Complexité - TD 04

Benjamin Bordais

10 Décembre 2020

Exercise 1 Mon tout premier problème PSPACE-complet

Montrer que le problème suivant est PSPACE-complet :
- Entrée : le code d'une machine de Turing M, un mot w, une quantité d'espace t écris en unaire
- Sortie : M accepte w en espace t

Solution :

Prouvons d'abord que ce problème est dans PSPACE. Considérons une instance (M, w, t) du problème. Il faut remarquer que l'on ne peut pas se contenter de simuler M sur w tout en s'assurant que l'espace pris ne dépasse pas t car alors on n'est pas garanti que l'algorithme termine. Pour cela, il faut rajouter un timeout qui dépendra de M, t et x. En effet, pour une machine M fonctionnant sur un alphabet Σ, sur un ensemble d'états Q, sur une entrée x de taille n, avec un espace pris inférieur à t, le nombre total de configurations possibles est inférieur à $f(n) = |Q| \cdot |\Sigma|^t \cdot t \cdot n$ (les deux derniers termes faisant référence aux têtes de lecture). L'algorithme consiste alors alors à simuler M sur x en s'assurant que l'espace pris n'excède pas t et le nombre d'étapes pris ne dépasse pas $f(n)$. A noter que $f(n)$ peut être représenté en espace polynomial, tout comme un état courant du calcul de M sur x.

Pour ce qui est de la PSPACE-dureté, si l'on considère un langage L dans PSPACE décidé en espace polynomial p par une machine de Turing M, on considère alors la réduction f telle que $f(x) = (M, x, p(|x|))$. On a alors $x \in L$ si et seulement si $f(x)$ est dans notre langage. De plus, f est constructible en espace logarithmique étant donné qu'il suffit juste d'écrire un polynome sur la bande de sortie (le polynome est fixé).

Exercise 2 Planification

On note $I = \{1, \dots, n\}$. Un problème de planification est donné par :
- n variables booléennes $\{x_i\}_{i \in I}$;
- m opérations où chaque opération est définie par une condition de la forme $\bigwedge_{i \in I'}(x_i = \alpha_i)$ avec $I' \subseteq I$ et une mise à jour de la forme $\{x_i \leftarrow \beta_i\}_{i \in I''}$ avec $I'' \subseteq I$.
  Voici un exemple d'opération : Si $x_1 = \mathtt{V} \wedge x_3 = \mathtt{F}$ Alors $x_1 \leftarrow \mathtt{F}; x_2 \leftarrow \mathtt{V}$
- Une configuration initiale s_{init} et une configuration finale s_{fin}.
- Une opération est applicable à une configuration si la condition de l'opération s'évalue à Vrai. Son application consiste à effectuer ses mises à jour pour obtenir la nouvelle configuration. Par exemple l'opération précédente est applicable à la configuration $(\mathtt{V}, \mathtt{F}, \mathtt{F})$ et conduit à la configuration $(\mathtt{F}, \mathtt{V}, \mathtt{F})$.

Le problème consiste à déterminer s'il existe une suite d'applications des opérations (avec éventuellement plusieurs applications d'une même opération) qui conduise de la configuration initiale à la configuration finale.

1. Montrez que le problème de planification est dans PSPACE.
2. Montrez que le problème de planification est PSPACE-dur. Pour cela, on montrera une réduction LOGSPACE du problème ci-dessous vers le problème de planification :
   — Entrée : Une machine $\mathcal{M}$, un espace t (en unaire) et un mot w ;
   — Sortie : $\mathcal{M}$ accepte w en espace t.
   (Hint : pour simplifier, on supposera que $\mathcal{M}$ a une seule bande de travail, et l'efface à l'issue du calcul (une fois dans l'état acceptant ou rejetant), ainsi que replace les têtes de lecture au début des bandes correspondantes.)

Solution :

1. On montre que le problème est dans NPSPACE (et on conclut à l'aide du théorème de Savitch).

```
Planification :
        x ← s_init
        c ← 0
        while x ≠ s_fin ∧ c ≤ 2^n do:
                // Choix non deterministe
                Choisir une operations op parmi m;
                Mettre a jour x avec op;
                c ← c + 1;
        end
        if c ≤ 2^n accept
        else reject
```

   Cet algorithme utilise bien un espace polynomial vu que x est de taille n (et à valeur dans $\{0, 1\}$) et l'écriture binaire de c prend au plus n bits.
2. Considérons une instance (M, w, t) du problème considéré, on construit alors une instance $\mathcal{P}$ de problème de planification de la manière suivante (pour $n = |w|$) :
   — n variables w_i qui codent l'entrée w. Ces variables ne seont pas modifiées ;
   — n autres variables pw_i qui codent la position de la tête de lecture sur la bande d'entrée, avec si la tête de lecture est sur w_i, alors il faut $pw_i = \texttt{V}$ et $pw_j = \texttt{F}$ pour tout $j \neq i$;
   — La bande de travail étant de taille au plus t, on peut la coder avec t variables x_i codant sa valeur et t variables px_i codant la position de la tête de lecture sur la bande de travail et il faut, comme pour la bande d'entrée, qu'à tout instant une seule variable px_i soit à V ;
   — Une variable q_i pour chaque état de la machine M (avec, là aussi, une seule variable q_i à V)
   — Les opérations de $\mathcal{P}$ correspondent exactement aux transitions de la machine (pour chaque lettre de la bande de travail)
   — s_{init} correspond à la configuration initiale de la machine
   — s_{fin} correspond à l'état final de la machine, dans l'état acceptant, la bande de travail effacé et les têtes de lecture réinitialisées.

   La construction considérée s'effectue bien en espace logarithmique et on a que M accepte w en espace t si et seulement si $\mathcal{P}$ a un run acceptant.

Exercise 3 Vacuité d'un Automate Déterministe Concurrent

Un automate déterministe concurrent $\mathcal{A}$ est donné par un ensemble d'automates déterministes $\{\mathcal{A}_i\}_{i \leq n}$ appelés composants. Un état de l'automate déterministe concurrent est un

tuple $(s_1, \ldots, s_n)$ composé d'un état par composant. Lorsqu'une lettre a est lue, les automates $\mathcal{A}_i$ qui ont une transition issue de s_i étiquetée par la lettre effectuent simultanément leur transition tandis que les autres conservent leur état. Pour qu'une lettre puisse être lue, au moins un automate doit effectuer une transition. Un mot est reconnu s'il conduit à un tuple d'états terminaux. Le problème de la vacuité consiste à savoir s'il existe au moins un mot accepté par l'automate.

1. Montrez que le problème de la vacuité des automates déterministes concurrents est dans PSPACE.
2. Montrez que le problème de la vacuité est PSPACE-difficile depuis la problème de planification.

Solution :

1. Comme pour la question précédente, on montre que le problème est dans NPSPACE (et on conclut à l'aide du théorème de Savitch). On dénote par m le nombre d'états maximum d'un automate déterministe parmi ceux composants l'automate concurrent.

```
Vacuite Automate Concurrennt :
        for i = 1 to n do
            s[i] ← s^i_init
        end
        c ← 0
        while s ≠ [s_fin] ∧ c ≤ m^n do
            // Choix non deterministe
            Choisir une lettre a ∈ Σ
            Mettre a jour s en simulant chaque automate sur a
            c ← c + 1
        end
        if c ≤ m^n accept
        else reject
```

   L'espace pris par cet algorithme est bien polynomial étant donné que s est un vecteur de taille n contenant les états de chaque automate, l'écriture binaire de c prend $n \cdot \log m$ bits et est donc polynomial en l'entrée.

2. On effectue une réduction depuis le problème de planification. Considérons une instance $\mathcal{P}$ du problème de planification, on construit alors un automate concurrent $\mathcal{A}$ tel que :
   — L'alphabet correspond à l'ensemble des opérations de $\mathcal{P}$;
   — Pour chaque variable x_i, on construit un automate à trois états $\{\mathtt{V}, \mathtt{F}, Fail\}$ dont l'état initial correspond à l'état spécifié dans s_{init}, l'état final correspond à l'état s_{fin}. Pour chaque lettre *op* de l'alphabet, pour chaque variable x_i, si la condition de *op* sur x_i est invalidé par sa valeur, l'automate correspondant passe dans l'état $Fail$, sinon il est mis à jour selon l'effet de *op* (potentiellement, rien ne se passe). De plus, un automate ne peut pas sortir de l'état $Fail$ (c'est un état puits).

   De cette manière, on peut passer de s_{init} à s_f dans $\mathcal{P}$ si et seulement si $\mathcal{A}$ accepte au moins un mot.

Exercise 4 Géographie

Le jeu Géographie se joue de la manière suivante :

— Le jeu commence avec un nom donné de ville, par exemple *Cachan* ;
— le premier joueur donne le nom d'une ville dont la première lettre coïncide avec la dernière lettre de la ville précédente, par exemple *Nice* ;
— le deuxième joueur donne un autre nom de ville, commençant également par la dernière lettre de la vile précédente, par exemple *Évry* ;
— le premier joueur joue à nouveau et ainsi de suite avec la restriction qu'aucun joueur ne peut donner le nom d'une ville déjà vu dans le jeu ;
— le perdant est le premier joueur qui ne peut trouver de nom de ville pour continuer.

Ce jeu peut être décrit à l'aide d'un graphe orienté dont les nœuds représente les villes et où une arête (X, Y) signifie que la dernière lettre de X est la première lettre de Y. Ce graphe a également un nœud distingué qui correspond à la ville initiale. Chaque joueur choisit un nœud du graphe successeur du nœud précédent, le joueur 1 choisit en premier et ensuite les deux joueurs alternent. Un joueur gagne si, à un moment donné, l'autre joueur ne peut plus choisir de nœud qui n'a pas déjà été visité.

Géography Généralisée (i.e. GG) correspond au problème suivant :
— Entrée : un graphe orienté G, et un nœud initial s.
— Sortie : le joueur 1 a une stratégie gagnante pour GG sur un graphe G depuis s

1. Montrer que GG est dans PSPACE.
2. Prouver que GG est PSPACE-dur via une réduction depuis QBF.

Solution :

1. On définit la fonction récursive win : $(G, s, F) \mapsto$ "True ssi le joueur 1 a une stratégie gagnante sur le graphe $G = (V, E)$, en partant de $s \in V$, pour une version du jeu où il est interdit de jouer un nœud dans $F \subseteq V$". Cela revient à dire que : "le premier joueur a une stratégie gagnante sur la restriction $G_{|V \setminus F}$." On peut remarquer que dans la spécification de win, on requiert que $s \notin F$.

   Il s'agit à présent d'une procédure récursive :

   $$\mathsf{win}(G, s, F) = \exists (s, t) \in E \text{ s.t. } t \notin F' \wedge \mathsf{win}(G, t, F') = \mathsf{False} \text{ for } F' = F \cup \{s\}.$$

   Un algorithme implémentant cette procédure explore tous les successeurs possibles et s'appelle récursivement sur ceux-ci. Elle termine puisque chaque appel a un F plus grand et que dès que F est égal à l'ensemble tout entier V, il s'arrête. Ainsi, il peut y avoir au plus $|V|$ appels imbriqués et le programme va utiliser une pile avec un nombre linéaire de niveau, chaque niveau étant de taille linaire (puisque que l'on stocke F et s). Cela correspond bien à un espace polynomial.

2. On construit une réduction fonctionnant en espace logarithmique tr à partir d'une instance de QBF vers une instance de GG telle que $\phi \in \mathsf{QBF} \Leftrightarrow tr(\phi) \in \mathsf{GG}$. Considérons une formule QBF telle que $\phi = Q_1 x_1 \cdot Q_2 x_2 \cdots Q_n x_n \cdot S$ où S est une formule propositionnelle en CNF dont chaque variable est dans $\{x_1, \dots x_n\}$ et $Q_i \in \{\exists, \forall\}$ pour tout $1 \leq i \leq n$. D'abord, on traduit ϕ en une formule équivalente tel que l'on ait une stricte alternance entre les quantificateurs (cela peur requérir de rajouter des variables qui n'apparaissent pas dans ϕ). On obtient $\phi' = \exists y_1 \cdot \forall y_2 \cdots \exists y_{2k-1} \cdots \forall y_{2k} \cdot S$ pour un $k \leq n$. Soit $S = \wedge_{1 \leq j \leq m} C_j$ avec $C_j = \vee_{1 \leq i \leq a_j} l_{i,j}$ où $l_{i,j}$ est littéral égal à y_l ou $\neg y_l$ pour un $1 \leq l \leq 2k$.

   On définit formellement $tr(\phi)$ comme le graphe $G_\phi = (V, E)$ où $V = \{s_i, y_i, \neg y_i, s'_i \mid 1 \leq i \leq 2k\} \cup \{C_j \mid 1 \leq j \leq m\} \cup \{l^c_{i,j} \mid 1 \leq j \leq m,\ 1 \leq i \leq 3\}$ et $E =$

$\{(s_i, y_i), (s_i, \neg y_i), (y_i, s'_i), (\neg y_i, s'_i) \mid 1 \leq i \leq 2k\} \cup \{(s'_i, s_{i+1}) \mid 1 \leq i \leq 2k-1\} \cup \{(s'_{2k}, C_j) \mid 1 \leq j \leq m\} \cup \{(C_j, l^c_{i,j}) \mid 1 \leq j \leq m,\ 1 \leq i \leq a_j\} \cup \{(l^c_{i,j}, l_{i,j}) \mid 1 \leq j \leq m,\ 1 \leq i \leq a_j\}$. Le sommet initial du graphe est s_1. On peut remarquer que, pour tout i le joueur 1 joue sur les sommet s_i et s'_i ssi i est impair. C'est également le tour du joueur 1 au smmet C_j pour tout $1 \leq j \leq m$.

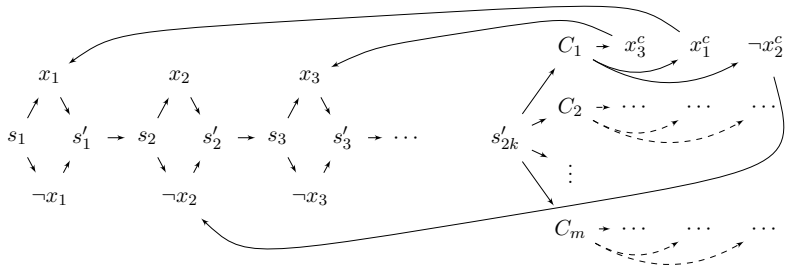

FIGURE 1: The graph G_ϕ for $\phi' = \exists x_1 \forall x_2 \cdots \forall x_{2k} \Big((x_3 \vee x_1 \vee \neg x_2) \wedge C_2 \wedge \cdots \wedge C_m \Big)$.

D'abord, notons que ce graphe peut être construit en espace logarithmique. En effet, on a seulement besoin d'un nombre fixé de pointeurs qui parcourt les variables et les clauses de la formule.

Prouvons à présent que $\phi \in \mathsf{QBF}$ si et seulement si le premier joueur a une stratégie gagnante dans le jeu de géographie généralisé G_ϕ. On définit dans un premier temps le chemin correspondant à la valuation pour les $0 \leq l \leq 2k$ premières variables : $p(\emptyset) = \epsilon$ et $p(\nu' = \nu \cdot \{y_l \to \_\}) = p(\nu) \cdot s_l \cdot a \cdot s'_l$ avec a le sommet/littéral égal soit à y_l ou $\neg y_l$ vérifiant $\nu' \models a$. Alors, pour une valuation ν des $1 \leq l \leq 2k$ premières variables, quand on dit que le jeu commence après que le chemin $p(\nu)$ est vu, cela signifie que les sommets dans $p(\nu)$ sont interdits, le sommet initial est s_{l+1} et c'est au joueur 1 de jouer ssi $l+1$ est impair. Pour le cas où $l = 2k$, alors le jeu commence dans un sommet c_j (selon ce qu'a joué le joueur 2 en s'_{2k}) avec le joueur 1 au trait.
On montre par induction la propriété suvante sur $1 \leq i \leq 2k+1$, $\mathcal{P}(i)$: "pour toute valuation ν des variables $y_1, \ldots, y_{i-1}$, on a $\nu \models Q_i y_i \cdots \forall y_{2k} \cdot S$ ssi le joueur 1 gagne dans le jeu qui commence après que $p(\nu)$ est vu".
Prouvons tout d'abord $\mathcal{P}(2k+1)$. Considérons une valuation ν des variables. Supposons que ν satisfait S et que le jeu commence en c_j pour un $1 \leq j \leq m$ après que $p(\nu)$ est vu. Prouvons qu'alors le joueur 1 gagne. Comme $\nu \models S$, il existe un littéral $l_{i,j}$ tel que $\nu \models l_{i,j}$ pour un $1 \leq i \leq a_j$. Supposons que le joueur 1 choisisse $l^c_{i,j}$ comme prochain sommet. Alors, le seul successeur possible est $l_{i,j}$ et par définition de $p(\nu)$, comme $\nu \models l_{i,j}$, il est interdit dans le graphe. Ainsi, le joueur 1 gagne. Supposons à présent que le joueur 1 gagne depuis tous les sommets $\{c_j \mid 1 \leq j \leq m\}$ après que $p(\nu)$ est vu. Prouvons que $\nu \models S$. Considérons une clause C_j pour un $1 \leq j \leq m$ et le choix du joueur 1 au sommet c_j. Le sommet $l^c_{i,j}$ est atteint pour un $1 \leq i \leq a_j$. Comme le joueur 1 gagne, il vient que le sommet $l_{i,j}$ est interdit, sinon le joueur 2 l'aurait choisi comme prochain sommet et le joueur 1 perd vu que, dans tous les cas, le successeur de $l_{i,j}$, c'est-à-dire s'_j, est interdit. Par définition de $p(\nu)$, le fait que le sommet $l_{i,j}$ est onterdit signifie que la valuation ν vérifie $\nu \models l_{i,j}$. Il vient que $\nu \models C_j$. Comme c'est est vérifié pour tout $1 \leq j \leq 2k$, on a $\nu \models S$.
Prouvons à présent que $\mathcal{P}(i+1) \Rightarrow \mathcal{P}(i)$ pour tout $1 \leq i \leq 2k$. Supposons que $\mathcal{P}(i+1)$ soit vérifié pour un i. On traite la cas où i est impair, l'autre cas étant analogue. Considérons une valuations ν des variables $y_1, \ldots, y_{i-1}$ et la formule $\exists y_i \cdots \forall y_{2k} \cdot S$. C'est au joueur 1 de jouer au sommet s_i. Supposons que le joueur

1 gagne après que $p(\nu)$ est vu. Considérons alors son choix de prochain sommet $l_i = (\neg)y_i$. Posons la valuation $\nu' = \nu \cdot \{y_i \to \_\}$ telle que $\nu' \models l_i$ (remarquons que cela est bien défini vu que la valuation ν ne traite pas la variable y_i). Comme le joueur 1 gagne après que $p(\nu)$ est vu, il gagne aussi après que $p(\nu')$ est vu comme $p(\nu')$ étend $p(\nu)$ en suivant son choix. Alors, par l'hypothèse d'induction $\mathcal{P}(i+1)$, $\nu' \models \forall y_{i+1} \cdots \forall y_{2k} \cdot S$. Il vient que $\nu \models \exists y_i \cdots \forall y_{2k} \cdot S$. Supposons à présent que $\nu \models \exists y_i \cdots \forall y_{2k} \cdot S$, prouvons que le joueur 1 gagne après que $p(\nu)$ est vu. Par définition de la sémantique de la satisfaction d'une formule quantifiée, $\nu \models \exists y_i \cdots \forall y_{2k} \cdot S$ signifie qu'il existe ν' étendant ν à la variable y_i telle que $\nu' \models \forall y_{i+1} \cdots \forall y_{2k} \cdot S$. Alors, par $\mathcal{P}(i+1)$, on a que le joueur 1 dans le jeu après que $p(\nu')$ est vu. Ainsi, au sommet s_i, le joueur 1 peut choisir le prochain sommet de telle manière d'imiter ce qui arrive dans le chemin $p(\nu')$. Donc, comme le joueur 1 gagne après que $p(\nu')$ est vu, le joueur 1 gagne aussi après que $p(\nu)$ est vu. On obtient que $\mathcal{P}(i)$ est vérifié.

On peut conclure que $\mathcal{P}(1)$ est vérifié, ce qui corresponds exactement à l'équivalence $\phi \in \mathsf{QBF} \Leftrightarrow tr(\phi) \in \mathsf{GG}$.