Théorie de la Complexité – Feuille de cours intégré n° 6
Complexité en espace

1 Complexité en espace

Nous avons vu précédemment les classes de complexité déterministes en temps : **DTIME**(f) où $f : \mathbb{N} \to \mathbb{N}$ est une fonction, et les classes importantes **P** et **EXPTIME**. Ces classes permettent d'évaluer le temps *suffisant* pour un algorithme résolvant un problème, et de différencier les problèmes pouvant se résoudre en temps polynomial de ceux nécessitant un temps exponentiel. Par exemple, être dans la classe **P** pour un problème de décision signifie qu'il peut être résolu par un algorithme donnant une réponse « oui » ou « non » en temps polynomial par rapport à la taille de l'entrée. Pour un problème donné, il est donc pertinent de trouver la *plus petite* classe de complexité en temps à laquelle le problème appartient. Le fait que certains problèmes, comme **SAT**, ne sont pas localisés précisément entre **P** et **EXPTIME** a conduit à définir des classes de complexité non-déterministes, comme **NP**. On a montré que :

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}.$$

La notion de complétude permet de caractériser des problèmes « difficiles » de la classe **NP** (vis-à-vis des réductions polynomiales). Si l'un quelconque de ces problèmes est dans **P**, alors les deux classes **P** et **NP** seraient les mêmes. Si au contraire ce n'est pas le cas, cela différencierait les classes **P** et **NP** et montrerait que le problème **SAT** n'est pas résoluble en temps déterministe polynomial.

Un autre paramètre important pour évaluer l'efficacité des algorithmes est leur complexité en ***espace***. On veut quantifier l'espace *suffisant* et l'espace *nécessaire* pour résoudre un problème. Comme dans le cas de la complexité en temps, on s'intéresse à la complexité d'un *problème*, qui est la meilleure complexité d'un algorithme le résolvant.

On va donc définir les classes de complexité en espace. Pour cela, il nous faut d'abord définir l'espace utilisé par une machine de Turing. Par comparaison avec la notion de complexité en temps, on va devoir utiliser des machines à *plusieurs bandes*. La raison est qu'on ne veut comptabiliser, dans l'espace consommé par une machine, ni l'espace pris par le mot d'entrée, ni celui pris par ce qui est écrit sur la bande de sortie. En effet, l'espace que l'on veut quantifier doit représenter la mémoire interne nécessaire au calcul. On utilise donc des machines ayant au moins *trois bandes* :

- Une unique *bande d'entrée* sur laquelle est écrit le mot d'entrée. La machine peut lire ce qui écrit sur cette bande mais ne peut pas la modifier (formellement, cela signifie que les transitions de la machine ne doivent pas modifier ce qui écrit sur cette bande).
- Une unique *bande de sortie* sur laquelle la machine va écrire le mot de sortie. Sur cette bande, chaque écriture déplace la tête à droite. En particulier, la machine peut écrire sur cette bande mais ne peut pas relire ce qu'elle a précédemment écrit (formellement, cela signifie que le seul symbole lu sur cette bande est $\square$, et donc que les transitions de la machine ne dépendent pas de ce qui est écrit sur cette bande).
- Un nombre arbitraire de *bandes de travail* sur lesquelles la machine peut lire et écrire.

Definition 1 - Espace utilisé par une machine de Turing déterministe à plusieurs bandes.

On considère un alphabet Σ et une machine de Turing déterministe M dont Σ est l'alphabet d'entrée. Étant donnée une fonction $f : \mathbb{N} \to \mathbb{N}$, on dit que l'espace utilisé par M est borné par f si pour toute entrée $x \in \Sigma^*$, l'exécution de M sur x **visite au plus** $f(|x|)$ cases **sur les bandes de travail**.

On remarque que contrairement à la définition de temps d'exécution d'une machine de Turing, la définition est pertinente pour des machines à plusieurs bandes seulement. Par ailleurs, il faut noter qu'au cours d'un calcul, ***une même case peut être utilisée plusieurs fois***. Cela fait que la complexité en espace peut être moindre que la complexité en temps. Il est naturel de généraliser la définition à des machines non-déterministes.

Definition 2 - Espace utilisé par une machine de Turing non-déterministe à plusieurs bandes.

On considère un alphabet Σ et une machine de Turing M dont Σ est l'alphabet d'entrée. Étant donnée une fonction $f : \mathbb{N} \to \mathbb{N}$, on dit que l'espace utilisé par M est borné par f si pour toute entrée $x \in \Sigma^*$, et pour **toute exécution de** M **sur** x, M **visite au plus** $f(|x|)$ cases **sur les bandes de travail**.

On peut maintenant définir les classes de complexité en espace. Ce sont des classes de **langages** (ou de *problèmes de décision*, ce qui est équivalent) : on se sert des machines de Turing de décision. Comme nous avons deux notions de machines, déterministes ou (possiblement) non-déterministes, cela conduit à définir deux sortes de classes.

Definition 3 - Classes de complexité en espace.

On considère une fonction $f : \mathbb{N} \to \mathbb{N}$. On note

- **DSPACE**(f) la classe de tous les langages L pour lesquels il existe une constante $h \in \mathbb{N}$ telle que L est décidé par une machine de Turing **déterministe** dont l'espace utilisé est borné par la fonction $n \mapsto h \times f(n)$.
- **NSPACE**(f) la classe de tous les langages L pour lesquels il existe une constante $h \in \mathbb{N}$ telle que L est décidé par une machine de Turing **possiblement non déterministe** dont l'espace utilisé est borné par la fonction $n \mapsto h \times f(n)$.

Exercice 1 – ☆ – Langages réguliers et complexité On considère l'alphabet $\Sigma = \{a, b\}$ et le langage $L = \Sigma^* a \Sigma^*$ (L est le langage des mots qui contiennent au moins une occurrence de la lettre « a »).

1. Quelle est la plus petite classe de complexité en temps que vous connaissez contenant L ?
2. Quelle est la plus petite classe de complexité en espace que vous connaissez contenant L ?
3. Plus généralement, répondre aux deux questions précédentes pour un langage régulier quelconque.

Exercice 2 – ☆ – Comparaison du temps et de l'espace On considère une fonction $f : \mathbb{N} \to \mathbb{N}$ quelconque. Comparer les classes suivantes :

- **DSPACE**(f) et **NSPACE**(f).
- **DSPACE**(f) et **DTIME**(f).
- **DSPACE**(f) et **DTIME**$(2^{O(f)}) = \bigcup_{k \in \mathbb{N}}$ **DTIME**$(2^{k \cdot f})$.

Comme pour la complexité en temps, nous allons principalement nous intéresser à un petit nombre de classes standard. Les plus importantes pour le cours sont les suivantes :

Definition 4 - Espace polynomial.

On note **PSPACE** et **NPSPACE** les classes suivantes :

$$\textbf{PSPACE} = \bigcup_{k\in\mathbb{N}} \textbf{DSPACE}(n^k) \qquad \text{et} \qquad \textbf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \textbf{NSPACE}(n^k).$$

Definition 5 - Espace logarithmique.

On note **L** et **NL** les classes suivantes :

$$\textbf{L} = \textbf{DSPACE}(\log(n)) \qquad \text{et} \qquad \textbf{NL} = \textbf{NSPACE}(\log(n)).$$

Commençons par considérer quelques problèmes dans les classes **L** et **NL**.

Exercice 3 – ☆ – Un langage dans L Montrer que le langage des mots sur l'alphabet $\{a, b\}$ contenant autant de a que de b est dans **L**.

Exercice 4 – ☆ – la variante la plus simple de SAT On considère le problème **1-SAT** :

ENTRÉE : φ une formule propositionnelle en 1-CNF (une conjonction de littéraux).
QUESTION : φ est-elle satisfaisable ?

Montrer que **1-SAT** est dans **L**.

Exercice 5 – ☆ – Accessibilité dans les graphes orientés On va considérer un problème standard de la théorie des graphes. Le problème **ACCESS** est défini comme suit :

ENTRÉE : Un graphe ***orienté*** $G = (V, E)$ et deux sommets $s, t \in V$.
QUESTION : Existe-t-il un chemin de s à t dans G ?

1. On fixe un graphe $G = (V, E)$. Donner une borne telle que pour tous sommets s et t dans V, si il existe un chemin de s à t, alors il en existe un dont la longueur est majorée par cette borne.
2. Montrer que **ACCESS** est dans **NL**.

Remarque 1. Nous verrons plus tard que **ACCESS** est un problème **NL**-complet (pour les *réductions en espace logarithmique*, une notion que nous définirons plus tard). Ce résultat est spécifique aux graphes **orientés**. Il a été montré que pour les graphes non-orientés, le problème est dans **L** (c'est un résultat difficile, prouvé en 2004 par Omer Reingold).

Exercice 6 – ❗☆ – Comparaisons entre classes de complexité Montrer qu'on a les inclusion suivantes entre les classes de complexité que nous avons introduites :

$$\textbf{L} \subseteq \textbf{NL} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subseteq \textbf{NPSPACE} \subseteq \textbf{EXPTIME} \subseteq \textbf{NEXPTIME}.$$

En ce qui concerne la complexité en temps, on rappelle que la question de l'égalité entre **P** et **NP** est ouverte. On conjecture que **P** est *strictement* incluse dans **NP** (c'est-à-dire que le non-déterminisme permet d'obtenir une meilleure complexité), mais on ne sait pas le montrer. Concernant l'espace polynomial, on va montrer que le non-déterminisme n'aide pas : en effet, le Théorème de Savitch dit que **PSPACE** = **NPSPACE**. On aura donc la situation représentée sur la Figure 1 ci-dessous (attention, elle est remplie de conjectures concernant le caractère strict ou non des inclusions).

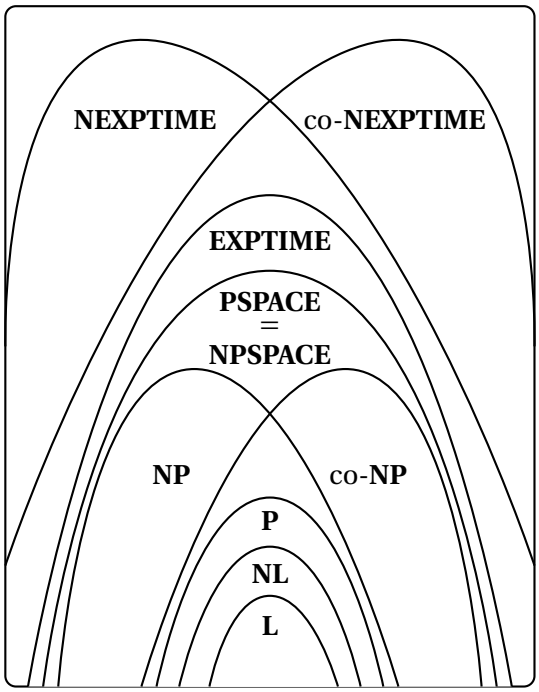


FIGURE 1 – Une comparaison des classes de complexité

Pour montrer que **PSPACE** = **NPSPACE**, on introduit un nouveau problème.

Definition 6 - Le problème QBF-SAT.

Une formule QBF (pour **Q**uantified **B**oolean **F**ormula) est une formule de la forme

$$Q_1 x_1 \, Q_2 x_2 \cdots Q_n x_n \varphi,$$

où les Q_i sont des quantificateurs (soit $\exists$, soit $\forall$), où les $x_1, \ldots, x_n$ sont des variables, et où φ est une formule propositionnelle sur les variables $x_1, \ldots, x_n$. Le quantificateur $\exists$ signifie « il existe ». Le quantificateur $\forall$ signifie « pour tout ».
Comme chacune des variables est quantifiée, une formule QBF est soit vraie, soit fausse (cela ne fait pas de sens de parler de satisfaisabilité pour une formule QBF, comme cela ne fait pas de sens de parler de véracité pour une formule SAT). Par exemple, la formule QBF $\varphi_1 = \forall x \exists y((x \wedge y) \vee (\neg x \wedge \neg y))$ signifie que pour tout x, il existe y rendant la formule $((x \wedge y) \vee (\neg x \wedge \neg y))$ vraie. La formule φ_1 est vraie : il suffit de choisir $y = x$. Par contre, la formule $\varphi_2 = \exists x \forall y((x \wedge y) \vee (\neg x \wedge \neg y))$ est fausse (tester les deux choix $x =$ Vrai et $x =$ Faux).

Le problème **QBF-SAT** est le suivant :

ENTRÉE	:	Une formule QBF.
QUESTION	:	φ est-elle vraie ?

Exercice 7 – ☆ – QBF-SAT et les classes NP et co-NP Montrer que **QBF-SAT** est à la fois **NP**-difficile et co-**NP**-difficile (pour les réductions polynomiales). Si on suppose vraie la conjecture affirmant que **NP** $\neq$ co-**NP**, que peut-on en déduire sur **QBF-SAT** ?

On va utiliser le problème **QBF-SAT** pour prouver que **PSPACE** = **NPSPACE**. Plus précisément, on va montrer simultanément que **QBF-SAT** est un problème **PSPACE**-complet et que **PSPACE** = **NPSPACE**.

Remarque 2 - Les langages PSPACE-complets. La notion de complétude est la même que celle que nous avons utilisée pour **NP**. C'est-à-dire qu'on langage L est **PSPACE**-complet (pour les réductions en temps polynomial) si et seulement si il satisfait les deux propriétés suivantes :

1. L est dans **PSPACE**.
2. L est **PSPACE**-difficile (pour les réductions en temps polynomial). C'est à dire que pour tout langage K dans **PSPACE**, il existe une réduction en temps polynomial de K vers L.

Exercice 8 – ❶☆ – Borne supérieure sur la complexité de QBF-SAT Montrer que le problème **QBF-SAT** est dans **PSPACE**.

Exercice 9 – ❶★★ – Borne inférieure sur la complexité de QBF-SAT On souhaite maintenant montrer que le problème **QBF-SAT** est **NPSPACE**-difficile. C'est-à-dire que nous devons montrer que tout problème de la classe **NPSPACE** se réduit à **QBF-SAT** par une réduction en temps polynomial.
Comme on ne connaît encore aucun problème **NPSPACE**-complet, on ne peut pas partir d'un tel problème et construire une réduction vers **QBF-SAT**. De façon analogue à la preuve de **NP**-complétude de **SAT**, nous allons devons faire une réduction « générique » : on doit partir d'un problème **arbitraire** de la classe **NPSPACE**, et montrer qu'il se réduit par réduction polynomiale à **QBF-SAT**.
On considère donc une machine de Turing M potentiellement **non-déterministe**, qui décide un langage en **espace polynomial**. La réduction doit construire, pour chaque mot d'entrée x, une formule QBF φ_x qui est vraie si et seulement si le mot x est accepté par M.
On considère le graphe orienté des configurations de M : ses sommets sont les configurations de M accessibles depuis la configuration initiale sur l'entrée x, et il y a un arc $c \longrightarrow c'$ si c' est une configuration accessible depuis la configuration c en appliquant **une** transition de M.

1. Quel est le nombre de bits nécessaires pour coder une configuration de M sur une entrée de taille n ?
2. On veut écrire une formule QBF $\psi_k(c, c')$, vraie si et seulement s'il y a un calcul de longueur au plus 2^k entre c et c' :
$$\underbrace{c \longrightarrow c_1 \longrightarrow \cdots \longrightarrow c'}_{\leqslant 2^k \text{ étapes de calcul}}$$
On veut également que ψ_k soit de taille polynomiale par rapport à k.
   a) Expliquez comment construire la formule ψ_0.
   b) Expliquez pourquoi la définition suivante exprime bien la propriété voulue, mais fournit des formules de taille *trop importante* :
$$\psi_k(c, c') = \exists c\check{}\,(\psi_{k-1}(c, c\check{}) \wedge \psi_{k-1}(c\check{}, c')).$$
   c) La construction précédente n'utilise que le quantificateur $\exists$. En utilisant une alternance entre un quantificateur $\exists$ et un quantificateur $\forall$, trouvez une autre formulation pour la formule ψ_k,
   - telle que $\psi_k(c, c')$ exprime qu'il existe un calcul de M de c à c' de longueur au plus 2^k,
   - de taille polynomiale en k.
3. Conclure des questions précédentes que QBF est **NPSPACE**-difficile.

On a montré que **QBF-SAT** est,

- dans la classe **PSPACE**,
- **NPSPACE**-difficile.

Ces deux propriétés impliquent les théorèmes suivants.

Théorème 3 — **Savitch**

Les classes de complexité en espace **PSPACE** et **NPSPACE** sont égales.

Théorème 4

Le problème **QBF-SAT** est **PSPACE**-complet.

Exercice 10 – ☆ – Conclusion de la preuve Prouver les deux théorèmes ci-dessus.

On va maintenant présenter deux autres problèmes **PSPACE**-complets. Le premier est simplement une variante simplifiée de **QBF-SAT**.

Exercice 11 – ❗★ – 3-QBF-SAT Le problème **3-QBF-SAT** est la restriction du problème QBF dans lequel la formule propositionnelle φ (celle qui est derrière le bloc de quantificateurs) est en forme 3-CNF. Montrer que le problème **3-QBF-SAT** est **PSPACE**-complet.

Exercice 12 – ❗★★ – 3-QBF-SAT Géographie généralisée On considère le jeu de géographie généralisée suivant. Le plateau est un graphe orienté $G = (V, E)$ qui contient un sommet $d \in V$ distingué (qu'on appelle le sommet de départ). De plus, il y a un unique jeton, qui, à tout moment, est posé sur un sommet du graphe. Au début du jeu, le jeton est posé sur le sommet distingué d.

Il y a deux joueurs que nous nommerons J_1 et J_2. Ils jouent à tour de rôle et J_1 commence. Un tour se déroule de la façon suivante : le joueur prend le jeton sur le graphe et le déplace sur un sommet adjacent à celui où le jeton se trouvait. Ce nouveau sommet **ne doit pas avoir été déjà visité lors des tours précédents**. La main passe ensuite à l'autre joueur et le tour suivant commence. Un des deux joueurs perd si il se retrouve bloqué, c'est-à-dire qu'il ne peut pas jouer car tous les sommets adjacents ont déjà été visités lors de tours précédents. De façon symétrique, l'un des deux joueurs gagne dès que l'autre a perdu. On considère le problème **GEO** suivant :

ENTRÉE	:	Un graphe $G = (V, E)$ et $d \in V$.
QUESTION	:	Le joueur J_1 a-t-il une stratégie gagnante pour le jeu de géographie généralisée sur ce plateau ?

Montrer que ce problème est **PSPACE**-complet.

2 La classe NL

On s'intéresse maintenant à la classe **NL**. Comme on l'a vu plus haut on sait que $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$. On commence par présenter un théorème que nous ne prouverons pas mais qui est important pour comprendre la situation.

Contrairement à la classe **PSPACE** qui est égale à **NPSPACE** par le théorème de Savitch, on conjecture généralement que $\mathbf{L} \neq \mathbf{NL}$ (mais, comme d'habitude, c'est seulement une conjecture). Cependant, nous disposons d'un résultat plus faible. On peut bien sûr définir la classe co-**NL** : elle contient tous les langages qui sont le complément d'un langage dans **NL**. Il se trouve que **NL** = co-**NL**.

Théorème 5 — **Immerman-Szelepcsényi**

Les classes de complexité en espace **NL** et co-**NL** sont égales.

On souhaite généraliser la notion de « problème complet » pour la classes **NL**. Naturellement, on va utiliser les réductions pour définir ce qu'est un « problème difficile » pour cette classe. Cependant, on ne peut plus utiliser les réductions polynomiales ici. On explique pourquoi dans l'exercice suivant.

Exercice 13 – ❗☆ – NL et les réductions en temps polynomial Montrer que tout problème non-trivial dans **P** est **NL**-difficile pour les réductions en temps polynomiale.

Pour définir une notion pertinente de **NL**-complétude, on va devoir utiliser des réductions plus fines. On considère des *réductions en espace logarithmique*.

Definition 7 - Réductions en espace logarithmique.

Considérons un alphabet Σ et deux langages $K, L \subseteq \Sigma^*$. Une réduction en espace logarithmique de K vers L est une fonction **calculable en espace logarithmique** $f : \Sigma^* \to \Sigma^*$ telle que :

$$\text{Pour tout } w \in \Sigma^*, \quad w \in K \text{ si et seulement si } f(w) \in L.$$

Avant de pouvoir utiliser de telles réductions, on doit prouver qu'elles sont transitives (ce qui est moins évident que pour les réductions polynomiales).

Exercice 14 – ❗★ – Transitivité des réductions en espace logarithmique On considère un alphabet Σ et deux fonctions $f : \Sigma^* \to \Sigma^*$ et $g : \Sigma^* \to \Sigma^*$ qui sont calculables en espace logarithmique.

1. Montrer que la fonction composée $f \circ g : x \mapsto f(g(x))$ est également calculable en espace logarithmique.
   Indice : Attention, il y a un piège. L'algorithme « naïf » qui consiste à d'abord calculer $g(x)$ à partir de x, puis $f(g(x))$ à partir de $g(x)$ utilise trop d'espace (potentiellement plus que logarithmique). Commencez par comprendre pourquoi.
2. En déduire que les réductions en espace logarithmique sont transitives : si il existe de telles réductions de K vers L et de L vers H, alors il en existe une de K vers H.

Definition 8 - Langages difficiles et complets pour les réductions en espace logarithmique.

Soit $\mathscr{C}$ une classe de complexité et L un langage.

- On dit que L est $\mathscr{C}$*-difficile* (pour les réductions en espace logarithmique) si et seulement si pour tout langage $K \in \mathscr{C}$, il existe une réduction en espace logarithmique de K vers L.
- On dit que L est $\mathscr{C}$*-complet* (pour les réductions en espace logarithmique) si et seulement si $L \in \mathscr{C}$ **et** L est $\mathscr{C}$-difficile.

Nous pouvons maintenant montrer que des problèmes sont **NL**-complets. On va commencer par le plus important de tous (que nous avons déjà introduit plus haut) : l'accessibilité dans les graphes orientés.

Exercice 15 – ❗★ – Transitivité des réductions en espace logarithmique On reprend le problème **ACCESS** de l'exercice 5 :

ENTRÉE : Un graphe ***orienté*** $G = (V, E)$ et deux sommets $s, t \in V$.
QUESTION : Existe-t-il un chemin de s à t dans G ?

Montrer que ce problème est **NL**-complet (pour les réductions en espace logarithmique).
Indice : On doit montrer que le problème est dans la classe **NL et** qu'il est **NL**-difficile. Pour cette dernière propriété, notez qu'on ne connaît pas encore de problème **NL**-difficile. Vous pouvez donc en déduire ce qu'il faut faire.

Remarque 6. Informellement, **ACCESS** est à la classe **NL** ce que **3-SAT** est à la classe **NP**. Lorsque l'on souhaite prouver qu'un problème est **NL**-complet, on utilise dans la plupart des cas une réduction de **ACCESS** vers ce problème.

Exercice 16 – ❗★ – Accessibilité paire et cycle impair On considère le problème **PACCESS** :

ENTRÉE	:	Un graphe ***orienté*** $G=(V,E)$ et deux sommets $s,t\in V$.
QUESTION	:	Existe-t-il un chemin de s à t de *longueur paire* dans G ?

et aussi le problème **ICYCLE** :

ENTRÉE	:	Un graphe orienté $G=(V,E)$.
QUESTION	:	Existe-t-il un cycle de *longueur impaire* dans G ?

1. Montrer que **ACCESS** et **PACCESS** se réduisent en espace logarithmique l'un à l'autre.
2. Montrer que **PACCESS** se réduit en espace logarithmique à **ICYCLE**.
3. Déduire que **PACCESS** et **ICYCLE** sont **NL**-complets.

Exercice 17 – ❗☆ – 2-Colorabilité On considère le problème **2-COL** vu dans la feuille précédente :

ENTRÉE	:	Un graphe G.
QUESTION	:	Existe-t-il un 2-coloriage de G ?

Montrer que **2-COL** est dans **NL**.

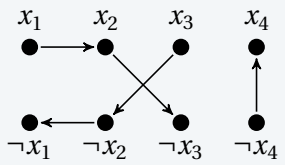
Remarque 7. En fait, **2-COL** est dans la classe **L**, mais cela résulte du résultat difficile dû à Omer Reingold concernant l'accessibilité dans les graphes non orientés.

Exercice 18 – ❗★ – 2-SAT On considère maintenant le problème **2-SAT** :

ENTRÉE	:	φ une formule propositionnelle en 2-CNF.
QUESTION	:	φ est-elle satisfaisable ?

On veut montrer que **2-SAT** est également **NL**-complet. En fait, on va considérer le problème complémentaire co-**2-SAT** (qui demande si la formule prise en entrée n'est *pas* satisfaisable) et montrer que co-**2-SAT** est **NL**-complet. Cette propriété implique de façon immédiate que **2-SAT** est co-**NL**-complet ce qui conclut la preuve puisque **NL** = co-**NL** par le théorème d'Immerman-Szelepcsényi.

1. Montrer que co-**2-SAT** est **NL**-difficile.
   Suggestion : on pourra réduire **ACCESS** à co-**2-SAT**.
2. On cherche maintenant à montrer que co-**2-SAT** est dans **NL**.
   Soit $\varphi=(\ell_{1,a}\vee\ell_{1,b})\wedge(\ell_{2,a}\vee\ell_{2,b})\wedge\ldots\wedge(\ell_{k,a}\vee\ell_{k,b})$ une formule d'entrée pour le problème co-**2-SAT**.
   Soit G_φ le graphe orienté ayant pour sommets les variables de φ et leurs négations et tels que, pour toute clause $(\ell_{i,a}\vee\ell_{i,b})$ de φ, G_φ possède les arrêtes $\neg\ell_{i,a}\longrightarrow\ell_{i,b}$ et $\neg\ell_{i,b}\longrightarrow\ell_{i,a}$ (on identifie $\neg\neg x$ avec x).
   Par exemple, si $\varphi=(\neg x_1\vee x_2)\wedge(\neg x_2\vee\neg x_3)\wedge(x_4\vee x_4)$, alors le graphe G_φ est



   On va maintenant montrer que la (non-)satisfiabilité de φ est équivalente à une propriété des circuits du graphe G_φ. Plus précisément, on va montrer que les deux propriétés suivantes sont équivalentes :

   1) φ n'est pas satisfaisable
   2) G_φ contient un circuit visitant une variable x et sa négation $\neg x$ (1)

a) Montrer que pour toute affectation θ des variables de φ, si θ satisfait à la fois φ et ℓ, et que (ℓ, ℓ') est une arrête dans G_φ, alors θ satisfait aussi ℓ'.

b) Montrer que si le graphe G_φ contient un circuit visitant une variable x et sa négation $\neg x$, alors φ n'est pas satisfaisable.

Suggestion : On pourra se servir de la question précédente.

c) Montrer que si φ n'est pas satisfaisable, alors le graphe G_φ contient un circuit visitant une variable x et sa négation $\neg x$. En déduire que (1) est vraie.

Suggestion : C'est le sens « difficile ». On pourra montrer la contraposée : si G_φ ne contient pas de circuit visitant une variable x et sa négation $\neg x$ alors φ est satisfaisable. Il faudra procéder par induction sur le nombre de variables.

d) Utiliser (1) pour donner un algorithme **NL** pour co-**2-SAT**. Conclure.