SÉANCE DE SOUTIEN

1 Automates à piles

1.1 Si vous ne l'avez pas fait au partiel

1. Soit L un langage algébrique et soit $\mathcal{A}$ un automate à pile reconnaissant L. On suppose qu'il existe un entier k tel que pour toute entrée x, et toute exécution de $\mathcal{A}$ sur x, la pile de l'automate contient au plus k éléments à tout instant de l'exécution. Montrer que L est rationnel.

   A: *L'alphabet de pile Z est fini. Notons ℓ son nombre d'éléments. On construit un automate fini $\mathcal{A}'$ qui simule l'automate à pile $\mathcal{A}$, ce qui montrera que L est rationnel. Pour simuler la pile, on utilise le fait qu'elle est bornée. Comme la taille de la pile est toujours inférieure à k, on a au plus ℓ^k états possibles pour la pile. C'est un nombre fini d'états, que l'on va pouvoir mémoriser avec l'état de l'automate fini. Soit $\mathcal{Q}$ les états de l'automate à pile $\mathcal{A}$. Les états de notre automate fini $\mathcal{A}'$ vont être $\mathcal{Q} \times (Z \cup \{B\})^k$ où B est un symbole spécial n'appartenant pas à Z (B servira à indiquer si la pile n'est pas remplie jusqu'en haut). Soit δ la fonction de transition de $\mathcal{A}$, on définit la fonction de transition δ' de $\mathcal{A}'$ par*

$$\delta'((q, (z_1, \cdots, z_j, B, \cdots, B)), a) = (q_{dest}, (z_1, \cdots, z_{j-1}, z_{dest}, B, \cdots, B),$$

   où $(q_{dest}, z_{dest}) = \delta(q, a, z_j)$ et $z_j \neq B$. Noter que z_{dest} peut contenir plus d'une lettre (ça peut être un n-uplet), mais d'après la propriété de l'énoncé, on sait que z_{dest} ne fera jamais déborder la pile. L'état initial de $\mathcal{A}'$ est $(q_0, (z_0, B, \cdots, B))$ et l'état final est $(q_f, (B, \cdots, B))$ (si $\mathcal{A}$ reconnaît par état final et pile vide, sinon il faut adapter un peu).

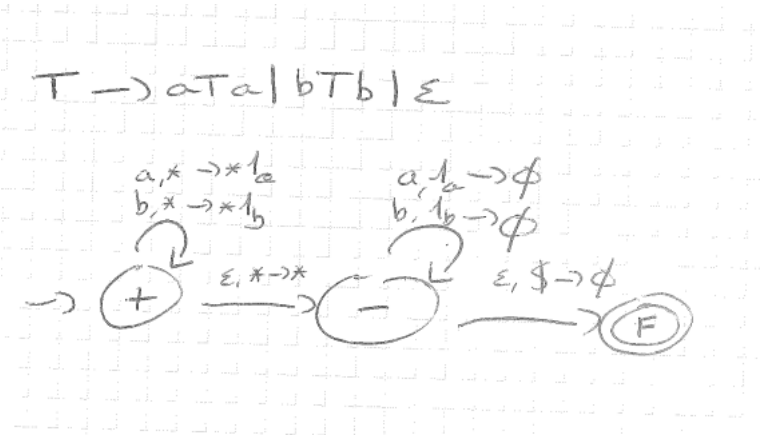
1.2 Construire des automates à pile

Construisez des automates à pile reconnaissant les langages suivants par état final, et justifier leur correction.

1. L le langage engendré par la grammaire suivante $S \to aSa \mid bSb \mid \varepsilon$.

   A: *Le langage reconnu est l'ensemble des mots qui sont des palindromes. Voici un automate à pile (non déterministe) qui le reconnaît par pile vide (et aussi état final). Dans toutes les solutions présentées ici, on suppose que le symbole de début de pile est le symbole \$. La transitions $a, 1 \to 21$ signifie que en lisant a et en ayant 1 en haut de la pile, on dépile 1 et on empile 21. Les transitions $a, 1 \to \emptyset$ signifient qu'on dépile 1 et qu'on ne remet rien sur la pile. Enfin, les transitions $a, * \to *1$ signifie que quelque soit le symbole $*$ en haut de la pile, on le dépile et on empile $*1$, où $*$ représente le même symbole que celui qui a été dépilé (globalement cela revient à ne pas dépiler et à ajouter 1 à la pile). Si une transition n'est pas représentée, cela signifie qu'elle mène dans un puits (le mot ne sera jamais accepté).*

   L'alphabet de pile de notre automate est $\{\$, 1_A, 1_b\}$.

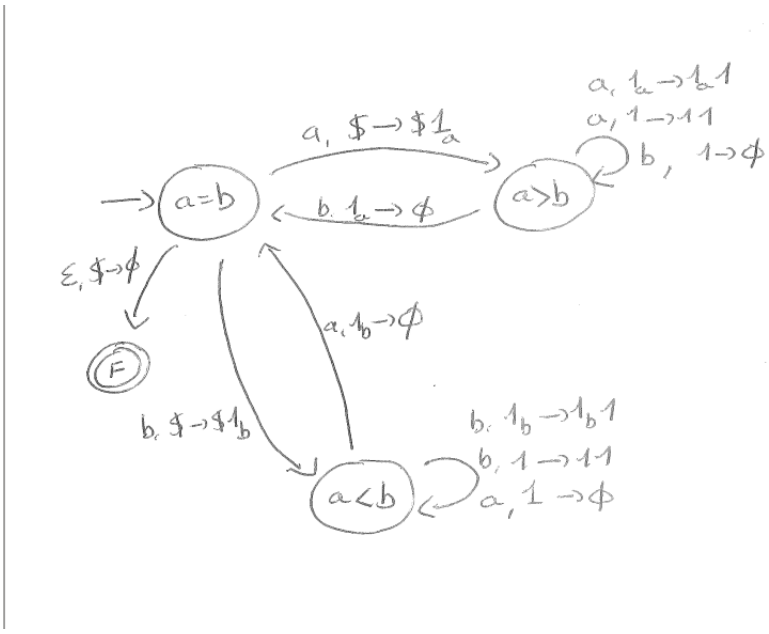


Si on veut reconnaître le mot $w\bar{w}$, on reste dans la partie $+$ de l'automate tant que l'on lit des lettres de w, puis on passe dans la partie $-$ pour lire les lettres de $\bar{w}$. Réciproquement, on peut se convaincre que tous les mots acceptés par cet automate sont de la forme $w\bar{w}$ pour un certain w.

2. $L = \{u \in \{a, b\}^* : |u|_a = |u|_b\}$.

A: *Avec les mêmes conventions que pour la question 1 pour les automates, on obtient l'automate suivant pour reconnaître L.*

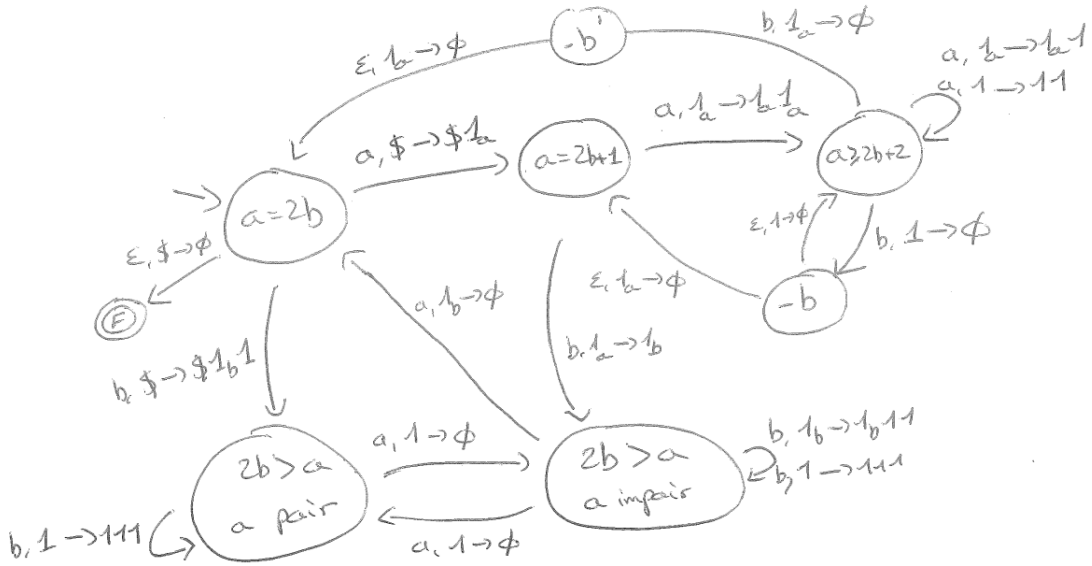
L'alphabet de pile de notre automate est $\{\$, 1_A, 1_b, 1\}$.



Les 1 dans la pile permettent de compter le nombre de a que l'on a en plus de b, si on est dans l'état $a > b$, et inversement si on est dans l'état $a < b$. Les lettres 1_a et 1_b permettent de savoir lorsque l'on revient au premier a ou au premier b (on aurait pu utiliser juste un symbole spécial, i.e. avoir $1_b = 1_a$, ou même utiliser des ε-transitions pour n'avoir aucun symboles spéciaux) et de revenir dans l'état d'égalité. L'état final est F, on doit y arriver avec la pile vide.

3. $L = \{u \in \{a, b\}^* : |u|_a = 2|u|_b\}$.

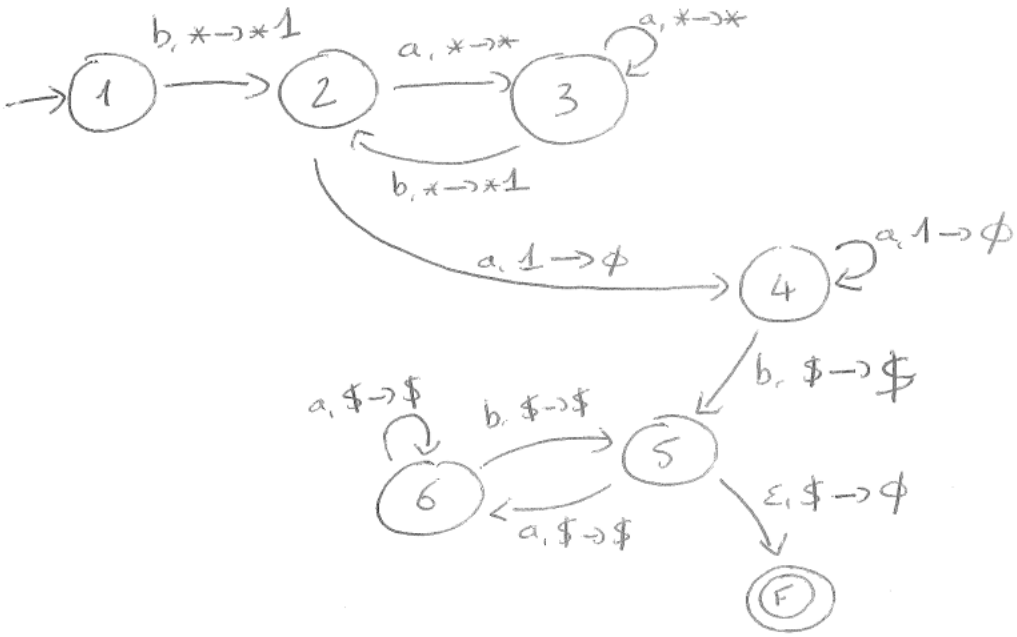
A: *L'automate est le suivant (l'alphabet de pile est toujours $\{\$, 1_A, 1_b, 1\}$).*



Lorsque l'on est dans la partie $a > 2b$, les 1 dans la pile comptent le nombre de a en trop pour avoir $a = 2b$ et lorsque l'on est dans la partie $a < 2b$, les 1 comptent le nombre de a qu'il manque pour obtenir $a = 2b$. Encore une fois, les symboles 1_a et 1_b permettent de savoir lorsque l'on revient à l'égalité (on pourrait s'en passer en utilisant des ε-transitions).

4. $L = \{ba^{i_1}ba^{i_2}b \cdots ba^{i_k}b : k \geq 1, \forall j \in [1,k]\ i_j \geq 1, \exists l \in [1,k] \text{ t.q. } i_l = l\}$.

A: *L'automate est le suivant (alphabet de pile $\{\$, 1\}$).*



L'idée est de reconnaître le motif ba^* en comptant le nombre de fois qu'il apparaît (on compte l'indice j). C'est ce qui est fait dans les états 1 à 3. Ensuite, lorsque l'on atteint l'indice l où l'on a $i_l = l$, on passe à l'état 4, où l'automate vérifie que le nombre de a est bien égal au nombre de 1 empilés jusqu'à présent (qui correspond à l). Une fois que la vérification est effectuée, on s'en fiche des indices i_j pour $j > l$ (on veut juste qu'ils soient supérieurs ou égaux à 1), donc on passe aux états 5 et 6 qui vérifient juste que la forme $ba^+ba^+b\cdots ba^+b$ est respectée (on n'a plus besoin de compter quoi que ce soit). Une fois que le mot est terminé, on sors dans l'état final F avec la pile vide.

2 Lemme d'Ogden

1. Soit $L = \{0^i1^j0^i1^j : i, j \geq 1\}$. Montrer que L n'est pas algébrique.

   A: *Supposons que L soit algébrique. Soit N la constante du lemme d'Ogden. On considère le mot $w = 0^N1^N0^N1^N$ dont toutes les positions sont distinguées. On a $|w| > N$ donc on peut écrire $w = uxyzv$ où x ou z est non vide et $|xyz| < N$. Comme xz est non vide, il contient au moins une lettre 0 ou 1. Supposons (sans perte de généralité) que xz contiennent une lettre 0. Comme $|xyz| < N$, xyz ne peut pas contenir des lettres 0 des deux blocs de 0. Le mot ux^2yz^2v a donc plus de 0 dans un bloc que dans l'autre (celui qui s'intersecte avec x ou z), et n'est donc pas dans L. Contradiction, on en déduit que L n'est pas algébrique.*

2. Montrer que le complémentaire de L est algébrique. Est-il rationnel ?

   A: *On considère les langages $L_1 = \{0,1\}^* \setminus 0^+1^+0^+1^+$, $L_2 = \cup_{m\neq n}0^m1^+0^n1^+$ et $L3 = \cup_{m\neq n}0^+1^m0^+1^n$. On remarque que $\{0,1\}^* \setminus L = L_1 \cup L_2 \cup L_3$, donc pour montrer que le complémentaire de L est algébrique, il suffit de montrer que L_1, L_2 et L_3 le sont (l'ensemble des langages algébriques est fermé par union finie).*

   Le langage L_1 est rationnel (car complémentaire d'un langage rationnel) donc algébrique.

   On montre que le langage L_2 est algébrique en exhibant une grammaire qui l'engendre.

   ```
   S -> UXV | XUV
   X -> aXa | V
   U -> U0 | 0
   V -> V1 | 1
   ```

   De même, le langage L_3 est algébrique, donc le complémentaire de L est bien algébrique.

   Le complémentaire de L n'est pas rationnel, car la classe des langages rationnels est stable par complémentaire. Donc si $\{0,1\}^* \setminus L$ était rationnel, L le serait aussi. Ce qui est contradictoire car on a montré que L n'est pas algébrique (donc en particulier pas rationnel).

3. Montrer que $L = \{a^ib^ic^j : j \neq i\}$ n'est pas algébrique.

   A: *On suppose par contradiction que L est algébrique. Soit N la constante du lemme d'Ogden. Ici il faut faire un peu attention aux positions distinguées, on ne peut pas choisir toutes les positions du mot comme étant distinguées. Soit $w = a^Nb^Nc^{N+M}$ pour un certain $M > 0$ que l'on précisera plus tard. On choisit comme marqueurs distinguées les a et les b de w. D'après le lemme d'Ogden, on peut factoriser $w = uxyzv$ avec x ou z qui contient au moins un a ou un b. Comme xyz contient au plus N marqueurs distingués et que le bloc de b contient N marqueurs distingués, on ne peut pas avoir à la fois des a et des c dans xyz. Supposons qu'il n'y ait pas de a dans xyz, alors on a au moins un b dans x ou dans z, et ux^2yz^2v contient plus de b que de a et n'est pas dans L. Ce n'est pas possible, on en déduit donc que xyz contient au moins un a et pas de c. Comme précédemment, si le nombre de a présents dans xz n'est pas le même que le nombre de b, alors ux^2yz^2v n'a pas le même nombre de a et de b et n'est pas dans L. Donc xz doit contenir autant de a que de b. Notons ℓ ce nombre. On sait que $1 \leq \ell \leq N$. On a alors que $ux^kyz^kv \in L$ contient $N + (k-1)\ell$ lettres a et b. On veut obtenir une contradiction, c'est à dire trouver k tel que $(k-1)\ell = M$ mais on ne choisit pas ℓ. En prenant $M = N!$, on est sûrs que quel que soit le choix de ℓ, il existera k tel que $(k-1)\ell = M$, et donc $ux^kyz^kv \notin L$. On a traité tous les cas possibles et on est arrivés à une contradiction à chaque fois. On en déduit donc que L n'est pas algébrique.*

3 Problèmes indécidables

1. Soient $\alpha, \beta \in \{0,1\}^*$. Montrer que le langage $L = \{\sigma : (M_\sigma(\alpha) = M_\sigma(\beta))$ ou $(M_\sigma(\alpha)$ et $M_\sigma(\beta)$ ne s'arrêtent pas$)\}$ n'est pas récursif. (Remarque : $M_\sigma(\alpha) = M_\sigma(\beta)$ sous-entend que M_σ s'arrête sur α et β).

   A: *On réduit L au problème de l'arrêt. Soit (M, x) une entrée du problème de l'arrêt. On veut construire une machine de Turing M_τ telle que τ soit dans L si et seulement si $M(x)$ s'arrête. Si L est décidable, on peut ensuite tester si M_τ est dans L ou non, et ainsi résoudre le problème de l'arrêt. Comme le problème de l'arrêt est indécidable on en déduira que L est indécidable aussi. Soit donc (M, x) une machine de Turing et une entrée pour lesquelles on veut tester si $M(x)$ s'arrête. Soit τ le code de la machine qui, sur l'entrée α fait délibérément une boucle infinie, et sur l'entrée β simule $M(x)$. Alors $\tau \in L$ si et seulement si $M(x)$ ne s'arrête pas, ce qui prouve que L n'est pas récursif.*

2. Montrer que le langage $L = \{\sigma : \exists x$ t.q. $M_\sigma(x)$ s'arrête en au plus $2|x|$ étapes$\}$ est indécidable. Est-il semi-récursif ?

   A: *Comme précédemment, soient M une machine de Turing et x une entrée. On veut construire une machine de Turing M_τ qui est dans L si et seulement si $M(x)$ s'arrête. Soit M_τ la machine qui, sur toute entrée y, efface son ruban puis simule $M(x)$. Si $M(x)$ ne s'arrête pas, alors $M_\tau(y)$ ne s'arrête sur aucune entrée y, et donc $\tau \notin L$. À l'inverse, si $M(x)$ s'arrête, soit k le nombre d'étapes nécessaires pour pour simuler $M(x)$ à partir d'une machine dont le ruban est vide. Alors pour toute entrée y de taille supérieure à k, $M_\tau(y)$ efface y en $|y|$ étapes, puis simule $M(x)$ en k étapes, ce qui est inférieur à $2|y|$. Donc $M_\tau(y)$ s'arrête en moins de $2|y|$ étapes et $\tau \in L$.*

   Le langage est semi-récursif. Étant donné une matrice M_σ, classer les entrées par ordre croissant de taille, et simuler M_σ sur chaque entrée x pendant $2|x|$ étapes. Si le calcul s'arrête, accepter M_σ. Si $\sigma \in L$, alors l'algorithme précédent s'arrêtera à un moment et acceptera M_σ.

4 Problème de correspondance de Post

Σ est un alphabet fini et P un ensemble fini de paires de mots sur Σ. Le Problème de Correspondance de Post associé à Σ, P est l'existence d'une suite finie non vide $(v_i, w_i)_i$ d'éléments de P telle que la concaténation des v_i soit égale à la concaténation des w_i. Le Problème de Correspondance de Post Modifié est celui de l'existence d'une telle suite lorsque le premier terme est fixé.

1. Résoudre PCP pour les instances suivantes :
   1. $P = (aab, ab), (bab, ba), (aab, abab)$
   2. $P = (a, ab), (ba, aba), (b, aba), (bba, b)$
   3. $P = (ab, bb), (aa, ba), (ab, abb), (bb, bab)$
   4. $P = (a, abb), (aab, b), (b, aa), (bb, bba)$

2. Montrer que si Σ ne contient qu'une lettre le problème est décidable.

3. Montrer l'équivalence entre PCP et PCPM, c'est à dire qu'à partir d'un algorithme résolvant toute instance de PCP, vous pouvez créer un algorithme résolvant toute instance de $PCPM$, et inversement.

   Indication : Dans le cas PCP permet de résoudre $PCPM$, vous pourrez ajouter à l'alphabet deux lettres $*$ et $\$$ et utiliser les deux morphismes p et s suivant :
   $\forall a_1, \ldots, a_k \in \Sigma^*$, $p(a_1 \ldots a_k) = *a_1 * \ldots * a_k$ et $s(a_1 \ldots a_k) = s_1 * \ldots s_k*$.

4. Montrer que PCP est indécidable.

   Indication : Pour montrer cela, vous pouvez montrer que PCP permet de résoudre l'arrêt : à une machine M (dont le ruban est semi-infini) et une entrée x on peut créer une instance de PCP qui est acceptée si et seulement si la machine M s'arrête sur l'entrée x.