Mémento de théorie de la complexité

Gilles Zémor

23 septembre 2008

1 Problèmes algorithmiques

1.1 Du semi-formel au formel

Exemples de problèmes algorithmiques :

- Calculer le pgcd de deux entiers, deux polynômes,
- Trouver un exposant secret RSA,
- Factoriser un entier,
- Trouver le plus court chemin dans un graphe,
- Décider si un graphe contient un triangle,
- Décider si un entier est premier.

Les deux derniers problèmes sont des exemples de problèmes dits *de décision*. Ils se définissent par une collection I d'*instances* et une question Q à laquelle la réponse est «oui» ou «non».

I : un graphe G Q : le graphe G contient-il un triangle (trois sommets deux à deux adjacents) ?

La formalisation complète d'un problème de décision implique une convention sur le *codage des données*. Qu'entend-on par «un graphe G» ? Destinée à être traitée par une machine, une instance est représentée par une suite d'éléments d'un alphabet fini Σ, c'est-à-dire un *mot* de Σ^*. Par quelle convention représente-t-on un graphe ? Si $\Sigma = \{0, 1\}$, la donnée d'une matrice d'adjacence du graphe suffit pour le décrire complètement et nécessite environ n^2 bits. La donnée des listes d'adjacence (liste des voisins, pour chaque sommet) nécessite parfois plus, ($n^2 \log n$ si le nombre d'arêtes est élevé, nettement moins si le graphe comporte moins d'arêtes). On voit que la convention adoptée influence la *taille des données*.

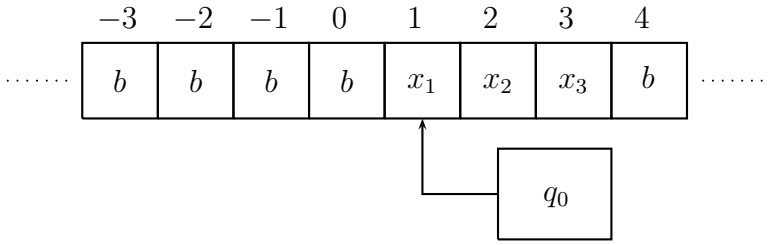

FIG. 1 – Machine de Turing dans l'état initial q_0

Formellement, si $x = x_1 \dots x_n$ est un mot de Σ^*, sa *taille* ou *longueur* est défini par le nombre n de symboles le constituant.

Une convention de codage des données étant adoptée ou sous-jacente, la formalisation d'un problème de décision est la donnée d'un *langage* L, c'est-à-dire d'une partie de Σ^* (un ensemble de mots). Les mots du langage L représentent les instances du problème de décision dont la réponse est «oui». Le complémentaire $\Sigma^* \setminus L$ de L représente la réunion des instances du problème dont la réponse est «non», et des mots ne représentant pas correctement une instance du problème. Les définitions formelles seront données pour des langages, mais nous utiliserons tout autant le vocabulaire des problèmes de décision.

1.2 Machines et algorithmes

Une machine de Turing M est schématisée (figure 1) par un *ruban*, représentant la mémoire de M, constitué d'un ensemble infini de cases, indexées par $\mathbb{Z}$. Elle est en outre munie d'un *curseur*, qui examine une case à la fois, sans connaître son numéro. La machine est dans un *état* qui est un élément d'un ensemble fini Q. Les symboles susceptibles d'être inscrit sur une quelconque case du ruban appartiennent à un alphabet Γ, dit *alphabet d'écriture*. La machine execute des *opérations élémentaires*. Une opération élémentaire consiste en les trois actions simultanées :

- remplacer le symbole lu par un symbole de Γ, différent ou identique au symbole lu,
- se déplacer d'une case vers la gauche, d'une case vers la droite, ou rester sur la même case,
- changer d'état.

Une opération élémentaire est entièrement déterminée par le symbole lu par le curseur et l'état dans lequel se trouve la machine. En résumé, la machine M est essentiellement la donnée de sa *fonction de transition*

$$\delta \ : \Gamma \times Q \to \Gamma \times Q \times \{-1, 0, 1\}.$$

	b	0	1
q_0	b,q_R,0	0,q_0,+1	1,q_1,+1
q_1	b,q_A,0	0,q_0,+1	1,q_1,+1

FIG. 2 – exemple de machine de Turing représentée par un tableau

Une machine M accepte des entrées $x \in \Sigma^*$ où Σ est *l'alphabet de lecture*. Le *calcul* associé est entièrement déterminé par l'entrée x. Il consiste en la suite d'itérations (opérations élémentaires) partant de la configuration initiale où
– les symboles de l'entrée $x = x_1 x_2 \cdots x_n$ occupent les cases $1, 2, \ldots, n$,
– les autres cases contiennent un symbole b appelé «blanc» appartenant à $\Gamma \setminus \Sigma$,
– la machine est dans un état privilégié q_0 appelé état initial.
Outre q_0, l'ensemble des états contient deux états privilégiés, appelés états finaux, q_A (accepte), q_R (rejette). Quand la machine aboutit dans un état final, elle arête de calculer. On appelle *temps de calcul sur l'entrée* x, et on note $t_M(x)$, le nombre d'opérations élémentaires menant de la configuration initiale à un état final. On peut avoir $t_M(x) = \infty$, ce qui veut dire que le calcul n'aboutit jamais dans un état final.

Définition 1 *On dit que la machine* M décide *le langage* $L \subset \Sigma^*$ *si*
– *sur toute entrée* $x \in L$*, la machine* M *aboutit à l'état* q_A*,*
– *sur toute entrée* $x \in \Sigma^* \setminus L$ *la machine aboutit à l'état* q_R*.*

Définition 2 *On dit que la machine* calcule *la fonction* $f : \Sigma^* \to \Sigma^*$ *si :*
– *sur toute entrée* x *du domaine de définition de* f*, la machine aboutit à l'état* q_A *et contient* $y_1, y_2, \ldots, y_n, b$ *dans les cases* $1, 2, \ldots, n, n+1$*, où* $y_1 y_2 \cdots y_n = f(x)$*.*
– *sur toute entrée* x *en dehors du domaine de définition de* f*, la machine aboutit à l'état* q_R*.*

Description d'une machine M**.** En résumé, un machine de Turing est donnée par ses deux alphabets Σ, Γ, de lecture et d'écriture, son ensemble d'états Q contenant l'état initial q_0 et les états finaux q_A et q_R, et enfin la fonction de transition δ. Toute machine M peut être décrite par un tableau T, dont les colonnes sont indexées par les symboles de Γ et les lignes par l'ensemble des états Q. L'entrée T_{ij} du tableau, pour un état i et un symbole j contient $\delta(j, i)$. Par exemple, la machine suivante (figure 2) décide si l'entrée $x = x_1 \ldots x_n \in \{0, 1\}^*$ se termine par $x_n = 1$.

Enfin, on pourra convenir d'un codage non ambigu d'une machine M, c'est-à-dire d'un tableau du type précédent, par un mot de Σ^*. En d'autres termes on conviendra d'un injection

$$\begin{array}{ccc} \mathcal{M} & \to & \Sigma^* \\ M & \mapsto & \langle M \rangle. \end{array}$$

de l'ensemble des machines de Turing vers Σ^*.

Machine de Turing universelle. On se convaincra de l'existence d'une machine U qui prend des entrées du type $(\langle M\rangle, x)$, et reproduit le calcul de M à partir de l'entrée x. En particulier U aboutit dans l'état q_A si et seulement si M aboutit dans l'état q_A à partir de x. Le temps de reproduction d'une opération élémentaire de M peut être rendu proportionnel à la longueur de l'entrée $(\langle M\rangle, x)$ de U.

2 Classes P et NP

La classe P est l'ensemble des langages décidables en temps polynomial, c'est-à-dire que L est dans P s'il existe un polynôme $p(X)$ et une machine de Turing M qui décide L tels que pour toute entrée $x \in \Sigma^*$,

$$t_M(x) \leqslant p(|x|)$$

où $|x|$ désigne la longueur de l'entrée x, c'est-à-dire le nombre de caractères qui compose la chaîne x.

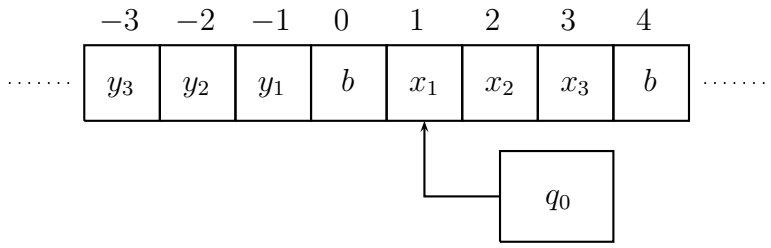

FIG. 3 – Machine de Turing non-déterministe dans l'état initial q_0, avec entrée $x = x_1x_2x_3$ et témoin $y = y_1y_2y_3$

Pour définir la classe NP, on introduit la notion de *Machine de Turing non-déterministe*. Une machine de Turing non-déterministe M est spécifiée de manière identique à une machine ordinaire (déterministe) M, c'est-à-dire par les alphabets Σ, Γ, l'ensemble des états Q, contenant q_0, q_A et q_R, et la fonction de transition δ. La seule différence est le début du calcul lorsqu'on lui soumet comme entrée x. Comme précédemment, la chaîne $x = x_1 \dots x_n \in \Sigma^*$ est inscrite dans les cases $1, 2, \dots, n$. Ensuite la machine procède à une étape d'écriture d'une chaîne arbitraire de symboles $y = y_1 \dots y_m$ de Γ^* dans les cases $-1, -2, \dots, -m$. Ensuite seulement, le curseur se met en position 1, la machine se met dans l'état initial q_0, et démarre son calcul comme une machine de Turing ordinaire. La chaîne y est appelée «certificat» ou «témoin», et des valeurs différentes de y peuvent causer,

pour une même entrée x, la machine M à aboutir dans l'état q_A, dans l'état q_R, ou même calculer indéfiniment sans jamais aboutir à un état final.

On notera $t_M(x, y)$ le nombre d'opérations élémentaires que met la machine non-déterministe M à aboutir un état final (q_A ou q_R) à partir de l'état q_0, sur entrée x et certificat y. On peut avoir $t_M(x, y) = \infty$ si la machine n'aboutit jamais dans un état final.

Définition 3 *On appelle classe* NP *l'ensemble des langages L pour lesquels il existe un polynôme p et une machine non-déterministe M tels que : $x \in L$ si et seulement s'il existe $y \in \Gamma^*$ vérifiant*
– M aboutit à l'état q_A sur l'entrée x associée au certificat y,
– $t_M(x, y) \leqslant p(|x|)$.

Informellement, il s'agit des langages admettant au moins un certificat d'appartenance, vérifiable en temps polynômial en la taille de l'entrée x. Dans le vocabulaire des problèmes de décision, il s'agit des problèmes admettant systématiquement une preuve que la réponse est «oui», vérifiable rapidement.

Remarque 1. Si la réponse est «non», (si l'entrée x n'est pas dans le langage), rien n'est exigé en ce qui concerne l'appartenance à la classe NP. Il n'existe pas forcément de certificat vérifiable rapidement de non-appartenance.

Remarque 2. On peut toujours supposer que le certificat y est court, par exemple $|y| \leqslant p(|x|)$, car la machine n'a pas le temps de lire des symboles de y qui seraient dans des cases au-delà de son temps de calcul imposé : ces symboles n'auraient donc aucune influence sur le calcul acceptant le couple (x, y), et peuvent aussi bien être supprimés dès le départ.

Quelques exemples de problèmes de la classe NP.

Le problème SAT (Satisfaisabilité).

I : Un ensemble fini V de variables
une formule booléenne F sous la forme
$C_1 \wedge C_2 \wedge \ldots \wedge C_k$
où C_i (une clause) est de la forme $Y_1 \vee \ldots \vee Y_k$
avec $Y_j = v$ ou $Y_j = \overline{v}$, $v \in V$.
Q : La formule F est-elle satisfaisable ? C'est-à-dire,
existe-t-il un $|V|$-uple de valeurs dans $\{0, 1\}^{|V|}$
rendant la formule F «vraie» (de valeur 1).

Dans ce cas le certificat y est donné par un des choix dans $\{0, 1\}^{|V|}$ rendant F vraie.

Le problème 3-SAT .

Il est identique que le précédent, à ceci près que les clauses impliquent toutes

trois variables exactement, c'est-à-dire sont de la forme $Y_1 \vee Y_2 \vee Y_3$.

Le problème CLIQUE.

I : un graphe G, un entier k
Q : le graphe G contient-il une clique à k sommets ?
(un sous-ensemble de k sommets deux à deux adjacents) ?

Le certificat y est simplement donné, lorsqu'elle existe, par la clique.

3 Transformations polynômiales, problèmes NP-complets

Définition 4 *On appelle* transformation *(ou réduction)* polynômiale *d'un langage L vers un langage L' est une fonction $f : \Sigma^* \to \Sigma^*$, calculable en temps polynômial en la taille de l'entrée, telle que*

$$x \in L \quad \textit{si et seulement si} \quad f(x) \in L'.$$

S'il existe une transformation polynômiale de L vers L' on note $L \propto L'$.

Exemple d'une transformation polynômiale de 3-SAT vers CLIQUE. Soit $F = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ une formule à k clauses. Il s'agit de la transformer en un graphe qui contient une k-clique si et seulement si f est satisfaisable. On définit un graphe G à $3k$ sommets où chaque clause donne naissance à trois sommets, indexés par les variables de la clause. L'ensemble des arêtes est défini ainsi : il y a une arête entre deux quelquonques sommets de G, sauf si
– les deux sommets sont issus de la même clause,
– les deux sommets sont issus de clauses différentes, mais sont indexés, l'un par une variable x, l'autre par la variable $\overline{x}$.

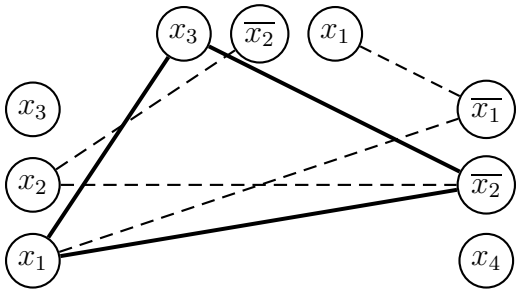

FIG. 4 – le graphe provenant de la formule $(x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \overline{x_2} \vee x_1) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$. En pointillé sont indiquées les arêtes interdites. Le triangle détermine un choix de valeurs satisfaisant F, $x_1 = 1, x_3 = 1, \overline{x_2} = 1$.

Un exemple de tel graphe est donné sur la figure 4.

– EXERCICE 1. *Montrer que la transformation définie ci-dessus est bien une transformation polynômiale de 3-SAT vers ClIQUE. Le caractère calculable en temps polynomial est clair, il s'agit de montrer que F est satisfaisable si et seulement si G contient une k-clique.*

Définition 5 *On dit que le langage Λ est* NP-complet *si pour tout $L \in$ NP on a*

$$L \propto \Lambda.$$

Existence de problèmes NP-complets. On considère le problème de décision suivant :

Arrêt en temps limité.

I : la description $\langle M \rangle$ d'une machine M, $x \in \Sigma^*$, 1^t
(t symboles «1» consécutifs)
Q : Existe-t-il $y \in \Sigma^*$ tel que la machine M accepte l'entrée x
et le certificat y en temps $\leqslant t$?

Théorème 6 *le problème Λ «Arrêt en temps limité» est* NP*-complet.*

Preuve : Vérifions tout d'abord que le langage Λ auquel on a affaire est bien dans NP. Pour cela il s'agit de se convaincre qu'il existe une machine de Turing non-déterministe U et un polynôme p tels que :

pour toute instance positive $X \in \Sigma^*$ du problème, il existe un certificat $Y \in \Gamma^*$, tel que U aboutit à l'état q_A («accepte») en un temps $\leqslant p(|X|)$.

Or, une instance positive X du problème est la donnée de $(\langle M \rangle, x, 1^t)$ pour lesquels il existe un certificat y, tel que M aboutit à l'état q_A en un temps $\leqslant t$ sur l'entrée (x, y).

Dans ces conditions, le couple U, Y qui convient est donné par
– $Y = y$,
– une machine U qui étant donnés $X = \langle M \rangle, x, 1^t$ et y, *reproduit* le comportement de M sur l'entrée (x, y). On se convaincra que le temps nécessaire pour reproduire une opération élémentaire de M est proportionnel à la longueur de l'entrée (X, y). La machine U est programmée pour reproduire t opérations élémentaires de M, puis de regarder si M est alors dans l'état q_A : si oui, U entre dans l'état «accepte», sinon elle entre dans l'état «rejette».

On voit que U reconnait l'appartenance de X au langage en un temps $\leqslant ct|X| \leqslant c|X|^2$ où c est une constante. Le langage Λ est donc dans NP.

Pour montrer maintenant que Λ est NP-complet, il s'agit d'exhiber, pour tout langage L de NP, une transformation polynômiale de L vers Λ. Soit donc un

langage L de NP, reconnu par une machine non-déterministe M_L. Ceci veut dire qu'il existe un polynôme p_M tel que, pour tout $x \in L$, il existe un certificat y tel que M accepte (x, y) en temps $\leqslant p_M(|x|)$. La transformation sera définie par, pour toute entrée $x \in \Sigma^*$,

$$x \mapsto \langle M_L \rangle, x, 1^{p_M(|x|)}.$$

Cette fonction est manisfestement calculable en temps polynomial. On a $x \in L$ si et seulement si $f(x) \in \Lambda$ par définition de M_L ∎

On établit sans peine la proposition suivante.

Proposition 7 *La relation $\propto$ est transitive. Si $L \propto L'$ et $L' \propto L''$ alors $L \propto L''$. En particulier si Λ est* NP*-complet et $\Lambda \propto L$ alors L est* NP*-complet.*

4 Pertinence de la question P=NP pour la cryptographie

Donnons la définition formelle suivante de fonction «à sens unique» (one-way).

Définition 8 *On appelle* fonction à sens unique *une fonction $f : \Sigma^* \to \Sigma^*$ vérifiant les propriétés suivantes.*

1. *Le domaine de définition de f est infini et pour tout x tel que $f(x)$ est défini, $f(x)$ est calculable en temps polynômial en $|x|$,*
2. *il n'existe pas d'algorithme qui, sur toute entrée y de l'image de f, donne en temps polynomial en $|y|$ un $x \in \Sigma^*$ tel que $f(x) = y$,*
3. *il existe un polynôme p tel que, pour tout $y \in \Sigma^*$ dans l'image de f, il existe un $x \in \Sigma^*$ tel que*
   - *$f(x) = y$*
   - *$|x| \leqslant p(|y|)$.*

La dernière condition, $|x| \leqslant p(|y|)$, est destinée à éliminer de la définition des fonctions à sens unique «triviales», comme

$$x \mapsto \lfloor \log |x| \rfloor$$

qui ne sont pas inversibles en temps polynomial juste parce que la *longueur* d'un quelconque antécédent de y n'est pas polynomiale en $|y|$.

La simple question «existe-t-il une fonction à sens unique ?» est centrale en cryptographie moderne. Il se trouve que c'est exactement la même question que «P≠NP ?».

Théorème 9 *Il existe une fonction à sens unique si et seulement si* P $\neq$ NP.

Preuve : Soit $f : \Sigma^* \to \Sigma^*$ une fonction vérifiant les conditions *1.* et *3.* de la définition d'une fonction à sens unique. Considérons le problème de décision associé P_f suivant :

I : les chaînes de symboles $y, z \in \Sigma^*$.
Q : Existe-t-il $x \in \Sigma^*$ tel que :
$f(zx) = y$, où zx désigne la concaténation des chaînes z et x,
$|zx| \leqslant p(y)$?

Le problème P_f est clairement dans NP, le certificat d'appartenance étant donné par la chaîne x. Supposons maintenant P=NP. Le problème P_f doit alors être décidable en temps polynomial. On en déduit le calcul d'un antécédent de y par f en temps polynomial : pour cela, on fait grandir progressivement un préfixe z de l'antécédent de y. Étant donné connu un z pour lequel il existe x tel que $f(zx) = y$, on soumettra à l'algorithme qui décide P_f les instances $y, z0$ et $y, z1$. On reçoit la réponse «oui» dans au moins un des cas, et on est assuré d'avoir trouvé un antécédent de y en au plus $2p(|y|)$ appels à l'algorithme qui décide P_f. La fonction f n'est donc pas à sens unique.

Réciproquement, supposons P$\neq$NP. Soit donc un langage $L \in$ NP\P. Soit une bijection

$$\begin{array}{rcl} \phi : \Sigma^* & \to & \Sigma^* \times \Sigma^* \\ z & \mapsto & \phi(z) = (\phi_1(z), \phi_2(z)) \end{array}$$

telle que ϕ et ϕ^{-1} soient calculables en temps polynomial (une telle fonction existe !). Soit maintenant la fonction $f : \Sigma^* \to \Sigma^*$ définie par

$$f(x) = \phi_1(x)$$

si $\phi_1(x) \in L$ et si $\phi_2(x)$ est un certificat d'appartenance à L vérifiable en temps polynomial. Si $\phi_1(x)$ et $\phi_2(x)$ ne vérifient pas ces conditions, on déclare x en dehors du domaine de définition de f. La fonction f vérifie clairement les conditions *1.* et *3.* de la définition d'une fonction à sens unique. Remarquons de plus que tout $y \in L$ est dans l'image de f. En effet, y admet un certificat z d'appartenance à L vérifiable en temps polynomial, et $\phi^{-1}(y, z)$ est un antécédent de y par f. Supposons maintenant que f ne vérifie pas la condition *2.* de la définition d'une fonction à sens unique. Dans ce cas, un antécédent x de y est calculable en temps polynomial, et $\phi_2(x)$ est un certificat d'appartenance à L pour y, vérifiable en temps polynomial, et calculable à partir de x, et donc à partir de y, ce qui implique $L \in$ P. Contradiction. Donc la fonction f est à sens unique. ∎

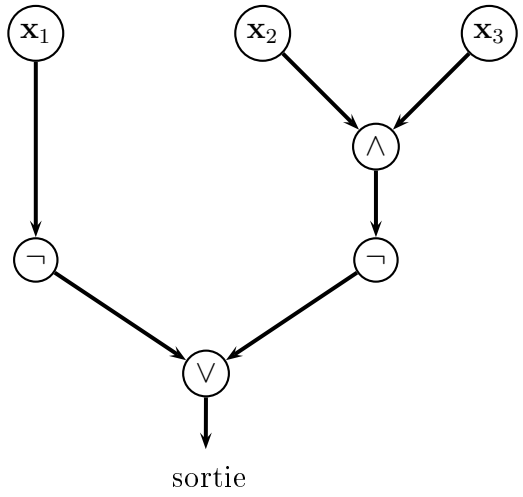

FIG. 5 – Exemple de circuit calculant la fonction booléenne f sur $\{0,1\}^3$ définie par $f(111) = 0$ et $f(x_1x_2x_3) = 1$ pour $x_1x_2x_3 \neq 111$.

5 Théorème de Cook

Théorème 10 (Cook) *Le langage (problème) SAT est* NP*-complet.*

Nous démontrons ce théorème en plusieurs étapes. Introduisons tout d'abord la notion de circuit de calcul.

Un *circuit* de calcul est un graphe orienté, dont les sommets sont étiquetés par un des termes $0, 1, \vee, \wedge, \neg, \mathbf{x}_1, \ldots, \mathbf{x}_n$, «sortie». De plus,
- Les sommets étiquetés $0, 1, x_i$ ont 0 comme degré rentrant.
- Les sommets étiquetés $\neg$ ont 1 comme degré rentrant.
- Les sommet étiquetés $\vee, \wedge$ ont 2 comme degré rentrant.
- Il y a un unique sommet étiqueté «sortie», il a 1 comme degré rentrant, et 0 comme degré sortant. Les sommets $\mathbf{x}_1, \ldots, \mathbf{x}_n$ sont appelés *sommets entrées*.

On dira que le circuit *calcule* la fonction booléenne $f : \{0,1\}^n \to \{0,1\}$ si pour tout $(x_1, \ldots, x_n) \in \{0,1\}^n$ soumis aux sommets entrées, la valeur de la sortie, après exécution de chacune des portes intermédiaires, égale $f(x_1, \ldots, x_n)$. Un exemple est représenté sur la figure 5.

Le problème SATC de la satisfaisabilité en circuit est une variante du problème SAT.

Le problème SATC (Satisfaisabilité en circuit).

I : Un circuit C à n entrées.
Q : Existe-t-il un choix de $(x_1, \ldots, x_n) \in \{0,1\}^n$ pour lequel la valeur de la sortie égale 1 ?

Le langage SATC est clairement dans NP.

– Exercice 2. *Montrer que $SAT \propto SATC$.*

Proposition 11 *$SATC \propto SAT$.*

Preuve : Il s'agit d'exhiber une transformation polynômiale de SATC vers SAT. Le principe en est le suivant : tout circuit C est transformé en une formule f à $n + \alpha$ variables, où $\{1, 2, \ldots, \alpha\}$ est l'ensemble des arcs du circuit qui ne sont pas issus d'un sommet «entrée». Les n sommets «entrée» $\mathbf{x}_1 \ldots \mathbf{x}_n$ donnent naissance à n variables $x_1, \ldots, x_n$. Les autres variables sont des variables auxiliaires, $y_1 \ldots y_\alpha$, où y_i joue un rôle de support pour la valeur intermédiaire du calcul qui circule sur l'arc i. Chaque sommet du graphe (une porte) qui n'est pas une entrée donne naissance à autant de sous-formules qu'il a de descendants, où chaque sous-formule est satisfaite si et seulement si la sortie de la porte respecte la manière dont le circuit traite les entrées de la porte.

Plus précisément, à chaque arc i ne provenant pas d'un sommet entrée est associée la variable y_i. À chaque arc issu d'un sommet $\mathbf{x}_i$ est associée la variable x_i.

Si une porte $\neg$ a un arc rentrant associé à la variable x et un arc sortant associé à la variable y, alors la sous-formule associée est

$$(x \vee y) \wedge (\overline{x} \vee \overline{y})$$

qui est satisfaite exactement lorsque la valeur de «sortie» y est le complémentaire de la valeur de «rentrée» x. Si une porte $\wedge$ a des arcs rentrants associés aux variables x et y et un arc sortant associé à la variable z, alors la sous-formule associée est

$$(\overline{x} \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee y \vee \overline{z})$$

qui est satisfaite exactement lorsque la valeur de «sortie» z égale $x \wedge y$. Si une porte $\vee$ a des arcs rentrants associés aux variables x et y et un arc sortant associé à la variable z, alors la sous-formule associée est

$$(\overline{x} \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y \vee z) \wedge (x \vee y \vee \overline{z})$$

qui est satisfaite exactement lorsque la valeur de «sortie» z est égale à $x \vee y$. Enfin, l'arc menant au sommet «sortie» du circuit est associé à la variable z, alors la sous-formule associée, satisfaite si et seulement si la valeur de z est 1 est tout simplement

$$z.$$

De même, l'arc i sortant d'un sommet «constante» (étiqueté 0 ou 1) donne naissance à la sous-formule y_i si la constante est 1, et $\overline{y}_i$ si la constante est 0.

La formule complète f issue du circuit est le «et» logique $\wedge$ de toutes les sous-formules précédemment citées.

On constate que, si le circuit est satisfaisable, c'est-à-dire qu'une certaine valeur de l'entrée $(x_1, \ldots x_n)$ du circuit lui fait sortir la valeur «1», alors ces mêmes valeurs des variables x_i, associées aux valeurs y_i qui correspondent aux valeurs intermédiaires circulant sur les arcs $1, 2, \ldots \alpha$, satisfont la formule f. Réciproquement, si un choix des valeurs de $x_1, \ldots, x_n$, $y_1 \ldots y_\alpha$ satisfait la formule f, alors l'entrée $(x_1, \ldots x_n)$ associée fait sortir 1 au circuit. En d'autres termes, on a bien affaire à une réduction, le circuit est satisfaisable si et seulement si la formule est satisfaisable. La transformation est clairement calculable en temps polynomial en la taille du circuit. ∎

Lemme 12 *Soit* $f : \{0,1\}^n \to \{0,1\}^n$ *une fonction booléenne. Il existe un circuit de complexité (nombre de sommets)* $O(2^n)$ *qui calcule* f.

Preuve : Il suffit de constituer

$$\bigvee_{y=(y_1 \ldots y_n), f(y)=1} \left(\bigwedge_{y_i=1} x_i \bigwedge_{y_i=0} \overline{x}_i \right).$$

∎

Théorème 13 *Le problème SATC est* NP*-complet.*

Preuve : Soit L un langage de la classe NP. Nous allons exhiber une réduction polynomiale de L vers SATC. Il s'agit donc d'associer à tout mot $x \in \Sigma^*$ un circuit tel que $x \in L$ si et seulement si $x \in L$.

Dire que L est dans NP veut dire qu'il existe une machine M_L et un polynôme p tels que, x est dans L si et seulement s'il existe $y \in \Sigma^*$ tel que M_L entre dans l'état «accepte» au bout d'un temps $\leqslant p(|x|)$. On peut en particulier supposer $|y| \leqslant p(|x|)$. Considérons le *tableau* (T_{ij}), de dimension $(t+1) \times (2t+1)$ où $t = p(|x|)$, défini de la manière suivante. appelons n la cardinalité de l'ensemble Q des états de M_L.

La ligne i du tableau constitue une représentation de l'état du ruban de calcul de la machine M_L au temps i, dans les positions

$$-t, -(t-1), \ldots, -1, 0, 1, \ldots, t.$$

Plus précisément, la quantité T_{ij} est la donnée du triplet (a_{ij}, h_{ij}, q_{ij}) où

1. a_{ij} est le symbole de l'alphabet d'écriture Γ se trouvant dans la case j du ruban, au temps i.

2. h_{ij} vaut 0 ou 1. $h_{ij} = 1$ si le curseur se trouve au temps i en position j, $h_{ij} = 0$ sinon.
3. q_{ij} donne le numéro $q = 1, 2, \ldots, n$ de l'état dans lequel se trouve la machine au temps i, *si le curseur est en position i, i.e. si* $h_{ij} = 1$, sinon $q_{ij} = 0$.

Chaque entrée T_{ij} du tableau appartient donc à un ensemble *fini* F qui ne dépend que de la machine M_L.

Le point clé de la preuve est la constatation suivante. Pour chaque $i \geqslant 1$, et chaque $j \neq -t, t$, l'entrée T_{ij} est entièrement déterminée par les entrées $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$:

$$T_{ij} = f(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1})$$

où f est une fonction de $F \times F \times F$ dans F. Les entrées $T_{i,-t}$ et $T_{i,t}$ sont, de manière similaire, des fonctions de $T_{i,-t}, T_{i,-t+1}$ et de $T_{i,t-1}, T_{i,t}$ respectivement.

Il existe donc un *circuit* de taille *finie* qui prend comme entrée

$$T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$$

et calcule T_{ij}. On réalisera donc, par concaténation de tous ces petits circuits, un circuit C qui
- prend comme entrée tous les contenus possibles des cases $-t, -(t-1), \ldots, -1$ au temps $i = 0$,
- pour lequel les contenus des cases $0, 1, \ldots, t$ au temps $i = 0$ sont des constantes,
- qui calcule la dernière ligne du tableau,
- puis fait sortir 1 si une des entrées T_{tj} est telle que q_{tj} représente l'état «accepte», et 0 sinon.

On constate que le circuit calcule 1 sur l'entrée $Y_{-t}, \ldots, Y_{-1}$ si et seulement s'il existe un certificat $y = y_1 \ldots y_k$, $k \leqslant t$, d'appartenance à L, où $Y_{-1} = y_1, \ldots, Y_{-k} = y_k$, et $Y_{-k+1} \ldots Y_{-t}$ sont égaux au symbole blanc b. ∎

Le théorème précédent, conjointement avec la réduction SATC $\propto$ SAT, démontrent le théorème de Cook. Nous avons la variante suivante :

Théorème 14 *Le problème 3-SAT est* NP*-complet.*

Preuve : Il suffit de réduire SAT à 3-SAT. Pour cela on utilise la transformation suivante. Soit $f = C_1 \wedge \ldots \wedge C_k$ une formule booléenne, instance de SAT. Il s'agit de la transformer en une instance de 3-SAT.

Si C_i est une clause de la forme $x \vee y$ on la remplace par la clause

$$(x \vee y \vee z) \wedge ((x \vee y \vee \overline{z})$$

où z est une variable auxiliaire.

Si C_i est une clause de la forme x, on se ramène au cas précédent en la remplaçant par $(x \vee y) \wedge (x \vee \overline{y})$ où y est une variable auxiliaire.

Si C_i est une clause de la forme $x_1 \vee x_2 \vee x_3 \vee x_4$, on la remplace par la clause

$$(x_1 \vee x_2 \vee y) \wedge (x_3 \vee x_4 \vee \overline{y})$$

où y est encore une variable auxiliaire.

On utilise une suite de transformations similaires, si C_i est une clause égale au «$\vee$» de cinq variables ou plus. Il est assez aisé de vérifier que l'on a bien affaire à une transformation polynomiale. ∎

6 La classe PSPACE

La classe PSPACE est la classe des langages $L \subset \Sigma^*$ pour lesquels il existe un polynôme p et une machine de Turing M tels que : l'entrée $x \in \Sigma^*$ appartient à L si et seulement si
- la machine M entre dans l'état «accepte» au bout d'un temps fini,
- au cours de calcul le curseur ne sors jamais de l'ensemble des cases numérotées $-p(|x|), \ldots, _1, 0, 1, \ldots, p(|x|)$.

En d'autres termes, la classe PSPACE caractérise les problèmes décidables en n'utilisant qu'une *mémoire* (ou un *espace*) polynomiale en la taille des données.

Comme une machine de Turing ne peut passer que d'une case à une case voisine en une opération élémentaire, on a clairement P $\subset$ PSPACE. On a plus :

Proposition 15 NP $\subset$ PSPACE.

Preuve : À partir d'une machine non-déterministe M qui décide l'appartenance au langage L de NP, il suffit de construire un algorithme déterministe qui décide $x \in L$ en examinant successivement, *dans l'ordre* (par exemple lexicographique), tous les certificats potentiels y, puis en simulant M sur l'entrée (x, y). Il suffit de garder en mémoire le *numéro* du certificat y en train d'être examiné : la taille du numéro identifiant y étant proportionnel à $|y|$, il s'agit d'une quantité polynomiale en $|x|$. La mémoire totale requise reste donc polynomiale en $|x|$. ∎

Proposition 16 *Soit $L \in$ PSPACE. Soit M une machine qui décide l'appartenance à L en espace polynomial. Il existe un polynôme q tel que l'on ait la majoration du temps de calcul sur l'entrée x : $t_M(x) \leqslant 2^{q(|x|)}$.*

Preuve : Considérons le tableau (T_{ij}) de configurations de la machine M défini pendant la démonstration du théorème 13, à ceci près que nous modifions les dimensions du tableau. Si l'espace nécessaire au calcul sur l'entrée x est majoré par $p(|x|)$, on prendra le nombre de colonnes du tableau égal à $(2t+1)$ où $t = p(|x|)$. Comme nombre de lignes on prendra un majorant m du nombre de contenus possibles pour une ligne. On voit qu'au bout d'un temps $\leqslant (2t+1)|Q||\Gamma|^{2t+1}$, où Q est l'ensemble des états de M, la machine doit avoir accepté (ou rejetté) x, sinon il existe un temps pour lequel la ligne du tableau T est égal à une ligne précédemment rencontrée, ce qui veut dire que la machine M boucle et reproduit indéfiniment une suite finie de configurations, sans jamais accepter ou rejetter. ∎

Le problème suivant joue, pour la classe PSPACE, un rôle équivalent à celui de SAT pour la classe NP.

FBQ (Formules Booléennes Quantifiées)

I : Un ensemble fini $\{x_1, \ldots x_n\}$ de variables et une affirmation
de la forme $\forall x_i \exists x_j \ldots F(x_1 \ldots x_n)$
où $F(x_1 \ldots x_n) = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ est une formule booléenne.
Q : L'affirmation (ou formule quantifiée) est-elle vraie ?

Définition 17 *On dit que le langage* Λ *est* PSPACE*-complet si*
– Λ *est dans* PSPACE
– *Pour tout* $L \in$ PSPACE, $L \propto \Lambda$.

Le théorème suivant est l'équivalent du théorème de Cook.

Théorème 18 *Le problème FBQ est* PSPACE*-complet.*

Idée de la preuve : Soit $L \in$ PSPACE et soit une machine M qui décide l'appartenance à M en espace polynomial. Soit, comme précédemment le tableau (T_{ij}) de configurations de la machine M où la dernière ligne est dans l'état «accepte» si et seulement si l'instance est dans L. D'après la proposition 16 nous pouvons supposer les dimensions du tableau $2^t \times t$ où t est polynômial en la taille de l'entrée. Il s'agit de coder le tableau de configurations (T_{ij}) par une formule quantifiée de taille polynomiale en l'entrée. L'idée est la suivante : étant données deux lignes possibles du tableau ℓ_1 et ℓ_2, c'est-à-dire simplement des suites de t symboles représentant des entrées du tableau, et étant donné un entier m, nous voulons associer une formule quantifiée $\phi_{\ell_1,\ell_2,m}$ qui soit vraie si et seulement si la machine M passe de la ligne ℓ_1 à la ligne ℓ_2 en un nombre d'étapes $\leqslant m$. Montrons comment récursivement construire une telle formule. Pour $m = 1$ nous mimons de près la technique employée pour démontrer le théorème 13. Supposons maintenant que

nous savons construire une telle formule pour m et montrons comment construire une formule pour $2m$. Une idée qui marche presque est d'écrire

$$\phi_{\ell_1,\ell_2,2m} = \exists x\ \phi_{\ell_1,x,m} \wedge \phi_{x,\ell_2,m}$$

où x est un ensemble de variables nécessaire pour représenter une ligne du tableau (T_{ij}). Cette idée ne marche pas telle quelle car on voit que la taille de la formule double à chaque fois que l'on double m. À l'aide du quantificateur $\forall$, nous remplaçons la formule ci-dessus par une formule équivalente qui ne fait qu'ajouter une constante à la longueur de la formule. Écrivons

$$\phi_{\ell_1,\ell_2,2m} = \exists x\ \forall(y,z)[(y = \ell_1, z = x) \vee (y = x, z = \ell_2)] \rightarrow \phi_{y,z,m}$$

Cette fois la formule pour m n'a été utilisée qu'une seule fois. On n'a rajouté qu'une constante fois le nombre de variables nécessaires pour encoder une ligne du tableau. Des techniques standard permettent de réécrire cette formule sous forme correcte (tous les quantificateurs en tête de la formule). ∎

7 Algorithmes probabilistes : les classes RP et BPP

Un *algorithme probabiliste* est une machine de Turing qui, sur l'entrée $x \in \Sigma^*$, commence par choisir une suite *aléatoire* $y_1 \dots y_q$ de symboles de Σ, de longueur q polynomiale en la taille $|x|$ de l'entrée, et puis ensuite procède à un calcul habituel. La seule différence entre une machine probabiliste et les machines nondéterministes de la section 2 est que le certificat $y_1 \dots y_q$ n'est plus fourni par un oracle, mais est choisi aléatoirement, ce qui permet de donner un sens à la *probabilité* qu'au bout d'un temps polynomial la machine accepte l'entrée x.

Définition 19 *On dit que le langage $L \subset \Sigma^*$ est dans la classe* RP *s'il existe un polynôme p et une machine de Turing probabiliste M tels que :*
– *la machine M accepte ou rejette toute entrée x en un temps $\leqslant p(|x|)$,*
– *si $x \notin L$, la machine rejette x avec probabilité 1,*
– *si $x \in L$, la machine accepte x avec probabilité $\geqslant 1/2$.*

Exemples : les tests classiques de *non*-primalité (Solovay-Strassen, Miller-Rabin) sont des algorithmes probabilistes au sens de la classe RP définie ci-dessus. Le langage est constitué des entiers *composés*. Si un entier n'est pas dans le langage, il est premier, et le test ne le déclare *jamais* composé (avec probabilité 1 donc). Des propriétés arithmétiques de $\mathbb{Z}/n\mathbb{Z}$ permettent d'établir que si l'entier est composé (dans le langage, donc) l'algorithme le décèle avec probabilité au moins $1/2$.

Notons que chaque fois qu'un algorithme probabiliste du type RP apporte une réponse «oui», on dispose d'un certificat d'appartenance au langage, i.e. d'une preuve déterministe, vérifiable en temps polynomial. En particulier on a :

Proposition 20 RP $\subset$ NP.

En effet, tout $x \in L$, pour $L \in$ RP, admet un moins un certificat $y_1 \dots y_q$ vérifiable en temps polynomial, puisqu'au moins la moitié des mots $y_1 \dots y_q$ sont de tels certificats.

Notons encore qu'en répétant l'algorithme probabiliste k fois, en choisissant les certificats potentiels $y_1 \dots y_n$ de manière *indépendante* (au sens des probabilités), on trouvera un certificat d'appartenance à L avec une probabilité $1 - 1/2^k$.

Il est tentant de considérer (et c'est ce que l'on fait en pratique) qu'un test de non-primalité a une valeur comme test de primalité. Plus généralement, l'algorithme probabiliste qui décide l'appartenance à $L \in$ RP, décide aussi, mais en un sens quelque peu plus faible, de la *non*-appartenance à RP. Après k applications de l'algorithme se concluant toutes par un rejet, on est convaincu qu'avec une probabilité $\geqslant 1 - 1/2^k$ l'entrée est dans $\overline{L}$. La différence est que l'algorithme ne fournit pas de certificat, i.e. pas de preuve courte d'appartenance à $\overline{L}$. Il n'y a d'ailleurs pas de raison pour que $L \in$ RP implique $\overline{L} \in$ NP. On en conclut cependant, que quitte à renoncer à disposer d'une preuve déterministe courte d'appartenance au langage L, on peut disposer de preuves *probabilistes* d'appartenance à des langages *en dehors* de la classe NP. On définit la classe suivante, a priori plus vaste que RP et appelée BPP, de langages décidables par des algorithmes probabilistes.

Définition 21 *On dit que le langage $L \subset \Sigma^*$ est dans la classe* BPP *s'il existe un polynôme p et une machine de Turing probabiliste M tels que :*
– *la machine M accepte ou rejette toute entrée x en un temps $\leqslant p(|x|)$,*
– *si $x \notin L$, la machine rejette x avec probabilité $\geqslant 2/3$,*
– *si $x \in L$, la machine accepte x avec probabilité $\geqslant 2/3$.*

Retenons que RP $\subset$ BPP (clairement) et que l'on est susceptible de trouver dans BPP des langages a priori en dehors de NP. Donnons-en un exemple.

Un *programme de branchement* (branching program) est un graphe orienté sans circuit, possédant
– un sommet racine r, de degré rentrant 0 et de degré sortant 1,
– deux sommets terminaux, de degré sortant 0, étiquettés 0 et 1,
– des sommets intermédiaires, étiquettés par des symboles $x_1 \dots x_n$, et possédant chacun deux arcs sortants, l'un étiquetté par 0, l'autre par 1.
Un exemple en est donné sur la figure 6.

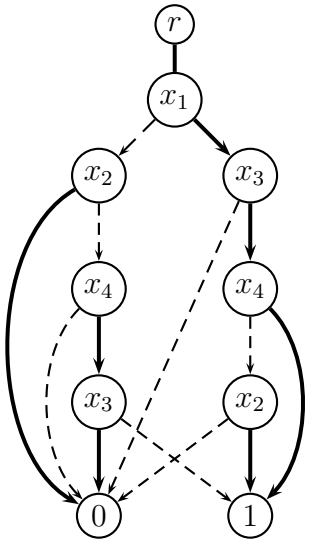

FIG. 6 – Programme de branchement. Les arcs pointillés correspondent à l'étiquette 0, les autres à l'étiquette 1.

Un programme de branchement est un mode de calcul d'une fonction booléenne. Un choix des variables $x_1, \ldots, x_n$ détermine un chemin dans le graphe, et son sommet terminal, 0 ou 1, donne la valeur de la fonction pour ce choix du n-uple $(x_1, \ldots, x_n)$. Nous nous restreindrons aux programmes dits «à lecture unique», c'est-à-dire ceux pour lesquels aucun chemin ne rencontre plus d'une fois une même étiquette x_i. Le programme de la figure 6 est à lecture unique.

Intéressons-nous au problème de décision suivant :

I : Deux programmes de branchement, B_1 et B_2, à lecture unique.
Q : Les programmes B_1 et B_2 déterminent-ils la même fonction booléenne ?

Ce problème ne paraît pas être dans NP, car on ne voit pas comment démontrer, de manière déterministe, que deux programmes sont équivalents sans parcourir tous les chemins possibles dans les graphes, ce qui n'est pas polynomial en la taille des données.

Il existe, cependant, un algorithme probabiliste pour ce problème qui en fait un membre de la classe BPP. Pour décrire cet algorithme il nous faut définir le polynôme P_B associé au programme B.

Le polynôme P_B est un polynôme en les variables $x_1 \ldots x_n$. Il est défini récursivement, en attribuant de proche en proche une valeur aux sommets du programme de branchement, ainsi qu'aux arêtes. À la racine du programme, ainsi qu'à son successeur, on attribue la valeur 1. Puis, si la valeur P est attribuée à un sommet s étiqueté x_i, on attribue Px_i à l'arc étiqueté 1 issu de s et $P(1-x_i)$ à l'arc étiqueté 0 issu de s. À un sommet s on attribue la somme des valeurs polynomiales

des arcs qui aboutissent à s. Enfin, le polynôme $P_B(x_1, \ldots, x_n)$ est déclaré être la valeur du sommet terminal 1. Par exemple, pour le programme de la figure 6, on a :

$$P_B(x_1, \ldots, x_n) = x_1x_3x_4 + x_1x_3(1 - x_4)x_2 + (1 - x_1)(1 - x_2)x_4(1 - x_3). \quad (1)$$

Remarquons que :

1. le polynôme $P_B(x_1, \ldots, x_n)$ est une somme de «monômes» (en x_i ou $(1-x_i)$) qui correspondent chacun à un chemin du programme d'extrémité 1.
2. Le programme est à lecture unique veut dire que le polynôme P_B est de degré 1 au plus en chacune des variables x_i.
3. Évalué en un quelconque élément de $\{0, 1\}^n$, l'addition et la multiplication étant effectuées dans $\mathbb{Z}$, le polynôme $P_B(x_1, \ldots, x_n)$ prend la valeur de la fonction booléenne représentée par le programme de branchement ; en particulier $P_B(x_1, \ldots, x_n)$ vaut 0 ou 1.

Pour décider si deux programmes de branchement représentent la même fonction booléenne, il suffit donc de trouver si les deux polynômes associés sont égaux. Notons que la complexité de la représentation polynômiale des deux programmes, au sens de l'écriture du type de l'exemple (1), est comparable à la complexité des programmes. On pourrait être tenté de développer les polynômes pour les écrire sous forme algébrique normale afin de voir s'ils sont égaux. Le problème est que la forme algébrique normale de l'écriture polynômiale d'un programme de branchement peut très bien se révéler de taille exponentielle en n, même si le programme ne l'est pas. L'approche probabiliste que nous décrivons ci-après consiste à *évaluer* les deux écritures polynômiales, en des valeurs *autres que* 0 *ou* 1.

Proposition 22 *Soient P et Q deux représentations polynomiales, à n variables, de deux programmes de branchement B_1 et B_2. Soit $r_1, \ldots, r_n$ des valeurs aléatoires uniformes et indépendantes choisies dans un corps $K = \mathbb{Z}/p\mathbb{Z}$ où p est un nombre premier $\geqslant kn$.*

- *Si les fonctions booléennes représentées par P et Q (ou les programmes de branchement correspondants) sont identiques, alors $P(r_1, \ldots, r_n) = Q(r_1, \ldots, r_n)$ dans K avec probabilité 1.*
- *Si les fonctions booléennes représentées par P et Q sont différentes, alors $P(r_1, \ldots, r_n) \neq Q(r_1, \ldots, r_n)$ dans K avec probabilité $\geqslant 1/k$.*

Pour démontrer la proposition nous avons besoin d'invoquer le lemme suivant.

Lemme 23 *Si K est un corps fini et si P est polynôme non nul de $K[X_1, \ldots, X_n]$ de degré au plus 1 en chaque X_i, alors on a*

$$\mathrm{P}[P(r_1, \ldots, r_n) = 0] \leqslant \frac{n}{|K|}$$

lorsque $(r_1, \ldots, r_n)$ est choisi aléatoirement et uniformément dans K^n.

Preuve : Si $n = 1$, le polynôme P possède au plus une racine, puisque son degré est au plus 1, ce qui démontre le lemme dans ce cas. Soit $n > 1$ et supposons le lemme démontré pour les polynômes à $n - 1$ variables. On peut écrire $P(X_1, \ldots, X_n)$ sous la forme

$$P(X_1, \ldots, X_n) = P_1(X_1, \ldots, X_{n-1}) + P_2(X_1, \ldots, X_{n-1})X_n.$$

On a, en notant E l'événement $P_2(r_1, \ldots, r_{n-1}) = 0$ et $\overline{E}$ l'événement complémentaire,

$$\begin{aligned}
\mathrm{P}[P(r_1, \ldots, r_n) = 0] &= \mathrm{P}[P(r_1, \ldots, r_n) = 0 \mid E]\mathrm{P}[E] \\
&\quad + \mathrm{P}[P(r_1, \ldots, r_n) = 0 \mid \overline{E}]\mathrm{P}[\overline{E}] \\
&\leqslant \mathrm{P}[E] + \mathrm{P}[P(r_1, \ldots, r_n = 0 \mid \overline{E}] \\
&= \mathrm{P}[E] + \mathrm{P}[r_n = -P_1(r_1, \ldots, r_{n-1})/P_2(r_1, \ldots, r_{n-1})] \\
&= \mathrm{P}[E] + \frac{1}{|K|} \\
&\leqslant \frac{n-1}{|K|} + \frac{1}{|K|}
\end{aligned}$$

en appliquant à $\mathrm{P}[E]$ l'hypothèse de récurrence. D'où le résultat. ∎

Preuve de la proposition 22 : Supposons d'abord que les programmes de branchement représentent la même fonction. Appelons «monôme» chaque produit en x_i ou $(1 - x_i)$ dont les écritures polynomiales P et Q sont une somme. Remarquons que si un monôme de P comporte, soit x_i soit $(1 - x_i)$ pour tout i, alors ce monôme représente une valeur de $(x_1, \ldots, x_n)$ pour laquelle la fonction booléenne vaut 1. Nous dirons qu'un tel monôme est un monôme *complet*. Maintenant, si un monôme μ de P ne contient ni x_i, ni $(1 - x_i)$, pour un ensemble I d'indices i, réécrivons-le

$$\mu \prod_{i \in I} (x_i + 1 - x_i)$$

et développons suivant chaque $x_i + (1 - x_i)$ pour obtenir une somme de $2^{|I|}$ monômes complets. Considérons maintenant les écritures polynomiales $\tilde{P}$ et $\tilde{Q}$ déduites de P et de Q par le processus de réécriture que nous venons d'expliciter. Celles-ci sont chacune égales à une somme de monômes complets qui représentent

l'ensemble des valeurs des n-uples $(x_1, \ldots, x_n)$ pour lesquelles la fonction booléenne égale 1. Comme $\tilde{P}$ et $\tilde{Q}$ représentent la même fonction booléenne les écritures polynomiales $\tilde{P}$ et $\tilde{Q}$ sont égales. Ceci veut dire que P et Q sont des écritures polynômiales, éventuellement différentes, du *même* polynôme de $\mathbb{Z}[X_1, \ldots, X_n]$. On en déduit que $P(r_1, \ldots, r_n) = Q(r_1, \ldots, r_n)$ pour tout $(r_1, \ldots, r_n)$ de $\mathbb{Z}^n$ et donc pour tout $(r_1, \ldots, r_n)$ de $(\mathbb{Z}/p\mathbb{Z})^n$.

Supposons maintenant que les programmes de branchement représentent des fonctions booléennes différentes. Dans ce cas les polynômes P et Q sont différents. Le polynôme $P - Q$ n'est donc pas le polynôme nul et il est de degré 1 en chaque x_i. Le résultat découle alors du lemme 23. ∎

8 La classe IP

La classe IP est une généralisation étonnante de la classe BPP qui contient la classe NP. Pour l'introduire introduisons la notion d'*isomorphisme* de graphes.

Soient G_0 et G_1 deux graphes. Soit ϕ une bijection de l'ensemble des sommets de G_0 vers l'ensemble des sommets de G_1. On dit que ϕ est un *isomorphisme* du graphe G_0 sur le graphe G_1 si :

pour toute paire de sommets x, y de G_0, il existe une arête dans G_0 entre x et y si et seulement s'il existe une arête dans G_1 entre $\phi(x)$ et $\phi(y)$.

On considère maintenant les problèmes de décision suivants.

ISO (Isomorphisme de graphes)

- I : Un graphe G_0 et un graphe G_1, déterminés par leurs matrices d'adjacence.
- Q : Les graphes G_0 et G_1 sont-ils isomorphes (il existe un isomorphisme de G_0 sur G_1) ?

Non-ISO (Non isomorphisme de graphes)

- I : Un graphe G_0 et un graphe G_1,
- Q : Les graphes G_0 et G_1 sont-ils non-isomorphes ?

Le problème ISO est clairement dans NP. Un certificat d'appartenance au langage est simplement la donnée d'un isomorphisme. Par contre, le problème complémentaire ne semble être, ni dans NP, ni dans BPP. Remplaçons l'idée d'un accès à un oracle qui donnerait un certificat $y_1 \ldots y_q$ d'appartenance, par celle d'une machine P, à la puissance de calcul illimitée. La notion d'algorithme polynomial non-déterministe qui demanderait un certificat à P est remplacé par un *protocole* entre une machine V, à la puissance de calcul polynomiale en la taille de l'entrée, susceptible d'interagir avec P, et de faire des choix aléatoires. On appelle P le

prouveur et V le *vérificateur*.

Illustrons la notion par un exemple. Le protocole suivant «certifie» le fait que deux graphes G_0 et G_1 sont non-isomorphes.

1. Le vérificateur V choisit un bit aléatoire ε, égal à 0 ou à 1. Puis il choisit un isomorphisme aléatoire de G_ε, ce qui créé un graphe Γ. le prouveur V donne le graphe Γ au prouveur P.
2. Le prouveur P répond en donnant la valeur de ε (0 ou 1).

Si les deux graphes sont effectivement non-isomorphes, alors un et un seul des deux graphes G_0 et G_1 est isomorphe à Γ. Le prouveur P qui a une puissance de calcul illimitée sait déterminer lequel, et peut ainsi trouver la valeur de ε sans erreur.

Par contre, si les deux graphes *sont isomorphes*, alors le graphe Γ est une copie isomorphe aléatoire *à la fois* de G_0 et de G_1. Si la copie Γ de G_ε a été choisie avec une loi uniforme, alors le prouveur P n'a aucune information à sa disposition lui permettant de privilégier l'une des deux hypothèses $\varepsilon = 0$ ou $\varepsilon = 1$. Sa puissance de calcul ne lui est d'aucune utilité ! Il doit donner la mauvaise réponse avec probabilité au moins $1/2$.

Au bout de k applications du protocole, le vérificateur qui reçoit toujours la bonne valeur de ε est convaincu que, soit il s'est produit un événement de probabilité $1/2^k$, soit les deux graphes G_0 et G_1 sont isomorphes.

Plus généralement, un protocole, ou *preuve interactive*, généralise la notion d'algorithme de la manière suivante : elle consiste en la donnée de *deux* machines de Turing P et V séparées, mais qui acceptent une entrée commune x. En outre de leurs rubans de travail habituels, elles disposent chacune de rubans de communication. Chaque machine est susceptible d'écrire sur un de ces rubans des messages destinés à l'autre machine. Chaque machine peut lire les messages qui lui sont destinés et s'en servir dans la suite de ses calculs. La machine P n'a pas de contrainte de temps de calcul mais le nombre total de messages transmis de P vers V ou de V vers P, ainsi que le temps de calcul de V doivent être polynomiaux en la taille de l'entrée initiale x. À la fin du calcul, la machine V accepte ou rejette l'entrée x.

On appelle IP la classe des langages L pour lesquels il existe un protocole, que l'on notera $(P \leftrightarrow V)$, qui distingue les cas $x \in L$ et $x \notin L$ de la manière suivante.

- Si $x \in L$, alors le protocole $(P \leftrightarrow V)$ aboutit à ce que V accepte x avec probabilité $\geqslant 2/3$.
- Si $x \notin L$, alors, quelle que soit la machine $\tilde{P}$, le protocole $(\tilde{P} \leftrightarrow V)$ aboutit à ce que V accepte x avec probabilité $\leqslant 1/3$.

On notera la similarité de cette définition avec celle de la classe BPP. Encore une

fois, les valeurs 2/3 et 1/3 sont arbitraires. Elles peuvent être remplacées par des valeurs p, q quelconques, $1 \geqslant p > 1/2 > q \geqslant 0$.

On a coutume de dire qu'un protocole est *complet*, s'il vérifie la première de ces propriétés. On dit que le protocole est *correct*, ou *valide* (sound) s'il vérifie la deuxième propriété.

Le problème suivant est dans la classe IP grâce à un protocole qui utilise la technique d'arithmétisation de la proposition 22.

On considère le problème de décision suivant, noté #SAT.

I : Une formule booléenne ϕ à n variables $x_1, \ldots, x_n$
et un entier k,
Q : Le nombre de choix de $(x_1, \ldots, x_n)$ satisfaisant ϕ
est-il égal à k ?

Ce problème ne semble pas être dans NP, ni même dans BPP. On peut tout de même décider de l'appartenance au langage associé de manière interactive. Décrivons le protocole. Tout d'abord, prouveur et vérificateur commencent par adopter une représentation polynômiale $p(x_1, \ldots, x_n)$ de la formule booléenne. Celle-ci est construite récursivement en appliquant les règles suivantes, où a et b sont des sous-formules,

$$\begin{aligned} a \wedge b &\to ab \\ \overline{a} &\to 1 - a \\ a \vee b &\to a + b - ab \end{aligned}$$

Le calcul de $p(x_1, \ldots, x_n)$ se fait clairement en temps polynomial. Il est important d'avoir à l'esprit que le polynôme $p(x_1, \ldots, x_n)$ est vu comme élément de $\mathbb{Z}[X_1 \ldots, X_n]$, c'est-à-dire qu'il se fonde sur l'addition et la multiplication dans $\mathbb{Z}$ (et non pas par exemple dans $\mathbb{Z}/2\mathbb{Z}$). Dans ces conditions, on constate que la valeur du polynôme $p(x_1, \ldots, x_n)$, évalué en tout n-uple de $\{0, 1\}^n$, coïncide avec l'évaluation de la formule booléenne ϕ (0 ou 1) en ce même n-uple.

On remarquera de plus que, pour chaque x_i, le degré du polynôme $p(x_1, \ldots, x_n)$ en x_i est majoré par la longueur de la formule, appelons-là ℓ.

Définissons les quantités suivantes :

$$\begin{aligned}
f_0() &= \sum_{(a_1 \ldots a_n) \in \{0,1\}^n} p(a_1, \ldots, a_n) \\
f_1(x_1) &= \sum_{(a_2 \ldots a_n) \in \{0,1\}^{n-1}} p(x_1, a_2, \ldots, a_n) \\
&\vdots \\
f_i(x_1, \ldots, x_i) &= \sum_{(a_{i+1} \ldots a_n) \in \{0,1\}^{n-i}} p(x_1, \ldots, x_i, a_{i+1}, \ldots, a_n) \\
&\vdots \\
f_n(x_1, \ldots, x_n) &= p(x_1, \ldots, x_n).
\end{aligned}$$

Ainsi $f_0()$ est la quantité que P doit démontrer être égale à k. La quantité f_i est un polynôme de $\mathbb{Z}[X_1, \ldots, X_i]$ d'autant plus difficile à calculer que i est petit. Notons que l'on a la relation :

$$f_i(x_1, \ldots, x_i) = f_{i+1}(x_1, \ldots, x_i, 0) + f_{i+1}(x_1, \ldots, x_i, 1). \tag{2}$$

P et V se mettent d'accord sur un nombre premier q. Les polynômes f_i seront aussi évalués dans $\mathbb{F}_q$. Le protocole est le suivant :

Étape 0. Le prouveur P donne à V la quantité $P_0 = f_0()$. Le vérificateur V vérifie que $k = P_0$. Si ce n'est pas le cas il rejette l'entrée (ϕ, k).

Étape 1. P donne à V le polynôme $P_1(X) = f_1(X) \in \mathbb{Z}[X]$. V vérifie que le degré de $P_1(X)$ est majoré par ℓ, évalue $P_1(0)$ et $P_1(1)$ et vérifie que

$$P_0 = P_1(0) + P_1(1).$$

Puis, V choisit un r_1 aléatoire dans $\mathbb{F}_q$ et l'envoie à P.

Étape 2. P donne à V le polynôme $P_2(X) = f_2(r_1, X) \in \mathbb{F}_q[X]$. V vérifie que le degré de $P_2(X)$ est majoré par ℓ, évalue $P_1(r_1)$, $P_2(0)$ et $P_2(1)$ dans $\mathbb{F}_q$ et vérifie que

$$P_1(r_1) = P_2(0) + P_2(1).$$

Puis, V choisit un r_2 aléatoire dans $\mathbb{F}_q$ et l'envoie à P.

$\vdots$

Étape i. P donne à V le polynôme $P_i(X) = f_i(r_1, \ldots, r_{i-1}, X) \in \mathbb{F}_q[X]$. V vérifie que le degré de $P_i(X)$ est majoré par ℓ, évalue $P_{i-1}(r_{i-1})$, $P_i(0)$ et $P_i(1)$ dans $\mathbb{F}_q$ et vérifie que

$$P_{i-1}(r_{i-1}) = P_i(0) + P_i(1).$$

Puis, V choisit un r_i aléatoire dans $\mathbb{F}_q$ et l'envoie à P.

$\vdots$

Étape n.

P donne à V le polynôme $P_n(X) = f_n(r_1, \ldots, r_{n-1}, X) \in \mathbb{F}_q[X]$. V vérifie que le degré de $P_n(X)$ est majoré par ℓ, évalue $P_{n-1}(r_{n-1})$, $P_n(0)$ et $P_n(1)$ dans $\mathbb{F}_q$ et vérifie que

$$P_{n-1}(r_{n-1}) = P_n(0) + P_n(1).$$

Puis enfin, V choisit un r_n aléatoire dans $\mathbb{F}_q$, évalue $P_n(r_n)$, et compare cette quantité avec $p(r_1, \ldots, r_{n-1}, r_n)$. Si cela concorde il accepte l'entrée (ϕ, k).

Complétude. Dire que ce protocole est complet veut simplement dire que si P suit à la lettre les instructions, elles seront acceptées par V, pour toute entrée (ϕ, k) du langage #SAT, c'est-à-dire dès lors que k est bien le nombre de n-uples satisfaits par la formule ϕ. Ceci découle directement de la propriété (2) qui, si elle a lieu dans $\mathbb{Z}$, a lieu dans $\mathbb{Z}/q\mathbb{Z}$.

Validité. Ce point met en valeur toute la subtilité du protocole. Il s'agit de voir que, quelles que soient les données intermédiaires qui lui seront fournies, par un «faux» prouveur $\tilde{P}$ qui respecte ou ne respecte pas les règles, le vérificateur n'acceptera une entrée (ϕ, k) qui n'est pas dans le langage qu'avec une probabilité très faible.

Pour montrer la validité on raisonne ainsi : supposons que k ne soit pas le nombre de n-uples satisfaisant ϕ. pour que V accepte l'étape 1, le prouveur doit donner une valeur $\tilde{P}_0$ différente de $f_0()$, puisqu'on doit avoir $\tilde{P}_0 = k$ et que $k \neq f_0()$ (cette dernière quantité étant la vraie valeur du nombre de n-uples satisfaisant ϕ). De $\tilde{P}_0 \neq P_0$ et de $P_0 = P_1(0) + P_1(1)$ on déduit $\tilde{P}_0 \neq P_1(0) + P_1(1)$. Donc, le prouveur doit donner à l'étape 1 un polynôme $\tilde{P}_1(X)$ *différent* de $P_1(X)$. Si la cardinalité q du corps $\mathbb{Z}/p\mathbb{Z}$ est suffisamment grande, alors avec probabilité proche de 1 on a $\tilde{P}_1(r_1) \neq P_1(r_1)$. En effet, le degré de ces polynômes étant majoré par ℓ, il y a au plus ℓ valeurs de r_1 telles que $\tilde{P}_1(r_1) - P_1(r_1) = 0$. La probabilité de cet événement est donc au plus ℓ/q. Si cet événement ne se produit pas on a $\tilde{P}_1(r_1) \neq P_2(0) + P_2(1)$, et le prouveur doit donner à l'étape 2 un polynôme $\tilde{P}_2(X)$ différent de $P_2(X)$. Ce phénomène se propage à travers les n étapes et à la fin, avec probabilité proche de 1, le vérificateur constate que $\tilde{P}_n(r_n) \neq p(r_1, \ldots, r_{n-1}, r_n)$. La seule possibilité pour que V accepte l'entrée (ϕ, k) est qu'à une des étapes i il choisisse malencontreusement un r_i tel que $que\tilde{P}_i(r_i) = P_i(r_i)$. La probabilité que cet événement se produise est inférieure à $n\ell/q$. On peut la rendre arbitrairement petite en choisissant q suffisamment grand.

Références

[1] M. R. Garey, D. S. Johnson, *Computers and Intractability : A guide to the theory of NP-completeness*, Freeman, 1979.

[2] O. Goldreich, *Complexity Theory - Material*, `http://www.wisdom.weizmann.ac.il/~oded/cc.html`

[3] M. Sipser, *Introduction to the theory of computation*, Thomson Course Technology, 2005.

[4] G. Zémor, *Cours de cryptographie*, Cassini, 2000.