Calculabilité et Complexité (11X008) - Printemps 2024

12. Théorème de Cook-Levin

Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

La stratégie habituelle pour montrer qu'un problème est NP-difficile est de choisir un autre problème NP-difficile et le réduire (en temps polynomial) au problème considéré. Mais comment a-t-on pu montrer cela la première fois, lorsqu'aucun autre problème n'existait pour effectuer une réduction ? L'objectif de ce dernier cours est de présenter le théorème de Cook-Levin, qui établit que le problème SAT est NP-difficile (bien sûr, il est aussi dans NP, donc NP-complet). Pour la définition de SAT, se référer au cours précédent (cours n°11).

Ce résultat a été obtenu à peu près en même temps (et indépendemment) par Stephen Cook (canadien) et Leonid Levin (alors soviétique) dans les années 70s. La preuve que l'on connaît le mieux est celle de Richard Karp, qui a adapté les idées de Cook.

12.1 Principe général

L'objectif est de montrer que tout langage dans NP peut se réduire à SAT. La preuve utilise la définition d'origine de la classe NP, à savoir les langages décidables en temps polynomial par une machine de Turing *non-déterministe*. Pour tout langage L dans NP, il existe une telle machine M qui décide L. On montre alors qu'étant donné une entrée w pour cette machine (une instance pour L), on peut construire en temps polynomial une formule SAT ϕ telle que ϕ est satisfaisable si et seulement si M accepte w.

12.2 Principe détaillé

Le fait que $L \in \mathsf{NP}$ nous dit qu'il existe une machine de Turing non-déterministe M et une borne $B = n^{O(1)}$ sur le temps d'exécution de M tels que $M(w)$ accepte ssi $w \in L$.

12.2.1 Préambule

Tout d'abord, restreignons le modèle en considérant une machine à une seule bande, sur laquelle le mot d'entrée est initialement écrit. On suppose également que M possède un état acceptant q_{accept} (comme au premier semestre). Ces hypothèses se font sans perte de généralité. Notons également w_i le i$^{\text{ème}}$ symbole du mot d'entrée et $n = |w|$ la taille de l'entrée. On a donc $w = w_1 \cdot w_2 \cdots w_n$.

Pour rappel, M accepte w si et seulement si au moins une branche d'exécution accepte. On ne connaît pas cette branche, mais on peut se la représenter mentalement comme une succession de configurations (état courant, contenu de la bande et position de la tête de lecture) satisfaisant certaines propriétés. Notamment, la première configuration est la configuration de départ de la machine, la dernière est une configuration dont l'état courant est q_{accept}, et chaque changement de configuration doit respecter la fonction de transition de la machine. Enfin, le nombre de configurations le long de cette branche est au plus de B.

Souvenons-nous aussi (cours de langages formels) que l'on peut représenter complètement une configuration de ce type en insérant l'état courant juste avant le symbole pointé par la tête de lecture. Par exemple, la configuration de départ s'écrit "$q_0 w_1 w_2 \cdots w_n \square \cdots$". La seconde configuration pourrait s'écrire (par exemple) à "$w_1 q_1 w_2 \cdots w_n \square \cdots$", etc.

Toujours mentalement, nous allons imaginer un tableau dont chaque ligne correspond à une configuration le long de la branche acceptante. Cela donne par exemple :

q_0	w_1	w_2	$\cdots$	w_n	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
w_1	q_1	w_2	$\cdots$	w_n	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
w_1	w_2	$\cdots$	w_n	1	0	0	1	q_{accept}	1	0	$\square$	$\square$	$\square$

12.2.2 La formule !

L'objectif est de créer une formule SAT ϕ qui représente implicitement tous les tableaux valides possibles, et telle que ϕ est satisfaisable si et seulement si un tel tableau existe. La formule sera gigantesque, mais polynomiale !

Notons i les indices du tableau en abscisse et j les indices en ordonnée. Clairement, $j \leq B$. On a aussi $i \leq B$ (car la machine ne peut faire qu'au plus un déplacement par opération). Notre formule doit encoder les contraintes suivantes :

1. La première ligne du tableau correspond à la configuration initiale
2. Il existe une ligne du tableau dont la configuration est acceptante
3. Chaque transition entre deux lignes respecte la fonction de transition de M
4. Une autre contrainte discutée plus tard

Chacune de ces contraintes va être encodée dans une partie différente de la formule, qui aura alors la forme générale :

$$\text{Contrainte 1} \wedge \text{Contrainte 2} \wedge \text{Contrainte 3} \wedge \text{Contrainte 4}$$

Nous pouvons maintenant traiter chaque contrainte séparément. Mais avant, définissons quelques variables que nous allons utiliser un peu partout. Pour chaque case i, j du tableau

et chaque valeur v possible dans une case, nous définissons une variable $X_{i,j,v}$, qui vaut vrai si la case (i, j) a la valeur v et faux sinon. Par exemple, dans le tableau ci-dessus, $X_{0,0,q_0}$ vaut `vrai`, mais $X_{2,1,\square}$ vaut `faux`.

Contrainte 1. C'est facile, il suffit de forcer les variables de la première ligne comme suit :

$$X_{0,0,q_0} \wedge X_{1,0,w_1} \wedge \cdots \wedge X_{n,1,w_n} \wedge X_{n+1,1,\square} \wedge \cdots \wedge X_{B,0,\square}.$$

Remarquez qu'il y a autant de clauses que de variables dans cette partie de la formule.

Contrainte 2. Ici, on veut simplement forcer au moins une case à contenir la valeur q_{accept}, quel que soit l'endroit (ce n'est pas forcément sur la ligne B, car B n'est qu'une *borne* sur le temps d'exécution, il se peut que l'exécution termine avant). On peut faire cela avec un OU sur toutes les cases, en demandant que la valeur de cette case soit q_{accept}. On peut noter cela de manière plus compacte comme suit :

$$\bigvee_{i \leq B, j \leq B} X_{i,j,q_{accept}}$$

Contrainte 3 Nous devons maintenant encoder le fait que chaque transition entre deux lignes respecte la fonction de transition de M. Tout d'abord, observez qu'une transition de la machine ne peut affecter que trois valeurs d'une ligne du tableau, car l'écriture se fait localement et il y a au plus un déplacement à chaque transition. Par exemple, une ligne $a, b, c, a, b, \boldsymbol{c, q_k, a}, b, c, a, b, c$ ne peut se transformer qu'en une ligne ou seules les valeurs en gras ont changé. Là encore, on ne sait pas quels seront ces trois symboles pour une ligne donnée, on doit donc considérer toutes les options possibles.

Cette partie n'est pas détaillée, mais on peut comprendre que c'est faisable. Essentiellement, pour chaque ligne i, pour chaque possibilité de trois cases consécutives dans cette ligne, et pour chaque valeur possible de ces trois symboles comprenant un état q_k de la machine, la ligne $i + 1$ doit avoir les mêmes valeurs en dehors de ces trois cases et doit avoir des valeurs sur ces trois cases qui correspondent à une transition valide depuis l'état q_k. Tout cela peut s'exprimer avec des $\vee$ et des $\wedge$ sur les valeurs des cases $X_{i,j,v}$. La taille de cette partie de la formule est grande, mais reste polynomiale en n.

Contrainte 4 Une dernière contrainte est nécessaire pour que la formule reflète réellement l'exécution. Il s'agit ici d'empêcher qu'une case du tableau ait deux valeurs différentes en même temps, par exemple, on veut interdire que $X_{i,j,v}$ et $X_{i,j,v'}$ puissent être toutes les deux à `vrai` si $v \neq v'$. Comment encoder cela ? Et bien en disant que pour toute case (i, j) et pour toute paire de valeurs v et v', au moins l'une des deux variables correspondantes est fausse.

Concrètement, en notant V l'ensemble des valeurs possibles d'une case, cela donne :

$$\bigwedge_{i\leq B, j\leq B, v\in V, v'\in V} (\overline{X_{i,j,v}} \vee \overline{X_{i,j,v'}})$$

Pour des raisons techniques, il faut également s'assurer que chaque case contient bien une valeur (potentiellement $\square$, c'est OK, ici c'est une valeur) :

$$\bigwedge_{i\leq B, j\leq B} \bigvee_{v\in V} X_{i,j,v}$$

(Litéralement, pour toute case, cette case a la première valeur ou la seconde valeur ou... etc.)

12.2.3 Discussions

La preuve donnée ci-dessus n'est pas une preuve complète et rigoureuse, mais elle devrait suffire à se persuader qu'on peut bien construire une formule ϕ à partir d'une machine non-déterministe M et d'un mot w, telle que ϕ est satisfaisable si et seulement si M accepte w.

Le fait que cette formule, bien que grande, soit polynomiale, résulte du fait que $B = n^{O(1)}$, et donc B^2 ou même $B^3 = n^{O(1)}$ également. Tous les autres facteurs qui viennent se rajouter à la complexité de la formule viennent du nombre d'état, du nombre de symboles dans l'alphabet, ou du nombre de transitions de la machine, qui sont tous *constants* (ne dépendent pas de n). La formule est donc bien polynomiale en la taille de l'entrée n. Elle peut également être produite en temps polynomial, ce qui implique que SAT est NP-difficile.

Fort heureusement, vous n'avons jamais besoin de refaire cette preuve. Une fois qu'un problème est NP-difficile, on peut l'utiliser pour faire d'autres réductions comme nous l'avons fait jusqu'à présent !