{array}{c|ccc} i & * & q & * \\ i+1 & * & * & * \end{array}$ des lignes i et $i+1$.

On a utilisé les notations suivantes :

- $(i,j) \in C_{i,j} := x_{i,j-1\in Q} \vee x_{i,j\in Q} \vee x_{i,j+1\in Q}$ avec $x_{i,j\in Q} := \bigvee_{q\in Q} x_{i,j,q}$ (pour savoir si on est ou non dans $C_{i,j}$) qui est de taille $O(3|Q|)$
- $(i,j) \notin C_{i,j} := \neg[(i,j) \in C_{i,j}]$
- $x_{i,j} = x_{k,l} := \bigvee_{a\in A} (x_{i,j,a} \wedge x_{k,l,a})$ (pour tester l'égalité entre les caractères de deux cases) qui est de taille $O(|A|)$
- $\delta^{\leftarrow}(i,j) := \bigvee_{\substack{\gamma_1,\gamma_2\in\Gamma \\ q\in Q \\ \delta(q,\gamma_2)=(q',\gamma_3,\leftarrow)}} \left(x_{i,j-1,\gamma_1} \wedge x_{i,j,q} \wedge x_{i,j+1,\gamma_2} \wedge x_{i+1,j-1,q'} \wedge x_{i+1,j,\gamma_1} \wedge x_{i+1,j+1,\gamma_3}\right)$ qui est de taille $O(|\Gamma|^2|Q||\delta|)$
- $\delta^{\rightarrow}(i,j) := \bigvee_{\substack{\gamma_1,\gamma_2\in\Gamma \\ q\in Q \\ \delta(q,\gamma_2)=(q',\gamma_3,\rightarrow)}} \left(x_{i,j-1,\gamma_1} \wedge x_{i,j,q} \wedge x_{i,j+1,\gamma_2} \wedge x_{i+1,j-1,\gamma_1} \wedge x_{i+1,j,\gamma_3} \wedge x_{i+1,j+1,q'}\right)$ (pour simuler la fonction de transition)

Donc T est de taille $O((P(n)+1)^2(3|Q| + |A| + 3|Q| + 2|Q||\Gamma|^2|\delta|))$.

Ainsi φ_w est satisfiable si et seulement si $w \in L$. De plus, φ_w est de taille polynomiale par rapport à n donc la réduction est de taille polynomiale.

En effet si φ_w est satisfiable, alors $w \in L$, car il suffit de remplir le tableau avec le caractère a en case (i,j) si $x_{i,j,a}$ est évaluée à vrai. Par existence et unicité, pour tout (i,j) il y a exactement un caractère a qui va dans la case (i,j) et par $I \wedge F \wedge T$, le tableau représente bien une suite de configurations valides de M qui accepte w, d'où $w \in L$.

Réciproquement, si $w \in L$, on pose $\nu(x_{i,j,a}) = 1$ si et seulement si a est dans la case (i,j) ($\nu(x_{i,j,a}) = 0$ sinon). La valuation ν satisfait donc la formule φ_w car U, E, I, F, T sont satisfaite par ν car on a bien l'exécution de la machine M décrite dans le tableau.

□

Remarques :

- À partir de ce problème, on peut prouver que Cnf-Sat et 3-Sat sont NP-complets (on explicite ces problèmes dans la partie dédiée aux questions). Pour la réduction à Cnf-Sat, on utilise la Transformation de Tseitin. Pour la réduction de Cnf-Sat à 3-Sat, il suffit de trouver des conjonctions de clauses à 3 littéraux qui sont équivalentes aux clauses initiales. En voici un exemple :

$$x \simeq (x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z),$$

$$x \vee y \simeq (x \vee y \vee z) \wedge (x \vee y \vee \neg z),$$

$$x_1 \vee \cdots \vee x_n \simeq (x_1 \vee x_2 \vee y_2) \wedge (\neg y_2 \vee x_3 \vee y_3) \wedge (\neg y_3 \vee x_4 \vee y_4) \wedge \cdots \wedge (\neg x_{n-2} \vee x_{n-1} \vee x_n),$$

  où toutes les nouvelles variables que l'on utilise sont des variables fraiches (*i.e.* qui n'apparaissent pas dans la formule initiale).
- Le problème 3-Sat est important car c'est à partir de lui que l'on prouve la NP-complétude d'autres problèmes comme des problèmes sur les graphes, des problèmes de paquets, etc. Voir [**?**].

— Aujourd'hui, on utilise des SAT-solvers qui sont des algorithmes tentant de se rapprocher d'algorithmes polynomiaux afin de résoudre la question de satisfiabilité d'une formule. Par exemple, les algorithmes DPLL utilisent le backtracking pour résoudre SAT. Leur complexité peut être constante pour certaines entrées, mais exponentielles pour d'autres.

Astuces de l'agrégatif :

Il faut bien comprendre que dans la formule T lorsqu'on parcourt les lignes si (i,j) n'est pas dans $C_{i,j}$, on vérifie l'égalité entre les cases (i,j) et $(i+1,j)$, puis si à la case (i,j) on a un état q, on vérifie les formules de transition ($\delta^{\rightarrow}$ et $\delta^{\leftarrow}$) sur les 6 cases de $C_{i,j}$, et pour les deux cases mitoyennes de l'état, on ne vérifie rien, puisque les vérifications sont effectuées avec les formules de transition sur les 6 cases de $C_{i,j}$ à la fois.

La formule T est valide à l'intérieur du tableau, il en faudrait une légèrement différente pour traiter les bords.

Bien comprendre les formules, pour bien les retrouver. Il y a d'autres présentations possibles mais il faut faire attention au temps que l'on a pour bien mettre en avant la structure du développement (NP puis NP-dur) sans perdre trop temps pour ne pas bacler les derniers points. Je pense qu'il faut au moins mettre une des deux formules $\delta^{\rightarrow}$ et $\delta^{\leftarrow}$.

Questions possibles :

- Le problème Sat est-il décidable ?
- Une machine de Turing non déterministe qui décide un problème s'arrête-t-elle sur toutes ses entrées ?
- Comment transformer une machine de Turing quelconque en la machine de Turing considérée dans ce développement ?
- Comment prouver que le problème Cnf-Sat ci-dessous est NP-complet ?

  Cnf-Sat
  - entrée : une formule φ en calcul propositionnel sous forme normale conjonctive ;
  - sortie : oui si φ est satisfiable, non sinon.

- Comment prouver que le problème 3-Sat ci-dessous est NP-complet ?

  3-Sat
  - entrée : une formule φ en calcul propositionnel sous forme normale conjonctive avec 3 littéraux par clause ;
  - sortie : oui si φ est satisfiable, non sinon.

- Ecrire la formule T sur les bords.