

Devoir à la maison

Informatique Fondamentale 1, ENS Rennes

Amaury Pouly

Elements de correction

1 Calculabilité

1.1 Langage $\mathcal{L}_1$

$\mathcal{L}_1$ est décidable car $\mathcal{L}_1 = \mathbf{a}^*$. En effet, pour tout $n \in \mathbb{N}$, on peut construire la machine $\mathcal{M}_n$ qui accepte uniquement les mots $\mathbf{a}, \mathbf{a}^2, \dots, \mathbf{a}^n$. Cette machine accepte clairement n mots donc $\mathbf{a}^n \in \mathcal{L}_1$.

1.2 Langage $\mathcal{L}_2$

$\mathcal{L}_2$ est semi-décidable car pour une paire $\langle [\mathcal{M}], \mathbf{a}^n \rangle$ on peut simuler en parallèle $\mathcal{M}$ sur toutes les entrées possibles et on s'arrête dès que $\mathcal{M}$ a accepté (au moins) n entrées. Plus formellement, on peut faire une boucle sur k et à chaque itération on simule k étapes de $\mathcal{M}$ sur toutes les entrées de taille au plus k . On stocke dans une liste les mots que $\mathcal{M}$ accepte au fur et à mesure de l'algorithme et on s'arrête dès que la liste est de longueur au moins k . Si la machine $\mathcal{M}$ accepte moins de n entrées alors le calcul ne termine jamais et $\langle [\mathcal{M}], \mathbf{a}^n \rangle$ n'est pas dans le langage ce qui est bien ce que l'on veut.

Toutefois $\mathcal{L}_2$ est indécidable par réduction depuis le langage ARRET qui est indécidable. En effet, soit $\langle [\mathcal{M}], w \rangle$ une instance. On construit la machine $\mathcal{A}_{\mathcal{M},w}$ qui sur l'entrée u simule $\mathcal{M}$ sur w et accepte si la simulation termine (sinon elle diverge).

Montrons que $\langle [\mathcal{M}], w \rangle \in \text{ARRET}$ si et seulement si $(\mathcal{A}_{\mathcal{M},w}, \mathbf{a}) \in \mathcal{L}_2$. Si $\langle [\mathcal{M}], w \rangle \in \text{ARRET}$ alors la simulation de $\mathcal{M}$ sur w termine donc $\mathcal{A}_{\mathcal{M},w}$ accepte tous les mots donc en particulier $\mathcal{A}_{\mathcal{M},w}$ accepte au moins un mot donc $\mathcal{A}_{\mathcal{M},w} \in \mathcal{L}_2$.

Inversement, si $\langle [\mathcal{M}], w \rangle \notin \text{ARRET}$ alors la simulation de $\mathcal{M}$ sur w diverge donc $\mathcal{A}_{\mathcal{M},w}$ refuse tous les mots donc en particulier n'accepte pas au moins un mot donc $\mathcal{A}_{\mathcal{M},w} \notin \mathcal{L}_2$.

1.3 Langage $\mathcal{L}_3$

Ce langage est décidable. Intuitivement il suffit de simuler $\mathcal{M}$ sur toutes les entrées de taille n mais en réalité il y a une petite subtilité car $\mathcal{M}$ pourrait accepter moins de n mots de taille n , mais en fait accepter un nombre infini de mots. Par exemple, si on prend $\mathcal{M}$ qui regarde uniquement les premières lettres de l'entrée et accepte si et seulement si c'est un $\mathbf{aa}$. Alors $\mathcal{M}$ accepte un nombre infini de mots, mais elle n'accepte qu'un seul mot de longueur 2.

Soit $\mathcal{M}$ et $n \in \mathbb{N}$, on pose $\mathcal{L}_{\mathcal{M} \leq n} = \{w \in \Sigma^* : \mathcal{M} \text{ accepte } w \text{ en moins de } n \text{ de étapes}\}$. On prétend que

$$\mathcal{L}_{\mathcal{M} \leq n} \text{ est infini} \quad \Leftrightarrow \quad \mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^n \neq \emptyset.$$

En effet, remarquons que s'il existe $w \in \mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^n$ alors $\mathcal{M}$ accepte w en moins de n étapes donc $\mathcal{M}$ ne peut lire que les n premières cases de l'entrée. En particulier, $\mathcal{M}$ se comporte exactement de la même façon sur w et sur wu pour tout $u \in \Sigma^*$ (puisque $\mathcal{M}$ ne peut jamais lire u puisque $|w| = n$), donc $\mathcal{M}$ accepte $w\Sigma^*$ en au plus n étapes ce qui montre bien que $\mathcal{L}_{\mathcal{M} \leq n}$ est infini. Inversement, si $\mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^n = \emptyset$ alors $\mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^{\geq n} = \emptyset$. En effet, si $\mathcal{M}$ acceptait un mot u de longueur $> n$, alors par le même raisonnement, $\mathcal{M}$ devrait accepter aussi le préfixe de u de longueur n , ce qui contredirait notre hypothèse. Ainsi $\mathcal{L}_{\mathcal{M} \leq n} \subseteq \Sigma^{< n}$ et donc est fini.

Nous pouvons maintenant montrer que $\mathcal{L}_3$ est décidable. Soit $\mathcal{M}$ et $n \in \mathbb{N}$: on commence par calculer $\mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^n$. Ceci est possible car Σ^n est fini et pour chaque mot il suffit de simuler $\mathcal{M}$ pendant n étapes au plus. Si cet ensemble est non vide alors $\mathcal{L}_{\mathcal{M} \leq n}$ est infini donc on accepte $\langle \mathcal{M}, \mathbf{a}^n \rangle$. Sinon, on sait que $\mathcal{L}_{\mathcal{M} \leq n} = \mathcal{L}_{\mathcal{M} \leq n} \cap \Sigma^{< n}$ donc on peut calculer cet ensemble (pour la même raison) et en déduire le nombre de mots acceptés en au plus n étapes, il suffit alors de comparer ce nombre à n .

1.4 Langage $\mathcal{L}_4$

Il s'agit d'une application du théorème de Rice. En effet, remarquons que pour toute machine $\mathcal{M}$, $n(\mathcal{M}) = n'(\mathcal{L}(\mathcal{M}))$ où pour tout langage $\mathcal{L}$,

$$n'(\mathcal{L}) = \begin{cases} 0 & \text{si } \mathcal{L} = \emptyset, \\ 1 + \min_{w \in \mathcal{L}} |w| & \text{sinon.} \end{cases}$$

Ainsi, $\mathcal{L}_4$ est exactement égal à $\text{Rice}(\mathcal{P})$ où

$$\mathcal{P} : \ll |\mathcal{L}| \leq n'(\mathcal{L}) \gg.$$

Or $\mathcal{P}$ est une propriété non triviale des langage semi-décidable car $\mathcal{P}(\emptyset)$ est vraie (puisque $|\emptyset| = 0 = n'(\emptyset)$) et $\mathcal{P}(\{\mathbf{a}, \varepsilon\})$ est fausse (puisque $|\{\mathbf{a}, \varepsilon\}| = 2 > 1 = n'(\{\mathbf{a}, \varepsilon\})$). On notera bien que $\emptyset$ et $\{\mathbf{a}, \varepsilon\}$ sont des langages semi-décidables. Ainsi par le théorème de Rice, $\mathcal{L}_4$ est indécidable.

Par ailleurs, le complémentaire de $\mathcal{L}_4$ est semi décidable. En effet, considérons l'algorithme suivant sur $\mathcal{M}$: on simule en parallèle $\mathcal{M}$ sur tous les mots, et on garde en mémoire :

- N : le nombre de mots acceptants trouvés jusqu'à maintenant (initialement 0),
- ℓ : un plus la longueur du plus court mot accepté trouvé jusqu'à maintenant (initialement ∞).

Si $N > \ell$ durant l'algorithme alors on accepte. Si $\mathcal{M} \in \overline{\mathcal{L}_4}$ alors $|\mathcal{L}(\mathcal{M})| > n(\mathcal{M})$, en particulier $\mathcal{L}(\mathcal{M})$ n'est pas vide donc $n(\mathcal{M}) > 0$. Donc il existe un mot accepté u de longueur $n(\mathcal{M}) - 1$ et il existe au moins $1 + n(\mathcal{M})$ mots acceptés par $\mathcal{M}$. Puisque l'algorithme ci-dessus énumère tous les mots, il finira par simuler entièrement le calcul de ces $1 + n(\mathcal{M})$ mots donc on aura $N \geq 1 + n(\mathcal{M})$. Par ailleurs, il simulera aussi le calcul sur u et conclura que $\ell \leq 1 + |u| = n(\mathcal{M})$ et donc l'algorithme acceptera puisqu'on aura $N > \ell$.

A l'inverse, si $\mathcal{M} \in \mathcal{L}_4$ alors on a $|\mathcal{L}(\mathcal{M})| \leq n(\mathcal{M})$ donc durant toute l'exécution de l'algorithme on aura $N \leq |\mathcal{L}(\mathcal{M})| \leq n(\mathcal{M})$. Et pour la même raison, on aura $\ell \geq n(\mathcal{M})$. Donc la condition $N > \ell$ ne sera jamais satisfaite et l'algorithme n'acceptera jamais.

1.5 Langage $\mathcal{L}_\infty$

Attention, il ne s'agit pas d'une instance du théorème de Rice ! Pour s'en convaincre, on peut voir que si on prend $\mathcal{M} \in \mathcal{L}_\infty$ et que l'on modifie $\mathcal{M}$ en $\mathcal{M}'$ en ajoutant un état inutile, alors $\mathcal{M}$ et $\mathcal{M}'$ acceptent le même langage, donc en particulier le même nombre de mots. Mais $|\mathcal{M}'| > |\mathcal{M}|$ donc $\mathcal{M}' \notin \mathcal{L}_\infty$, ce qui montre que $\mathcal{L}_\infty$ n'est pas une propriété de langage.

Toutefois $\mathcal{L}_\infty$ est indécidable, non semi-décidable et de complémentaire non semi-décidable. On commence par montrer que $\mathcal{L}_\infty$ n'est pas semi-décidable en faisant une réduction à partir du complémentaire de $\overline{\text{ARRET}}$ qui n'est pas semi-décidable. Soit $\langle [M], w \rangle$ une instance de $\overline{\text{ARRET}}$. On construit la machine $\mathcal{A}_{\mathcal{M},w}$ qui sur toute entrée $\langle x, u \rangle$:

- accepte si u est de la forme $\mathbf{a}^i$ avec $i \in \{1, \dots, |x|\}$,
- simule $\mathcal{M}$ sur w et accepte si la simulation termine (sinon elle diverge).

On utilise maintenant le théorème 9 du cours sur $\mathcal{A}_{\mathcal{M},w}$ pour construire $\mathcal{B}_{\mathcal{M},w}$ telle que pour tout entrée u , $\mathcal{B}_{\mathcal{M},w}(u) = \mathcal{A}_{\mathcal{M},w}(\langle [\mathcal{B}_{\mathcal{M},w}], u \rangle)$.

Montrons que $\langle [M], w \rangle \in \overline{\text{ARRET}}$ si et seulement si $[\mathcal{B}_{\mathcal{M},w}] \in \mathcal{L}_\infty$. Si $\langle [M], w \rangle \in \overline{\text{ARRET}}$ alors pour tout entrée u , $\mathcal{B}_{\mathcal{M},w}$ accepte u si et seulement si $\mathcal{A}_{\mathcal{M},w}$ accepte $\langle x, u \rangle$ avec $x = [\mathcal{B}_{\mathcal{M},w}]$. Or la simulation de $\mathcal{M}$ sur w ne termine jamais donc les seuls mots acceptés sont $u = \mathbf{a}^n$ avec $n \in \{1, \dots, |x|\}$. Or $|x| = |\mathcal{B}_{\mathcal{M},w}|$ est indépendant de u donc $\mathcal{B}_{\mathcal{M},w}$ accepte exactement $\mathbf{a}, \mathbf{a}^2, \dots, \mathbf{a}^{|\mathcal{B}_{\mathcal{M},w}|}$, c'est à dire $|\mathcal{B}_{\mathcal{M},w}|$ mots. Donc $\mathcal{B}_{\mathcal{M},w} \in \mathcal{L}_\infty$.

Inversement, supposons que $\langle [M], w \rangle \in \text{ARRET}$. Alors pour tout entrées u et x , $\mathcal{A}_{\mathcal{M},w}$ accepte car même si u n'est pas de la forme $u = \mathbf{a}^n$ pour une bonne valeur de n , la simulation de $\mathcal{M}$ sur w termine donc $\mathcal{A}_{\mathcal{M},w}$ accepte. Ainsi $\mathcal{B}_{\mathcal{M},w}$ accepte tous les mots donc en particulier un nombre infini, ce qui implique que $\mathcal{B}_{\mathcal{M},w} \notin \mathcal{L}_\infty$.

On utilise la même idée pour montrer que le complémentaire n'est pas semi-décidable en faisant encore une réduction à partir de $\overline{\text{ARRET}}$. Soit $\langle [M], w \rangle$ une instance de $\overline{\text{ARRET}}$. On construit la machine $\mathcal{C}_{\mathcal{M},w}$ qui sur toute entrée $\langle x, u \rangle$:

- refuse si u n'est pas de la forme $\mathbf{a}^i$ avec $i \in \{1, \dots, |x|\}$,
- simule $\mathcal{M}$ sur w et accepte si la simulation termine (sinon elle diverge).

On utilise de nouveau le théorème du théorème 9 du cours sur $\mathcal{C}_{\mathcal{M},w}$ pour construire $\mathcal{D}_{\mathcal{M},w}$ telle que pour tout entrée u , $\mathcal{D}_{\mathcal{M},w}(u) = \mathcal{C}_{\mathcal{M},w}(\langle [\mathcal{D}_{\mathcal{M},w}], u \rangle)$.

Montrons que $\langle [M], w \rangle \in \overline{\text{ARRET}}$ si et seulement si $[\mathcal{D}_{\mathcal{M},w}] \notin \mathcal{L}_\infty$. Si $\langle [M], w \rangle \in \overline{\text{ARRET}}$ alors pour tout entrées u et x , $\mathcal{C}_{\mathcal{M},w}$ refuse $\langle x, u \rangle$ car la simulation de $\mathcal{M}$ sur w diverge. Ainsi $\mathcal{D}_{\mathcal{M},w}$ n'accepte aucun mot. Par ailleurs, l'encodage d'une machine de Turing n'est jamais vide donc $|\mathcal{D}_{\mathcal{M},w}| > 0$. On a donc bien que $\mathcal{D}_{\mathcal{M},w} \notin \mathcal{L}_\infty$.

Inversement, supposons que $\langle [M], w \rangle \in \text{ARRET}$. Alors pour tout entrées u et x , $\mathcal{C}_{\mathcal{M},w}$ accepte exactement les mots de la forme $u = \mathbf{a}^n$ pour $n \in \{1, \dots, |x|\}$ car la simulation de $\mathcal{M}$ sur w termine. Ainsi par le même raisonnement que plus haut, $\mathcal{D}_{\mathcal{M},w}$ accepte exactement $\mathbf{a}, \mathbf{a}^2, \dots, \mathbf{a}^{|\mathcal{D}_{\mathcal{M},w}|}$, c'est à dire $|\mathcal{D}_{\mathcal{M},w}|$ mots. Donc $\mathcal{D}_{\mathcal{M},w} \in \mathcal{L}_\infty$.

2 Théorème Immerman–Szelepcsényi

2.1 Question 1

Il est clair que $N^{|V|}(S) \subseteq N^*(S)$, il s'agit donc de montrer l'autre inclusion. Soit $t \in N^*(S)$, alors il existe $s \in S$ et $s = v_0 \rightarrow \dots \rightarrow v_k = t$ avec $k \in \mathbb{N}$ et $v_i \in V$. S'il existe $i < j$ tels que $v_i = v_j$ alors on peut considérer le chemin

$$s = v_0 \rightarrow \dots \rightarrow v_i = v_j \rightarrow v_{j+1} \rightarrow \dots \rightarrow v_k = t.$$

Ce chemin est de longueur $k - (j - i) < k$. Par récurrence (sur la longueur), on peut donc lui appliquer le même raisonnement jusqu'à trouver un chemin de s à t dont tous les sommets sont distincts. Un tel chemin est alors nécessairement de longueur au plus $|V|$, ce qui montre que $t \in N^{|V|}(S)$.

2.2 Question 2

On sait que si $t \in N^d(s)$ alors il existe $v_0, \dots, v_k \in V$ avec $k \leq d$ tels que $s = v_0 \rightarrow \dots \rightarrow v_k = t$. Il suffit donc de deviner non déterministiquement les sommets intermédiaires et de vérifier :

- Poser $v = s$.
- Pour $i = 0, \dots, d$
 - Si $v = t$ alors accepter.
 - Deviner $v' \in V$.
 - Vérifier si $v \rightarrow v'$, rejeter si ce n'est pas le cas.
 - Poser $v = v'$.
- Rejeter.

Cette algorithmes a besoin d'un espace $\log d$ pour stocker i , d'un espace $O(\log |G|)$ pour stocker v et v' , et d'un espace $O(\log |G|)$ pour vérifier si $v \rightarrow v'$.

On vérifie que cet algorithme est correct : si $t \in N^d(s)$ alors il existe $v_0, \dots, v_k$ avec $k \leq d$ tels que $s = v_0 \rightarrow \dots \rightarrow v_k = t$. Donc dans la branche de calcul qui devine successivement $v_1, \dots, v_k$, la vérification sera bonne à chaque étape et l'algorithme accepte lorsqu'il vérifie que $v_k = t$.

Inversement si l'algorithme accepte alors il a construit une suite de sommets $v_0, \dots, v_k$ avec $k \leq d$ tels que $s = v_0 \rightarrow \dots \rightarrow v_k = t$ donc par définition $t \in N^d(s)$.

Puisque $N^*(s) = N^{|V|}(s)$, il suffit de prendre $d = |V|$ pour vérifier si $t \in N^*(s)$ et alors l'espace est en $O(\log |G| + \log |V|) = O(\log |G|)$.

2.3 Question 3

D'après la question 1, il suffit de vérifier que $v \neq t$ pour tout $v \in N^{|V|}(s)$. Or par le Lemme 1, on peut calculer la taille de cet ensemble. On peut donc deviner l'ensemble de ces éléments, vérifier qu'ils sont bien dans l'ensemble et vérifier qu'ils ne sont pas égaux à t . Il y a une petite subtilité : comment deviner les éléments de l'ensemble sans deviner deux fois le même ? On peut pour cela les deviner dans l'ordre lexicographique (n'importe quel ordre total sur les mots conviendrait).

- Calculer $n = |N^{|V|}(s)|$ par le Lemme 1
- Poser $v = \varepsilon$ (c'est le plus petit élément pour l'ordre lexicographique et il n'appartient pas à V).
- Pour $i = 1, \dots, n$
 - Deviner $v' > v$ (dans l'ordre lexicographique) dans V .
 - Si $\text{VerifChemin}(G, s, v')$ rejette, alors rejeter.
 - Vérifier que $v' \neq t$, rejeter si ce n'est pas le cas.
 - Poser $v = v'$.
- Accepter.

Cet algorithme fonctionne en espace $O(\log |G|)$ car on peut calculer n en cet espace d'après le Lemme 1. Il suffit d'un espace $O(\log |G|)$ pour stocker i puisque par définition $\log(i) \leq \log(n) \leq O(\log |G|)$. La vérification de v' se fait en espace $O(\log |G|)$ par la question 2.

On vérifie que cet algorithme est correct par la question précédente : si cet algorithme accepte alors il a deviné des sommets $v_1 < \dots < v_n$ dans $N^{|V|}(s)$ qui sont tous différents de t , avec $n = |N^{|V|}(s)|$. Ainsi, l'algorithme a énuméré tous les éléments de $N^{|V|}(s) = N^*(s)$ ce qui prouve que $t \notin N^*(s)$.

Inversement, si $t \notin N^*(s) = N^{|V|}(s)$ alors on peut lister les éléments $v_1 < \dots < v_n$ de $N^*(s)$ par ordre lexicographique, avec $n = |N^*(s)|$. Si l'algorithme devine les sommets dans cet ordre, la vérification de chaque configuration va marcher puisque le sommet est accessible depuis s mais différent de t . Donc l'algorithme accepte sur au moins une branche de calcul.

2.4 Question 4

Supposons que le Lemme 1 est vrai. Soit $G = (V, E)$ et $s, t \in V$ sur le ruban d'entrée. On lance l'algorithme de la question précédente et on accepte si et seulement s'il accepte. Par construction cet algorithme fonctionne en espace non déterministe $O(\log |G|)$ et accepte si et seulement si $t \notin N^*(s)$. Donc cet algorithme décide s-t-NON-CONN ce qui montre le résultat car la taille de l'entrée est $|G| + |s| + |t| \geq |G|$.

2.5 Question 5

Pour s'exécuter, l'algorithme doit stocker total et v , et tester si $v \rightarrow t$, ce qui prend seulement un espace $O(\log |G|)$ car $\text{total} \leq |V|$. L'appel à $\text{VerifChemin}(G, s, v, k)$ fonctionne en espace $O(\log |G|)$ par la question 2 puisque $k \leq |V| \leq |G|$.

2.6 Question 6

Si $t \in N^{k+1}(s)$ alors il existe $v \in N^k(s)$ tel que $v = t$ ou $v \rightarrow t$. Ainsi, lorsque l'algorithme regarde v avec la boucle, il suffit qu'il devine $b = 1$ pour ce v (et $b = 0$ pour les sommets avant v). Il suffit alors que $\text{VerifChemin}(G, s, v, k)$ suive une branche acceptante (qui existe par la question 2 car $v \in N^k(s)$). Il teste ensuite que $v = t$ ou $v \rightarrow t$, ce qui est vrai, et donc retourne « oui ».

2.7 Question 7

Supposons que l'algorithme retourne « oui », alors il a trouvé $v \in V$ puis une branche acceptante de $\text{VerifChemin}(G, s, v, k)$ et en enfin testé que $v = t$ ou $v \rightarrow t$. Par la question 2, ceci implique que $v \in N^k(s)$ et donc que $t \in N^{k+1}(s)$.

2.8 Question 8

Posons $n = |N^k(s)|$. Supposons que dans la boucle, l'algorithme devine $b = 0$ si $v \notin N^k(s)$ et devine $b = 1$ si $v \in N^k(s)$. Ainsi l'algorithme va incrémenter total exactement n fois s'il ne rejette pas entre temps. Or pour chaque $b = 1$, on peut suivre une branch acceptante de $\text{VerifChemin}(G, s, v, k)$ par la question 2 car $v \in N^k(s)$ puis le test $v = t$ ou $v \rightarrow t$ va échouer car sinon on aurait $t \in N^{k+1}(s)$, ce qui n'est pas le cas. Ainsi l'algorithme ne retourne jamais « oui » sur cette branche de calcul et termine la boucle avec $\text{total} = n$ sans rejeter. Le test final échoue et l'algorithme retourne « non ».

2.9 Question 9

Supposons que l'algorithme retourne « non ». Alors il a deviné $b = 1$ pour exactement $n = |N^k(s)|$ sommets puisque le test $\text{total} \neq n$ à la fin a échoué (sinon l'algorithme aurait rejeté). Notons $v_1, \dots, v_n$ ces sommets (distincts). Pour chacun de ces sommets, la vérification $v_i \in N^k(s)$ a réussi ce qui implique que $N^k(s) = \{v_1, \dots, v_n\}$ (sinon l'algorithme aurait rejeté). Par ailleurs, le test $v_i = t$ ou $v_i \rightarrow t$ est faux pour chaque v_i , ce qui implique donc bien que $t \notin N^{k+1}(s)$.

2.10 Question 10

On crée un compteur puis on énumère tous les sommets $t \in V$. Pour chacun, on lance l'algorithme $\text{Accessible}(G, s, t, k, n)$ et on incrémente notre compteur s'il retourne « oui ». On retourne la valeur du compteur à la fin.

Il est clair que cet algorithme fonctionne en espace $O(\log |G|)$ puisqu'il suffit de stocker t et de lancer Accessible ce qui fonctionne en espace $O(\log |G|)$ par la question 5.

La correction de l'algorithme est claire par les questions précédentes.

2.11 Question 11

On remarque que $|N^0(s)| = 1$. Grâce à la question 10, on peut calculer $|N^{k+1}(s)|$ si l'on connaît $|N^k(s)|$. On peut donc calculer $|N^{|V|}(s)|$ par une boucle. L'espace utilisé est celui de la question 10 plus un compteur allant de 1 à $|V|$, ce qui prend un espace $O(\log |G|)$.

2.12 Question 12

Soit $\mathcal{L} \in \text{NL}$. Soit $\mathcal{M}$ une machine de Turing non déterministe fonctionnant en espace $S(n)$ et décidant $\mathcal{L}$, avec $S(n) = O(\log n)$. Pour tout mot w , on définit le graphe orienté $\mathcal{G}_{\mathcal{M}, w} = (V_{\mathcal{M}, w}, E_{\mathcal{M}, w})$ ainsi :

- l'ensemble $V_{\mathcal{M}, w}$ des sommets est l'ensemble des configurations de $\mathcal{M}$ utilisant un espace au plus $S(|w|)$, **auquel on ajoute** un sommet spécial que l'on note C_{acc} (ce n'est pas une configuration)

— l'ensemble $E_{\mathcal{M},w}$ des arêtes est l'ensemble des (C, C') tels que C' est un successeur de C , **ou bien** C est acceptante et $C' = C_{acc}$.

On remarquera que ce graphe est de taille $|\Gamma|^{O(S(|w|))}$ et donc que $\log |G| = O(S(|w|))$ (puisque l'alphabet est fixe, il compte comme une constante). Clairement, tout sommet de $V_{\mathcal{M},w}$ peut se représenter en utilisant un espace $O(S(|w|))$ en stockant **uniquement le ruban de travail** (et pas l'entrée w !), l'état et la position de la tête. Etant donné deux configurations C et C' , on peut vérifier en espace $O(S(|w|))$ si $C \rightsquigarrow C'$ en utilisant la description de la machine $\mathcal{M}$ (qui est fixée, elle ne fait pas partie de l'entrée).

Vérifions par ailleurs que pour tout mot w , $\mathcal{M}$ rejette w si et seulement si $C_{acc} \notin N_{G_{\mathcal{M},w}}^*(C_0(w))$. En effet, si $\mathcal{M}$ rejette w alors pour toute configuration $C \in N_{G_{\mathcal{M},w}}^*(C_0(w))$, on a une suite $C_0(w) \rightsquigarrow C_1 \rightsquigarrow \dots \rightsquigarrow C_k = C$ de configurations mais alors C n'est pas acceptante (sinon $\mathcal{M}$ accepterait). Ainsi aucune configuration acceptante n'est accessible depuis $C_0(w)$, or la seule façon d'accéder à C_{acc} est d'atteindre une configuration acceptante, donc $C_{acc} \notin N_{G_{\mathcal{M},w}}^*(C_0(w))$.

Inversement, si $\mathcal{M}$ accepte w alors il existe une suite $C_0(w) \rightsquigarrow C_1 \rightsquigarrow \dots \rightsquigarrow C_k$ de configurations avec C_k acceptante et donc $C_k \rightarrow C_{acc}$ dans $G_{\mathcal{M},w}$, ce qui montre que $C_{acc} \in N_{G_{\mathcal{M},w}}^*(C_0(w))$.

Grâce au raisonnement ci-dessus, on conclut donc que pour décider si $\mathcal{M}$ rejette w , il faut et il suffit de décider si $C_{acc} \notin N_{G_{\mathcal{M},w}}^*(C_0(w))$, c'est à dire résoudre s-t-NON-CONN pour $G = G_{\mathcal{M},w}$, $s = C_0(w)$ et $t = C_{acc}$. Or on peut résoudre ce problème en espace non déterministe $O(\log |G_{\mathcal{M},w}|) = O(S(|w|))$ ce qui montre que $\overline{\mathcal{L}} \in \text{NL}$.

Il y a toutefois une petite subtilité : l'algorithme pour s-t-NON-CONN suppose que le graphe G est disponible sur le ruban d'entrée. Or sur l'entrée nous avons w mais nous n'avons pas l'espace pour écrire $G_{\mathcal{M},w}$. (cf problème de composer des réductions en espace logarithmique si on ne met pas de borne sur la taille de la sortie). Ici cela marche quand même car nous avons vu que pour décider s-t-NON-CONN, l'algorithme a seulement besoin de savoir énumérer les sommets de G et décider s'il existe une arête entre deux sommets. Or on peut prendre l'algorithme de s-t-NON-CONN et remplacer chaque test d'arête par un appel à une sous-machine qui vérifie qu'une configuration est successeure d'une autre (en ayant seulement w en entrée). Le résultat est bien une machine qui fonctionne en espace $O(\log |G_{\mathcal{M},w}|) = O(S(|w|))$.