IFT-2002 Informatique Théorique
Devoir 3

Question 1.

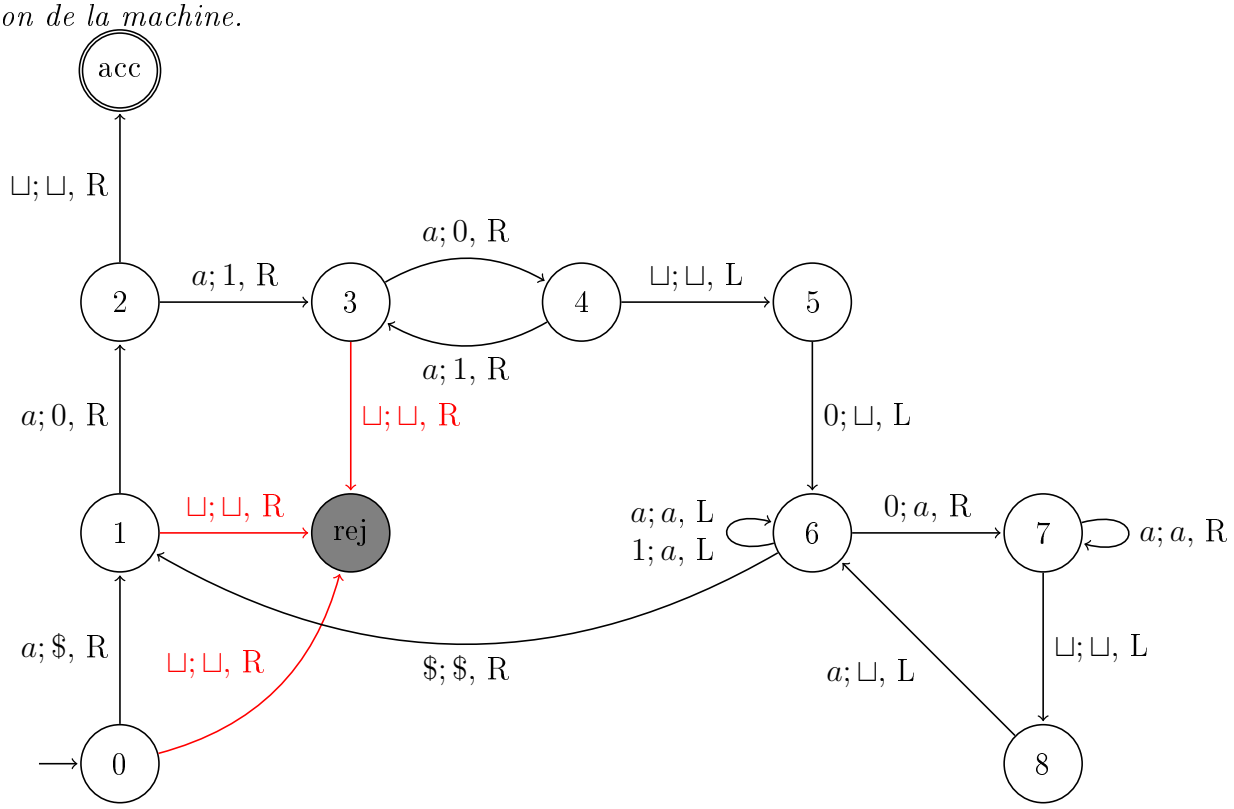
Donner le diagramme de transitions d'une machine de Turing qui accepte le langage $\{a^{2^n} : n \in \mathbb{N}^+\}$. Expliquez également le fonctionnement de votre machine (stratégie de haut niveau, rôle de chaque groupe d'états etc.) *Cette explication est obligatoire.*

Il y a plusieurs façons de faire mais la plus simple est sans doute d'éliminer à chaque passage la moitié des a.

Vous ne pouvez pas utiliser les outils de construction modulaire du chapitre 3.2 des anciennes notes de cours ni des machines à plusieurs rubans.

Réponse:

Les transitions manquantes s'en vont toutes à l'état rejetant, mais sont omises pour simplifier la présentation de la machine.



La stratégie de cet automate est de supprimer un caractère sur deux à chaque itération, et d'accepter si on arrive avec un mot de 2 caractères.

Pour se faire, on marque le début du mot avec un \$ (état 0) et on remplace le 2e a par un 0 (état 1).

(État 2)Si on arrive arrive au bout du ruban à ce moment, la machine accepte.

Sinon, les états 3 et 4 vont alterner la chaine avec des 1 et des 0 jusqu'au bout du mots sur le ruban, puis va regarder le derniere caractère du mot (État 5).

Les états 6, 7 et 8 vont boucler sur le ruban pour retirer tous les caractères 0 en s'assurant de ne pas avoir d'espaces et remplacer les 1 par des a. On se retrouve ainsi avec la moitié des lettres inscrite sur le ruban.

Dès que l'on trouve un \$, alors on sait que l'on est revenu au début du mot et tourne à l'état 1 pour refaire toutes la boucle.

Les états 0, 1 vérifient que le mot sur le ruban est d'au moins de longueur 2 et l'état 3 vérifie qu'il est de longueur pair. Si ce n'est pas le cas, le mot est refusé.

Question 2.

Un automate à deux piles consiste d'un ensemble fini d'états, d'un ruban d'entrée et de deux piles. Le fonctionnement est très semblable à celui d'un automate avec une seule pile: à chaque étape de son calcul, l'automate peut lire un symbole sur le ruban d'entrée, dépiler un symbole sur chaque pile et empiler un symbole sur chaque pile. Comme dans les automates à pile, l'automate n'est pas obligé de lire un symbole sur le ruban, ni de dépiler, ni d'empiler sur l'une ou l'autre des piles.

Un peu plus formellement un automate à deux piles est un sextuplet $P = (Q, \Sigma, \Gamma, \rho, q_0, F)$ où ρ est la relation de transition et

$$\rho \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q \times (\Gamma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}).$$

Une séquence w est acceptée si l'automate peut atteindre un des états acceptants en ayant terminé la lecture de w.

Montrez que tout langage Turing-acceptable peut être reconnu par un automate à deux piles. Autrement dit, montrez que pour toute machine de Turing M il existe un automate à deux piles P tel que $L(M) = L(P)$.

Vous n'avez pas à donner une preuve formelle mais simplement une explication à la fois claire et convaincante. (concise serait aussi appréciée)

Réponse:

Il est possible de voir le ruban de la machine de turing comme deux piles de cette façon:

Tous les éléments à la gauche de la tête de lecture se trouve dans la pile 1 (L'élément le plus près de la tête de lecture est au sommet de la pile).

Tous les éléments à la droite de la tête de lecture se trouve dans la pile 2 (L'élément le plus près de la tête de lecture est au sommet de la pile).

Au départ, la pile 1 est vide et la pile 2 contiens toute la séquence du ruban. L'automate commence par lire sur la pile 2. Si la machine de turing veut faire un déplacement à droite, alors on ajoute un symbole sur la pile 1. Dès qu'il y a un déplacement à gauche, on peut écrire sur la pile 2 un symbole, puis on se met à lire sur la pile 1 et ce, jusqu'à ce qu'il y est à nouveau un déplacement vers à droite sur le ruban.

Question 3.

Une *machine de Turing sans retour* est une machine de Turing M qui à chaque étape de son calcul bouge systématiquement sa tête de lecture/écriture vers la droite.
Le but de l'exercice est de comprendre la puissance de calcul des machines de Turing sans retour. Dites lequel des trois énoncés suivants est vrai (il n'est pas nécessaire de justifier votre réponse)

- un langage L peut être reconnu par une machine de Turing sans retour si et seulement si L peut-être reconnu par un automate fini.
- un langage L peut être reconnu par une machine de Turing sans retour si et seulement si L peut-être reconnu par un automate à pile.
- un langage L peut être reconnu par une machine de Turing sans retour si et seulement si L peut-être reconnu par une machine de Turing.

Réponse:
Un langage L peut être reconnu par une machine de Turing sans retour si et seulement si L peut-être reconnu par un **automate fini**.

Question 4.

Montrez que le langage suivant est décidable:

$P_{AFD} = \{\langle M \rangle : M$ est un automate fini déterministe et $L(M)$ contient au moins un mot de longueur paire$\}$.

En d'autres mots, donnez un algorithme qui reçoit en entrée un automate fini déterministe et qui détermine si oui ou non cet automate accepte au moins un mot de longueur paire. Cet algorithme doit s'arrêter peu importe l'automate fourni en entrée.

Réponse:

Obtenir le nombre d'état de M et nommons ce nombre n.
Si n est impaire, retirer 1 et affecter cette nouvelle valeur à n.
Obtenir un iterateur i qui commence à 0.

Tant que $i < n$ faire

- Pour chaque tous les mots possible de longueur i en utilisant l'alphabet de M, vérifier si ce mot est accepté par M.
- Si le mot est accepté par M: *Accepter*.
- Sinon, augmenter i de 2 et retourner au début de la boucle.

Si la machine ne s'est pas arrêter avec *Accepter*, alors $Refuser$.

Question 5. Montrez que le langage suivant est Turing-acceptable.
$\text{NE}_{\text{GNC}} = \{\langle G, H\rangle : G \text{ et } H \text{ sont des grammaires non-contextuelles en forme normale de Chomsky et } L(G) \neq L(H)\}$.

En d'autres mots, décrivez un algorithme A qui reçoit en entrée deux grammaires non-contextuelles en forme normale de Chomsky et qui a le comportement suivant. Si $L(G) \neq L(H)$, l'algorithme A s'arrête et accepte après un nombre fini d'étapes mais si $L(G) = L(H)$ alors A soit rejette après un nombre fini d'étapes soit ne s'arrête jamais.

Réponse:

(1) Si l'alphabet de symbole terminaux de G est différent de l'alphabet de symbole terminaux de H, alors $Refuser$.

(2) Utiliser la force brute pour générer un mot w avec les symboles terminaux.

(3) En utilisant la force brute, tenter de faire une dérivation de G en utilisant au plus $2|w| - 1$ étapes. Si trouvé, $R_1 = Vrai$ sinon, $R_1 = Faux$.

(4) En utilisant la force brute, tenter de faire une dérivation de H en utilisant au plus $2|w| - 1$ étapes. Si trouvé, $R_2 = Vrai$ sinon, $R_2 = Faux$.

(5) Si $R_1 = R_2$, générer un nouveau w (le mot suivant) et retourner à l'étape (3) avec de nouveau w.

(6) Si $R_1 \neq R_2$, alors $Refuser$.

Question 6.

Montrez que le langage suivant est indécidable

$P_{\text{MT}} = \{\langle M \rangle : M \text{ est une machine de Turing qui accepte au moins un mot de longueur paire}\}$.

En d'autres mots, vous voulez montrer qu'il n'existe aucun algorithme capable de prendre en entrée un programme M et de déterminer s'il existe un x de longueur paire tel que M accepte x.

Réponse:

Question 7. Vrai ou faux? Justifiez brièvement.

1. Si L est un langage non-contextuel alors son complément L^c est décidable.

   Réponse

   Vrai, selon le théorème que si L est décidable, alors le langage L^c est aussi décidable. Ainsi que le théorème qui indique que le langage non-contextuel est décidable.

   $\text{EstNonContextuel}(L) \implies \text{EstDecidable}(L) \implies \text{EstDecidable}(L^c)$

2. Si L_1 et L_2 sont des langages Turing-acceptables, alors $L_1 - L_2$ est aussi Turing-acceptable.

   Histoire d'éviter toute ambiguïté, $L_1 - L_2 = \{x : x \in L_1 \wedge x \notin L_2\}$.

   Réponse:

   Faux, si nous avons un langage L_2 qui refuse toutes les mots de L_1, alors en aucun cas, la machine $L_1 - L_2$ ne pourra jamais s'arrêter en acceptant.

Question facultative.

Soit $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n = 0$ une équation avec $a_0, \ldots, a_n \in \mathbb{Z}$. Appelons $a_{max} = \max_i |a_i|$.

Montrez que si cette équation possède une solution alors elle possède une solution dans laquelle $|x| \leq a_{max}(n+1)$.

Montrez qu'il existe un algorithme qui étant donné une équation de ce type, détermine en temps fini si elle possède ou non une solution x telle que $x \in \mathbb{Z}$.