Non déterminisme & réductions

Corrigé partiel de la feuille de travaux dirigés n°2

17 octobre 2017

Dans toute la suite, nous parlons de *langages* au lieu de *problèmes*. Dans le polycopié de cours, toutes ces notions ont été vues en termes de problèmes. Les deux notions sont équivalentes car tout problème peut être codé comme un langage dont les instances positives ont les mots du langage et les instances négatives dans le complémentaire de ce langage. Pour plus de précision, on peut se reporter à [4].
On rappelle la définition d'une transformation polynomiale :

Définition 1 *Soient $L_1 \subseteq \Sigma_1^\star$ et $L_2 \subseteq \Sigma_2^\star$, deux langages. Une **transformation polynomiale** de L_1 vers L_2 (notée $L_1 \propto L_2$) est une fonction $f : \Sigma_1^\star \to \Sigma_2^\star$ qui satisfait :*

1. *f est calculable en temps polynomial par un algorithme déterministe ;*
2. *$f(x) \in L_2 \Leftrightarrow x \in L_1$.*

1. Montrer que **SSP** est dans NP revient à construire une machine de Turing non déterministe qui accepte le langage qui code notre problème.
Pour avoir une machine plus simple, on supposera que les tailles et la capacité sont donnés en unaire. L'entrée de la machine de Turing est alors la suivante :

\$	C	♯	s_1	♯	s_2	♯	...	♯	s_n	\$

On supposera aussi que la machine de Turing dispose de deux rubans. Son scénario est alors le suivant :
— elle recopie C sur le ruban 2 en l'effaçant du ruban 1 ;
— pour chaque s_i, elle peut soit effacer un nombre correspondant de 1 sur le ruban 2 soit passer à s_{i+1}.
— si un des calculs a permis d'effacer tout le contenu du ruban 2 et que la tête sur le ruban 1 soit positionnée sur un ♯, l'entrée est acceptée et l'instance du problème a une solution.

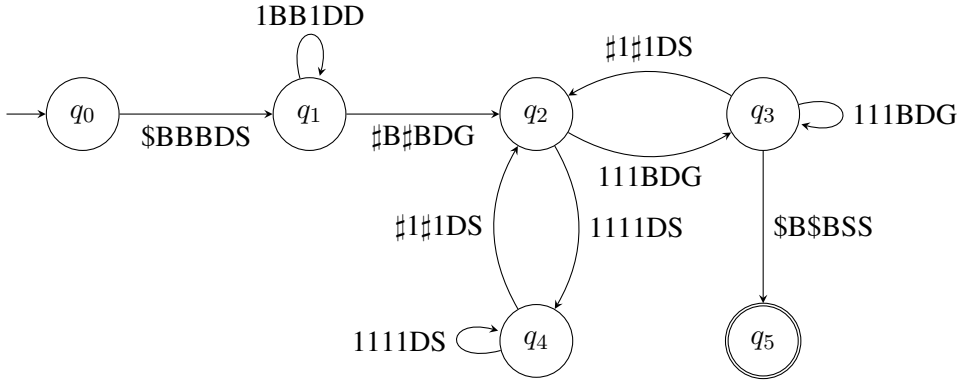

FIGURE 1 – Machine de Turing non déterministe à deux bandes pour SSP

La figure 2 donne l'arbre des calculs à partir de l'état q_1. Observons qu'il existe un calcul qui accepte l'entrée. La machine de Turing non déterministe répond donc positivement à la question "existe-t-il un sous-ensemble de $\{3, 4, 7\}$ dont la somme vaut 11 ?".

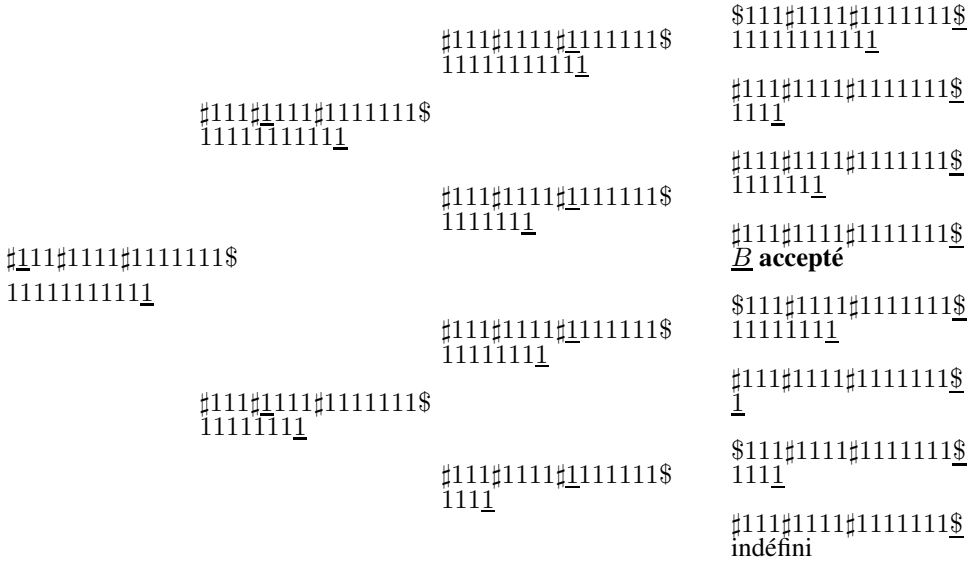

FIGURE 2 – Arbre des calculs de la NDTM pour $E = \{3, 4, 7\}$ et $C = 11$.

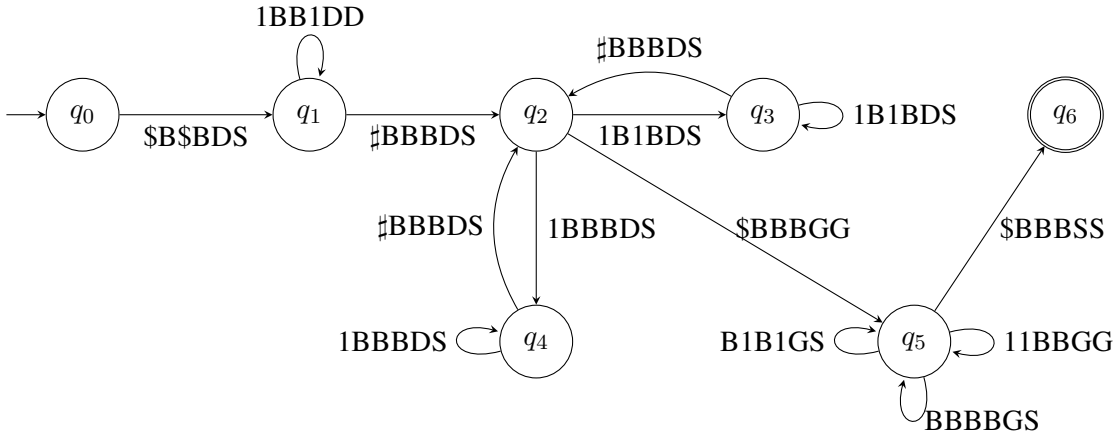

FIGURE 3 – Machine de Turing non déterministe à deux bandes pour SSP

Si les données ne sont pas données en unaire, alors la machine est un peu plus compliqué, car il faut faire des additions binaires pour chaque nombre choisi.
Comme on a remarqué, on peut construire d'abord la "prétendue solution" et ensuite vérifier, comme dans la figure 3. Dans ce cas on fait un premier parcours des données où on recopie C vers la deuxième bande et ensuite on choisit de manière non-déterministe les nombres à utiliser. Lors du deuxième passage on compare et on efface les bandes.

2. Pour montrer que le problème **3DM** est dans NP nous avons comme d'habitude deux choix possibles :

* décrire une machine de Turing non-déterministe qui accepte une instance de **3DM** en temps polynomial : Ainsi, il suffit de parcourir en temps linéaire la liste des triplets données, et choisir de manière non-déterministe ceux qui sont censés faire partie d'une solution. Nous obtenons ainsi une "prétendue solution" qu'on doit être capable de vérifier de manière déterministe en temps polynomiale (elle est détaillé dans la suite).
* montrer qu'une "prétendue solution" peut-être décrite en espace polynomiale et peut être vérifiée de manière déterministe en temps polynomiale. La "prétendue solution" consiste en l'énumération des triplets dont est composée la solution. Ainsi elle nécessite un espace linéaire en la taille des données. Pour la vérification, il suffit de vérifier qu'en effet chacun des $3q$ éléments figure dans un triplet et par exemple que le nombre de triplets est q. Ceci peut être fait en temps O(q^2) par exemple.

3. Nous ne donnons ici qu'une description brève des réductions et l'essentiel des justifications :

1. **Cheminham** $\propto$ **Circuitham** : On peut utiliser la même méthode qu'en cours pour la réduction **Chaîneham** $\propto$ **Cycleham**.

   En effet on peut rajouter un sommet ayant comme successeurs et comme prédécesseurs tous les autres sommets.

   Une autre version consiste à séparer le double rôle que joue le sommet rajouté. Ainsi on ajoute deux sommets, x ayant comme successeurs tous les "anciens" sommets et y ayant tous ces sommets comme prédécesseurs. Enfin on ajoute l'arc yx. La transformation se fait en temps polynomial. Si G admet un chemin hamiltonien, $s_1, s_2, \dots s_n$ alors $y, x, s_1, s_2, \dots s_n$ est un circuit hamiltonien dans G'. Et dans l'autre sens, si on a un circuit hamiltonien dans G', alors le successeur de y sur le circuit est nécessairement x. Donc le circuit est de la forme $y, x, s_1, s_2, \dots s_n$, ce qui permet de conclure que $s_1, s_2, \dots s_n$ est un chemin hamiltonien dans G.
2. **Cycleham** $\propto$ **Circuitham** : Il suffit de remplacer chaque arête par deux arcs (un dans chaque sens). Cette transformation se fait en temps linéaire. On remarque qu'il s'agit tous simplement de recopier les données à l'identique pour passer du graphe non-orienté sous forme de liste de voisins en un graphe orienté sous forme de listes de successeurs [1]. Ainsi, dans le graphe non-orienté il existe le cycle hamiltonien $s_1, s_2, \dots s_n$ si et seulement si dans dans le graphe orienté il existe le circuit hamiltonien $s_1, s_2, \dots s_n$ et v
3. **Chaîneham** $\propto$ **Cheminham** : comme le précédent.
4. **Cycleham** $\propto$ **Chaîneham** : On choisit un sommet x. On rajoute un sommet y ayant les même voisins que x. On rajoute un sommet x' ayant comme unique voisin x et un sommet y' ayant comme unique voisin y. Cette transformation est polynomiale.

   Si dans G on a un cycle hamiltonien $x, s_2, \dots s_n$, alors dans G' on a la chaîne hamiltonienne $x', x, s_2, \dots s_n, y, y'$. Et si dans G' on a une chaîne hamiltonienne, alors x' et y', des sommets n'ayant qu'un seul voisin sont ses extrémités. La chaîne est donc forcément de la forme $x', x, s_2, \dots s_n, y, y'$. Par la construction, s_n est voisin de x et donc $x, s_2, \dots s_n$, est un cycle hamiltonien dans G.
5. **Circuitham** $\propto$ **Cheminham** : même idée que le précédent, mais un peu plus simple, car on peut tenir compte des orientations. On choisit un sommet x. On rajoute un sommet y ayant les même successeurs que x. On rajoute un sommet x' ayant comme unique prédécesseur x. Le sommet y' n'est plus nécessaire, car y n'ayant pas de prédécesseurs est nécessairement l'extrémité initiale du chemin hamiltonien.
6. **Circuitham** $\propto$ **Cycleham** : On remplace chaque sommet x par deux sommets, l'un x^a "d'arrivée", l'autre ^{d} "de départ". Un arc xy devient une arête $x^d y^a$. De plus, pour éviter les éventuels changements de direction, le sommet x^a est relié au sommet x^d par la chaîne formée des deux arêtes $x^a x^m$ et $x^m x^d$ où x^m est un sommet "milieu" associé au sommet x. La transformation est polynomiale.

   Si dans G on a un circuit hamiltonien $s_1, s_2, \dots s_n$ alors dans G' on a le cycle hamiltonien $s_1^a, s_1^m, s_1^d, s_2^a, s_2^m, s_2^d, \dots s_n^a, s_n^m, s_n^d$.

   Supposons que G' admet un cycle hamiltonien $s_1, s_2, \dots s_3 n$. On peut remarquer qu'en fait le graphe que nous avons est un graphe triparti, avec les parties qui correspondent aux trois types de sommets. Nous ne connaissons pas les étiquettes des nœuds donc ne savons pas si un sommet est un arrivée, un milieu ou un départ. Cependant, on sait que les sommets de milieu sont à une distance 3 l'un de l'autre. Comme il y a n sommets milieu ils sont forcément chaque troisième sommet. Nous avons deux cas : soit chaque sommet milieu est précédé par un sommet arrivée et suivi par un sommet départ soit l'inverse. Dans le premier cas, dans G il existe un arc du sommet dont est issue ce sommet de milieu vers le suivant et donc on peut ainsi conclure que nous avons un circuit hamiltonien dans G. Dans le cas contraire, il suffit de parcourir le cycle hamiltonien de G dans l'autre sens, pour se retrouver dans le premier cas.
7. **Cheminham** $\propto$ **Chaîneham** : même réduction que la précédente.

Par ailleurs, nous pouvons remarquer, que plusieurs de nos réductions ne sont pas vraiment nécessaires, car la réduction se déduit par transitivité des réductions précédentes.

Un bon exercice conseillé est d'énoncer les même quatre problèmes pour graphes bipartis [2] et ensuite adapter les réductions pour prouver que ces problèmes sont aussi équivalents.

1. Ce qui justifie l'approche de C. Berge qui définit un graphe non-orienté comme un graphe orienté symétrique [3]

2. Un graphe est dit biparti si son ensemble de sommets peut être partitionné en deux sous-ensembles disjoints U et V tels que chaque arête (respectivement arc) ait une extrémité dans U et l'autre dans V.

4. Le problème **nombre composé** :

1. Rappelons tout d'abord que n est **composé** s'il existe deux entiers p et q tels que $n = p \times q$. Montrer que le problème **nombre composé** $\in$ NP revient à construire une machine de Turing non-déterministe qui
   — choisit de façon non déterministe un entier p ;
   — accepte si p divise n ;
   — rejette si p ne divise pas n.
   Tout est fait par l'arbre des calculs non déterministes. On rappelle que n est composé s'il existe un calcul acceptant dans l'arbre des calculs tout entier. Quelle est la complexité de cette machine de Turing ? Il faut évaluer la complexité de la division. En l'implémentant de façon simple au moyen de soustractions et de décalages, il est clair que l'algorithme ci-dessus est polynomial. Cependant, dans le cas où $k = 2$, il est possible d'améliorer sa complexité. Dans [1], on trouve les deux résultats suivants :

   Théorème 1 *$M(n) = \theta(D(n))$ où $M(n)$ (resp. $D(n)$) représente le temps de calcul du produit (resp. de la division) de deux nombres de n bits.*

   Corollaire 1 *La division d'un entier de $2n$ bits par un entier de n bits peut être réalisée en temps $\Omega(n \log n \log\log n)$ en utilisant l'algorithme de Schönhage-Strassen.*

   En utilisant le corollaire, on en déduit que notre algorithme non-déterministe travaille en temps sub-quadratique et donc que notre problème est dans la classe du temps polynomial pour les modèles non-déterministes.

2. Si nous travaillons en unaire, ce problème devient polynomial en temps pour une machine de Turing déterministe. En effet, une telle machine de Turing à deux rubans doit :
   — engendrer l'entier $p = 2$ en unaire sur le ruban 2 ;
   — accepter si p divise n, i.e. si le reste de la division de n par p est nul ;
   — incrémenter p si p ne divise pas n ($2 \leq p \leq n-1$) et retourner à l'étape précédente.
   La première et la dernière étape sont simples à réaliser : il suffit d'ajouter un bâtonnet sur le ruban 2 ce qui prend un temps constant. La seconde étape mérite plus de détails. Pour réaliser la deuxième étape, il suffit de transformer en a autant de bâtonnets du ruban 1 que le nombre de bâtonnets du ruban 2 tant que le nombre de bâtonnets du ruban 1 est supérieur ou égal au nombre de bâtonnets du ruban 2.
   L'exemple suivant illustre le calcul du reste de la division de 4 par 2 en unaire.

	$\underline{1}$	1	1	1	
	$\underline{1}$	1			
	a	$\underline{1}$	1	1	
	1	$\underline{1}$			
	a	a	$\underline{1}$	1	
	1	1	$\underline{B}$		
	a	a	$\underline{1}$	1	
	1	$\underline{1}$			
	a	a	a	$\underline{1}$	
	$\underline{1}$	1			
	a	a	a	a	$\underline{B}$
$\underline{B}$	1	1			

   La complexité de cette division est linéaire sur la taille de n qui est ici également n. Pour notre algorithme déterministe, comme il faut effectuer au plus $n-2$ divisions, sa complexité est en $O(n^2)$.

3. Il est possible d'améliorer encore cet algorithme en utilisant le crible d'Eratosthène, pour lequel n est premier (non composé) s'il n'est ni un multiple de 2, ni un multiple de 3, ni d'aucun entier impair inférieur à $\sqrt{n}$. Il suffit donc d'engendrer tous les nombres impairs inférieurs à $\sqrt{n}$ et de vérifier que n est impair. Pour ce faire, on adapte l'algorithme vu dans un TD précédent qui vérifie qu'un entier en unaire est un carré parfait de complexité linéaire en ajoutant un ruban supplémentaire. La complexité de notre algorithme devient donc $O(n^{\frac{3}{2}})$.

Remarque : En 2004 ([2]), trois chercheurs indiens, Manindra Agrawal, Neeraj Kayal et Nitin Saxena ont prouvé que le problème de la primalité est dans P. Ainsi, dans tous les cas, notre problème s'avère aussi d'être en P.

5.

1. Montrons que **2-SAT**$\propto$ **X2-SAT**. Soit donc ϕ une instance positive de **2-SAT** ϕ est sous la forme $\phi = \bigwedge_{i=1}^{n} C_i$ avec $C_i = (l_{i,1} \vee l_{i,2})$ ou $C_i = l_{i,1}$. on appellera un **orphelin** une clause de la forme $C_i = l_i$. On applique la transformation suivante : on remplace chaque littéral d'une clause orpheline par la valeur 1 dans chacune des clauses où l_i apparaît. On simplifie ensuite ϕ en utilisant les règles suivantes :

$$x \vee 1 \equiv 1$$
$$x \vee 0 \equiv x$$
$$x \wedge 1 = x$$

Deux cas peuvent alors se produire :
— soit ϕ est sous la forme **X2-SAT** ;
— soit ϕ contient encore des clauses orphelines auquel cas on itère le procédé.
Dans le pire des cas, on n'obtient que des clauses orphelines à chaque étape et à la fin ϕ est entièrement évaluée. La complexité de ce procédé est obtenue dans le pire des cas et est de $O(n^2)$ si on a n clauses au départ. En effet, chaque substitution/simplification requiert un temps linéaire et il nous faut itérer ce procédé au plus n fois. Observons que $\phi \in$**2-SAT** si et seulement si $\phi \in$ **X2-SAT** par construction.
Exemple : Soit $\phi = (a \vee b) \wedge \neg c \wedge (c \vee \neg a) \wedge b$. Alors,

$$(a \vee b) \wedge \neg c \wedge (c \vee \neg a) \wedge b \equiv (a \vee b) \wedge 1 \wedge \neg a \wedge b \equiv (a \vee b) \wedge \neg a \wedge b \equiv (0 \vee b) \wedge 1 \wedge b \equiv b \wedge b \equiv 1$$

2. Montrons que **k-SAT**$\propto$**Xk-SAT**. Il suffit de rajouter des variables à chaque clause "déficitaire" selon le principe suivant : supposons que $C = x \vee y$. On ajoute une nouvelle variable z comme suit $(C \vee z) \wedge (C \vee \neg z)$. Si $\mathbf{V}(z) = 1$, l'ensemble des deux clauses se simplifie en $1 \wedge C \equiv C$ (Il en est de même si $\mathbf{V}(z) = 0$). On itère ce procédé jusqu'à "remplissage" exacte des clauses. Il est clair que toute valuation satisfaisant ϕ de **k-SAT** satisfait également ϕ' de **Xk-SAT**. Le procédé est clairement polynomial.
3. Le même procédé sert également à montrer que **k-SAT**$\propto$ **(k+1)-SAT**.
4. L'indication donnée proposait de transformer une clause $C_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,k+1}$ en la conjonction de deux clauses $C'_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,t} \vee y_i$ et $C''_i = \neg y_i \vee l_{i,t+1} \vee l_{i,t+2} \vee \ldots \vee l_{i,k+1}$, où y_i est une nouvelle variable qui n'apparait nulle-part ailleurs. Cette transformation est utile, car la nouvelle variable introduite ne peut pas être vraie dans les deux clauses, ce qui implique que une des deux clauses soit satisfaite autrement. Ainsi la clause de départ était satisfaite. Par ailleurs, si la grande clause est satisfaite, alors une des deux nouvelles clauses l'est aussi et l'autre peut être satisfait par le choix convenable de la valeur de vérité du nouveau variable. En faisant cela on obtient deux clauses, de degrés $t+1$ et $k+2-t$ respectivement. Le but étant de réduire le degré, il faut que $t+1 < k+1$ et $k+2-t < k+1$. Comme les degrés sont des nombres entiers, cela implique $t+1 \leq k$ et $k+2-t \leq k$. En sommant les deux inégalités on obtient $k \geq 3$, ce qui limite donc l'utilisation de cette technique et explique pourquoi on ne peut pas réduire les problème de satisfiabilité à des degrés inférieurs à 3.
5. Les preuves ci-dessus permettent de conclure la NP-difficulté de **k-SAT** et **Xk-SAT** pour $k \geq 3$.
6. Montrons que **X2-SAT**$\in$P. Nous nous inspirons de la démonstration de [5]. Soit ϕ une instance de **X2-SAT**. On construit le graphe $G(\phi)$ défini par : les sommets de $G(\phi)$ sont les variables de ϕ et leur négation ; il y a un arc (α, β) si et seulement si la clause $(\neg\alpha \vee \beta)$ ou la clause $(\beta \vee \neg\alpha)$ occure dans ϕ. Intuitivement, ces arcs expriment les implications logiques de ϕ. En conséquence, ϕ a la symétrie suivante : si (α, β) est un arc de $G(\phi)$, alors $(\neg\beta, \neg\alpha)$ est également un arc de ϕ. Les chemins de $G(\phi)$ sont aussi des implications logiques (par la transitivité de l'implication). Montrons maintenant que ϕ est une antilogie si et seulement si il existe une variable x telle qu'il existe un chemin de x vers $\neg x$ et un chemin de $\neg x$ vers x.
   Par contradiction, supposons que de tels chemins existent et que ϕ est satisfiable par la valuation $\mathbf{V}$. Supposons en outre que $\mathbf{V}(x) = 1$ (la valuation contraire conduirait à une preuve similaire). Comme il existe un chemin de x vers $\neg x$ et que $\mathbf{V}(x) = 1$ et $\mathbf{V}(\neg x) = 0$, il existe un arc (α, β) tel que $\mathbf{V}(\alpha) = 1$ et $\mathbf{V}(\beta) = 0$. Cependant, comme (α, β) est un arc de $G(\phi)$, $(\neg\alpha \vee \beta)$ est une clause de ϕ. Cette clause n'est pas satisfaite par la valuation, une contradiction.
   Réciproquement, supposons qu'il n'existe pas de variable avec de tels chemins dans $G(\phi)$. Nous construisons une valuation telle qu'il n'existe pas de sommets ayant un chemin allant de vrai à faux. On itère le procédé suivant :

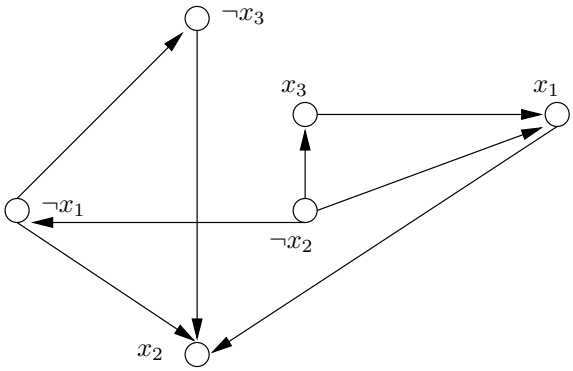

FIGURE 4 – Graphe associé à $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$.

on choisit un sommet α dont la valuation n'a pas encore été fixée et tel qu'il n'existe pas de chemin de α vers $\neg\alpha$. On considère ensuite l'ensemble des sommets que l'on peut atteindre depuis α dans $G(\phi)$ et on leur associe la valeur vrai. On associe également la valeur faux à la négation de ces sommets (ces négations correspondent aux sommets depuis lesquels $\neg\alpha$ peut être atteint). Ce procédé est bien défini car s'il existait simultanément des chemins de α vers β et $\neg\beta$, alors il y aurait des chemins de $\neg\alpha$ vers chacun d'entre eux (par la symétrie de $G(\phi)$) et donc un chemin de α vers $\neg\alpha$ une contradiction. De plus, s'il existait un chemin de α vers un sommet déjà affecté à faux dans une étape précédente, α serait un prédécesseur de ce sommet et aurait donc déjà dû être affecté à faux à cet instant.

On répète cette étape jusqu'à ce que tous les sommets aient une valuation. Comme nous avions supposé qu'il n'y avait pas de chemin entre un quelconque x et $\neg x$ ni de chemin entre un quelconque $\neg x$ et x, tous les sommets reçoivent une valeur de vérité. Comme, de plus, les étapes sont telles que lorsqu'un sommet est affecté à vrai, tous ses successeurs sont affectés à vrai, et de même pour faux, il ne peut y avoir d'arc vrai vers faux. La valuation satisfait donc ϕ.

Ce procédé s'effectue bien en temps polynomial. Il suffit de calculer la fermeture transitive de $G(\phi)$ qui peut être calculée en $O(V(G(\phi)) \times E(G(\phi)))$ (cf cours d'algorithmique de troisième année).

Références

[1] A. V. Aho, J. E. Hopcroft & J. D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley, 1974.

[2] M. Agrawal, N. Kayal & N. Saxena. *PRIMES is in P*. Annals of Mathematics 160(2) : 781-793, 2004.

[3] C. Berge. *Graphes et Hypergraphes*. Dunod, 1969.

[4] M. R. Garey & D. S. Johnson. *Computers and intractability*. Freeman, 1979.

[5] C. H. Papadimitriou. *Computational complexity*. Addison Wesley, 1994.