Théorème de Cook

P. Berthomé

10 mai 2017

Résumé

Ce document présente un idée de la preuve du Théorème de Cook, en complément du cours de *Calculabilité et Complexité* de 4A STI.

1 Situation et énoncé

On rappelle qu'un problème $\mathcal{P}$ est *NP-Complet* si :

1. $\mathcal{P}$ est dans *NP* ;
2. Tout problème dans NP est polynomialement transformable en $\mathcal{P}$.

En fait, cette définition est un peu plus puissante que la définition initiale (on ne sait pas si la définition initiale est équivalente à celle-ci, mais elle permet de résoudre un certain nombre de cas.)

On rappelle qu'un langage L est polynômialement transformable en un autre, appelé L_0, s'il existe une machine de Turing $\mathcal{M}$ qui convertit en temps polynômial toute suite ω de l'alphabet de L en une suite ω_0 de l'alphabet de L_0 de sorte que :

$$\omega \in L \Longleftrightarrow \omega_0 \in L_0.$$

La difficulté initiale est de trouver un langage/problème qui est NP-complet. Les autres seront alors déduits par réduction polynomiale. En 1971, Stephen Cook (né en 1939 et toujours actif scientifiquement) a montré que le problème SAT était un de ces problèmes. Le résultat a été montré la même année par Leonid Levin (né en 1948), ce qui fait que le théorème est souvent appelé le *théorème de Cook-Levin*

Le problème SAT s'énonce simplement. On considère des formules booléennes de longueur n contenant un certain nombre de variables reliées par les connecteurs logiques (et, ou, non) avec éventuellement des parenthèses. Étant donné une formule booléenne f, le problème SAT consiste à savoir s'il existe une affectation des variables telle que la formule est vraie (on dit alors que la formule est *satisfiable*).

Par exemple, la formule $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ est satisfiable (en prenant par exemple $x_1 = T$ et $x_2 = T$) ; par contre la formule $g(x) = x \wedge \neg x$ n'est pas satisfiable.

Théorème 1 *Le problème SAT est NP-complet*

Les parties suivantes montrent comment s'articule la preuve de ce résultat.

2 SAT est dans NP

Afin de montrer cette partie, il faut trouver une machine de Turing non déterministe qui résout SAT. Premièrement, on va coder de manière directe sur le ruban d'entrée de machine de Turing la formule. Pour cela, on utilise l'alphabet $\Sigma = \{0, 1, +, *, \neg, (,)\}$, le symbole $+$ dénotera $\vee$ et $*$ dénotera $\wedge$. Les variables de la formule sont notées x_1, x_2 $\ldots x_n$; elles seront codées par l'écriture de leurs indices en base 2. Ainsi, la formule

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

sera codée

$$(1 + 10) * 11$$

Il faut donc trouver une machine de Turing non déterministe qui accepte les mots x sur Σ tels que :

1. x est une formule booléenne
2. x est satisfiable

2.1 Montrer que le mot d'entrée est une formule booléenne

On peut facilement construire une machine de Turing qui vérifie la syntaxe. Cependant, une autre méthode consiste à travailler par inclusion de langages. On construit une grammaire algébrique qui permet de générer toutes les formules logiques bien construites. La grammaire suivante d'axiome F le permet :

$$\begin{array}{lcl} F & \longrightarrow & F \vee F \\ F & \longrightarrow & F \wedge F \\ F & \longrightarrow & (F) \\ F & \longrightarrow & T \\ F & \longrightarrow & \neg T \\ T & \longrightarrow & 1S \\ S & \longrightarrow & 0S \\ S & \longrightarrow & 1S \\ S & \longrightarrow & \varepsilon \end{array}$$

Les langages algébriques sont reconnaissables par un automate à pile. Dans le cas présent, l'automate sera non déterministe car il faut *deviner* les bonnes dérivations. L'automate à pile est directement simulable par une machine de Turing.

De manière directe, on peut construire une machine de Turing déterministe qui vérifie la syntaxe correcte du mot. Cela prend un temps $O(n^2)$.

2.2 Montrer que la formule est satisfiable

Pour cela, on construit cette machine en deux temps : la première consiste à donner une affectation à chaque variable de manière non-déterministe ; la deuxième consiste à évaluer la formule logique ainsi créée.

Pour la première, on peut réaliser l'affectation de la manière suivante. Quand on rencontre une nouvelle variable, on décide d'une affectation (T pour Vrai et F pour Faux) de manière non déterministe. Ensuite, on propage cette affectation à toute la formule. On

recommence ensuite du début en recherchant la variable suivante. Différentes méthodes permettent de gérer le fait d'avoir des variables sur plusieurs caractères. L'un des plus simples consiste à remplacer tous les caractères par l'affectation. La simplification se fera ensuite lors de l'étape d'après. Cette étape prend $O(n^2)$.

À partir ce cet instant, il faut évaluer la formule ainsi obtenue. Là encore, l'évaluation prend $O(n^2)$ étapes élémentaires et se fait de manière déterministe.

3 Tout problème de NP est réductible à SAT

On considère un langage L' dans NP. Il est donc accepté par une machine de Turing non déterministe M en temps polynômial. Sans perte de généralité, on peut supposer que cette machine utilise un seul ruban semi-infini et une seule tête de lecture. On rappelle que les mimes qui permettent de réduire toute machine de Turing à une machine d'un tel format utilisent des réductions polynomiales.

Soit ω une entrée de L' de longueur n.

On cherche ici à trouver une transformation de ω en ω_0 telle que $\omega \in L'$ équivaut à ω_0 est satisfiable.

La machine de Turing M est décrite plus précisément par :

— 1 seul ruban semi-infini

— s états : q_1, ..., q_s

— un alphabet à m lettres X_1, ..., X_m ($X_1 = B$)

— cette machine reconnaît le langage en temps polynomial $p(n)$

Soit Q_0, ..., Q_q la suite des descriptions instantanées. On construit alors une expression booléenne ω_0 qui simule la suite des descriptions instantanées telle que :

— toute affectation à Vrai ou Faux des variables de ω_0 représente au plus une suite de descriptions instantanées de M possible (légale vis à vis du fonctionnement de M) ;

— ω_0 prend la valeur Vrai si et seulement si la suite Q_0, ..., Q_q que représente ω_0 est légale et Q_q représente une description instantanée où M est dans l'état d'acceptation.

3.1 Définition de variables booléennes

Pour construire la formule booléenne, on a besoin de variables qui vont indiquer l'état de la machine de Turing à chaque instant.

Valeur des cellules :

$$C(i, j, t) = T$$

ssi la i-ème cellule du ruban de M contient la lettre X_j au temps t. ($1 \leq i \leq p(n)$, $1 \leq j \leq m$, $1 \leq t \leq p(n)$).

On introduit ici $O((p(n))^2)$ variables (toujours polynomial)

État de la machine :

$$S(k, t) = T$$

ssi la machine M est dans l'état q_k à l'étape t.

On ajoute $O(p(n))$ variables.

Tête de lecture :

$$H(i,t) = T$$

ssi la RWH de M se trouve sur la cellule i au temps t.

On introduit ici $O((p(n))^2)$ variables.

Globalement, nous avons $O((p(n))^2)$ variables, ce qui est polynomial en n. Donc chaque variable dans la formule logique peut être représentée par une suite de 0 et de 1 de longueur petite ($O(\log_2(n))$)

3.2 Lemme intermédiaire

On définit la formule logique suivante à n variables :

$$U(x_1, x_2, \ldots x_n) = (x_1 + x_2 + \ldots + x_n) \prod_{1 \leq i < j \leq n} (\neg x_i + \neg x_j)$$

Théorème 2 *La formule $U(x_1, x_2, \ldots x_n)$ est vraie ssi une et une seule de ses variables est vraie.*

La preuve est simple, il suffit de regarder les deux parties. La première implique qu'au moins une variable soit Vraie, la deuxième indique que pour toute paire de variable, l'une des deux est fausse. Ceci implique simplement le résultat souhaité.

En ce qui concerne le codage de cette expression, si les variables sont des lettres, la longueur est $O(n^2)$; si chaque variable est codée sur α bits, la longueur totale est en $\alpha O(n^2)$ bits.

3.3 Décomposition du fonctionnement de la machine de Turing

L'expression booléenne recherchée se décompose en 7 propriétés principales qui expriment le fonctionnement de la machine de Turing non déterministe.

Propriété 1 *La RWH occupe une et une seule cellule dans chaque description instantanée.*

Pour chaque étape t, il suffit de considérer l'expression suivante :

$$A_t = U(H(1,t), H(2,t), \ldots, H(p(n),t))$$

Le théorème précédent montre que cette expression est vraie ssi une et une seule des variables $H(i,t)$ est vraie. Cette expression a pour longueur :

$$c \log_2 n \times O(p^2(n) = O(p^3(n))$$

On construit A comme le produit des expressions A_t. Cette expression est de longueur $O(p^4(n))$.

Propriété 2 *La $i$ème case du ruban contient une et une seule lettre au temps t.*

Sur le même principe, on construit :

$$B_{i,t} = U(C(i,1,t), C(i,2,t), \ldots, C(i,m,t))$$

et

$$B = \prod_{1 \leq i,t \leq p(n)} B_{i,t}$$

Propriété 3 *À chaque instant, M est dans un état et un seul.*

On construit l'expression C_t

$$C_t = U(S(1,t), S(2,t), \ldots, S(s,t))$$

et

$$C = \prod_{1 \leq t \leq p(n)} C_t$$

Propriété 4 *En une unité de temps, M change au plus le contenu d'une cellule du ruban*

On définit l'expression suivante qui indique qu'on ne pourra modifier que la cellule qui contient la RWH :

$$D(i,j,t) = \neg H(i,t) * C(i,j,t) \Rightarrow C(i,j,t+1)$$

En mettant le tout ensemble ;

$$D = \prod D(i,j,t)$$

Propriété 5 *Les transformations de lettres et état se font conformément au fonctionnement de M*

On définit l'expression $E_{i,j,k,t}$ qui indique que :
— la i-ème cellule ne contient pas la lettre X_j au temps t,
— **ou** la RWH n'est pas sur la case i au temps t,
— **ou** ma machine de Turing M n'est pas dans l'état q_k au temps t,
— **ou** la bonne transition s'applique.

$$\begin{aligned} E(i,j,k,t) &= \neg C(i,j,t) + \neg H(i,t) + \neg S(k,t) \\ &+ \sum_{l \in \Lambda} (C(i,j'_l,t+1) S(k_l,t+1) H(i+mvt_l,t+1)) \end{aligned}$$

avec $\delta(X_j, q_k) = \{X_{j'_l}, q_{k_l}, mvt_l \mid l \in \Lambda\}$

Propriété 6 *La description instantanée de départ est correcte*

$$F = S(1,1) * H(1,1) \prod_{1 \leq i \leq n} C(i,j,1) \prod_{n < i \leq p(n)} C(i,1,1)$$

Les quatre termes peuvent s'interpréter simplement comme suit :

— $S(1,1)$ donne l'état initial
— $H(1,1)$ donne la position de la RWH au début
— $\prod_{1\leq i\leq n} C(i,j,1)$ donne le mot d'entrée
— $\prod_{n<i\leq p(n)} C(i,1,1)$ indique que les autres cellules sont vides.

Propriété 7 *La machine de Turing s'arrête*

Si q_s est l'état d'acceptation

$$G = S(s, p(n))$$

3.4 Au final

On considère la formule logique :

$$\omega_0 = ABCDEFG$$

Sa longueur est de l'ordre de $O(p^4(n))$. Si la formule est satisfiable, on peut construire une exécution de la machine M (non déterministe qui termine sur l'entrée donnée.