# Travaux Dirigés d’Introduction à la Calculabilité | Chapitre 1

## Exercice 2 :

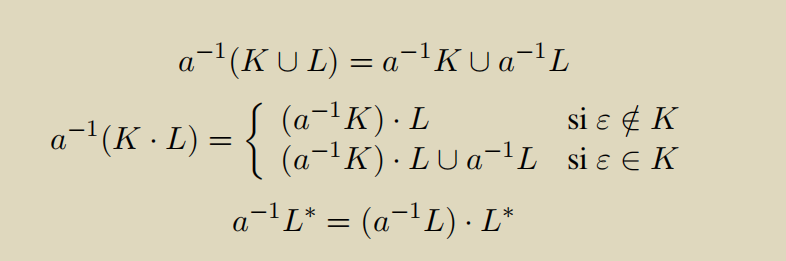
Soit A = {a, b}. Montrer que le langage de A∗ formé des mots comportant le facteur ab mais pas le facteur aa est rationnel.   
Soit E une expression rationnelle dénotée par le langage. Montrons que L(E) est rationnel. On a :  
E = b\*(ab)(ab)\*b\*.   
L(E) = L(b\*(ab)(ab)\*b\*) = L(b\*) L(ab) L((ab)\*) L(b\*) = L(b\*) L(ab) L(ab)\*L(b\*)  
L(E) = L(b)\* L(ab) L(ab)\* L(b)\* = {b}\*{ab}{ab}\*{b}\*  
Ce langage est bel et bien rationnel car la classe des langages rationnelles est stable par l’étoile et la concaténation.

## Exercice 3 :

Soit A = {a, b}. Montrer que le langage de A∗ formé des mots comportant soit le facteur ab soit le facteur ba mais pas les deux est rationnel.   
De la même manière que précédemment, on a : E = a\*(ab)b\* + b\*(ba)a\*. Ainsi :   
L(E) = L(a\*(ab)b\* + b\*(ba)a\*) = L(a\*)L(ab)L(b\*) + L(b\*)L(ba)L(a\*) = L(a)\*L(ab)L(b)\* + L(b)\*L(ba)L(a)\*  
L(E) = ({a}\*{ab}{b}\*) ∪ ({b}\*{ab}{a}\*  
Ce langage est régulier car la classe des langages réguliers est stable par l’étoile, la concaténation et l’union.

## Exercice 4

On rappelle que si u est un mot de A∗ et L est un langage de A∗, on définit u−1L = {v ∈ A∗ | uv ∈ L}. Soit a ∈ A une lettre et soient K, L ⊆ A∗ deux langages.

Montrer que :

**Pour effectuer ces démonstrations, nous procéderons par une double inclusion :**

* **Demo 1**
  + **P1**
  + **P2**
* **Deom2**
  + **P2**
  + **P2**
* **Demo3**
  + **P1**
  + **P2**

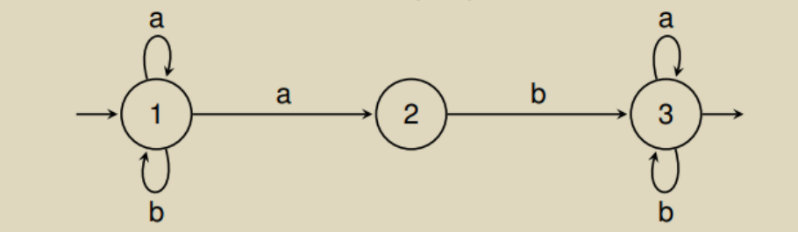
2. En déduire que si L est un langage rationnel alors a−1L est aussi un langage rationnel.

3. Montrer que si L ⊆ A∗ est un langage rationnel et si u ∈ A∗ alors u−1L et Lu−1 sont aussi rationnels.

* Ici

# Travaux Dirigés sur les Automates finis | Chapitre 2

## Exercice 1

On considère l’automate A de **l’exemple 2** du cours :

1) Montrer que le mot a3ba est l’étiquette d’exactement quatre chemins : de 1 à 1, de 1 à 2, de

1 à 3 et de 3 à 3.

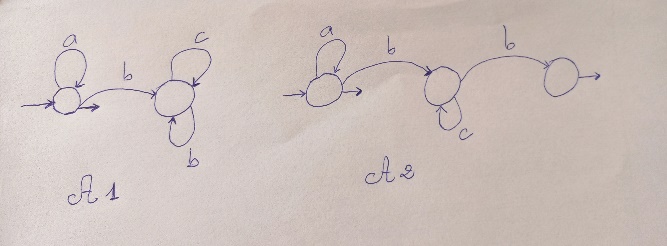
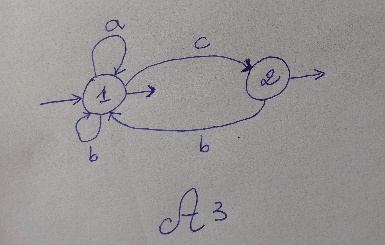
* Un chemin dans A est une suite c de transitions consécutives. Autrement dit, l’état d’arrivée de chaque transition est l’état de départ de la transition suivante. Partant de cette définition, on constate que le chemin donné ci-dessus est erroné. En guise de rectification, nous proposons le chemin : de 1 à 1, de 1 à 2, ***de 2 à 3*** et de 3 à 3. (1 à 3 → 2 à 3)  
  De là, nous tirons le chemin c = (1,a,1) (1,a,1) (1,a,2) (2, b,3) (3,a,3) et son étiquette correspondant e = aaaba = a3ba. Ce chemin étant bel et bien réussi, donc a3ba est l’étiquette d’exactement quatre chemins.

2) Montrer que l’ensemble des mots qui contiennent au moins une occurrence de la lettre a suivie d’un b, c’est-à-dire le langage L2 = Σ∗abΣ∗ est reconnu par l’automate de l’exemple 2.

* On sait qu’un chemin est réussi ssi son état initial est dans *I* et son état final dans *F.* Un mot w est reconnu par A s’il existe dans A un chemin réussi d’étiquette w. Le langage reconnu par A est l’ensemble des chemins réussis de A.   
  Pour qu’un chemin soit réussi dans A, il faudra passer successivement par les transitions (1,a,2) et (2,b,3). L’étiquette du chemin correspondant à ces transitions étant ***ab***, nous concluons que tout mot de L2, contenant au moins une occurrence de la lettre a suivie d’un b, peut constituer une étiquette d’un chemin réussi dans A. (En effet ∀ w ∊ L2, son état initial est dans I et son état final dans F). Donc L2 reconnu par l’automate.

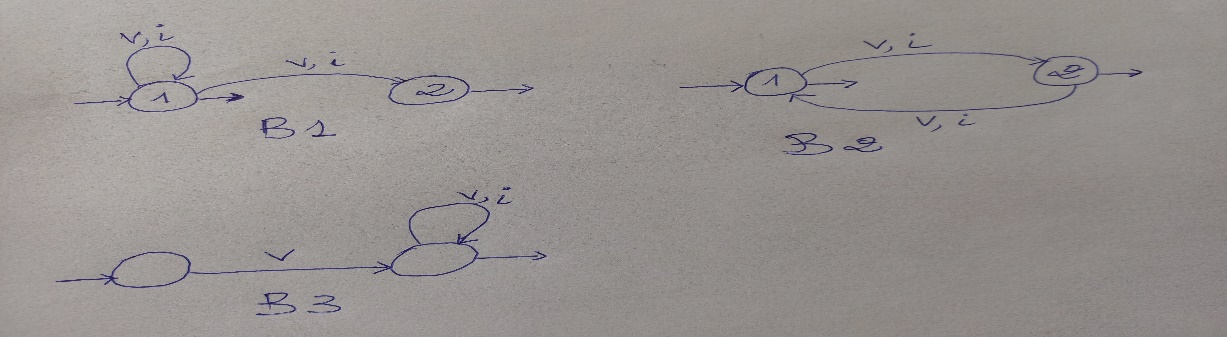
3) Remarque : **Un langage donné peut être reconnu par plus d’un automate.** Illustrer cette remarque par des exemples.

* Soit Σ = {a,b,c}. Considérons le langage L = {a,abcb} de Σ.  
  Ce langage est reconnu par les trois automates ci-dessous.



**Figure 1**

* Considérons à présent le langage L = viΣ\* de Σ = {v,i}. Les automates de la figure 2 ci-dessous reconnaissent bel et bien ce langage. D’où la validité de la remarque.



**Figure 2**

4) Montrons que l’automate de l’exemple 2 n’est pas complet.

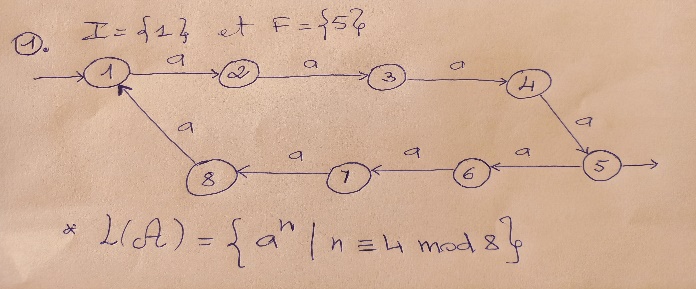
* Un automate A = (Q,T,I,F) est complet si pour état q ∊ Q et toute lettre a ∊ Σ, ∃ au moins un état q’ tel que (q,a,q’) ∊ T. Ainsi, en considérant l’état 2 de l’automate et la lettre a de Σ, on constate qu’il n’existe pas une transition de la forme (2,a,q’) tel que q’ ∊ Q. Donc l’automate n’est pas complet.

## Exercice 2

Soit un automate à 8 états sur l’alphabet constitué du singleton {a}, l’état I = 1 et l’état F = 5.

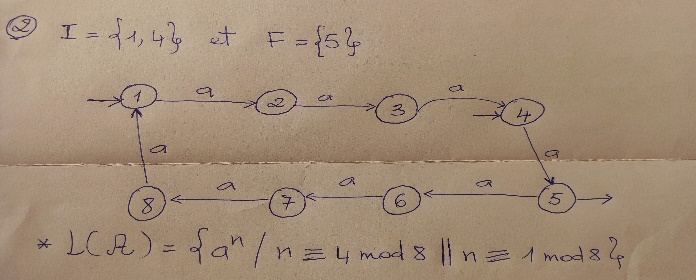
1. Déterminer L(A).

* L(A) = {an | n ≡ 4 mod 8}



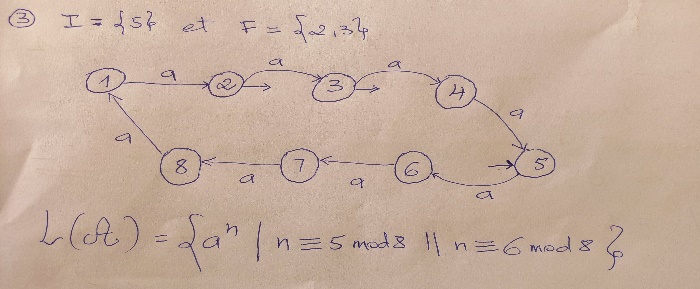
1. Déterminer L(A) pour I = 1 et 4 et F = 5

* L(A) = {an | n ≡ 4 mod 8 || n ≡ 1 mod 8}



1. Déterminer L(A) pour I = 5 et F = 1 et 3

* L(A) = {an | n ≡ 5 mod 8 || n ≡ 6 mod 8}



# Chapitre de Synthèse | Calculabilité

## Recherches complémentaires :

1. **Fonctions élémentaires | Fonctions primitives récursives :** 
   1. **Définition de Fonctions primitives récursives :**

Ces fonctions sont définies à partir de fonctions de base et d’opérations élémentaires et regroupent presque tous les algorithmes que nous connaissons.

Une fonction (possiblement partielle**) f : Nn → N** est **récursive** si elle est soit **la constante 0**, soit l’une des fonctions :

* Zero : x → 0 la fonction 0 ;
* Succ : x → x + 1 la fonction successeur ;
* Projin : (x1, . . . , xn) → xi les fonctions de projection, pour 1 ≤ i ≤ n ;
* Compm(g, h1, . . . , hm) : (x1, . . . , xn) → g(h1(x1, . . . , xn), . . . , hm(x1, . . . , xn)), la composition des fonctions récursives g, h1, . . . , hm ;
* Rec(g, h) la fonction définie par récurrence comme : f(0, x2, . . . , xn) = g(x2, . . . , xn) et f(x1 + 1, x2, . . . , xn) = h(f(x1, . . . , xn), x1, . . . , xn), où g et h sont récursives
* Min(g) la fonction qui à (x2, . . . , xn) associe le plus petit y ∈ N tel que g(y, x2, . . . , xn) = 1 s’il en existe (et qui n’est pas définie sinon) où g est récursive.  
  Une fonction primitive récursive est une fonction qui se définit sans utiliser le schéma Min.
  1. **Lien entre fonction récursives et machines de Turing :**
* Chaque fonction récursive (partielle) peut être calculée par une machine de Turing.
* Chaque fonction calculée par une machine de Turing est récursive (partielle).
* Une fonction de Nk dans N est récursive si et seulement si elle est **calculable** par une machine de Turing
* La classe de fonctions calculables par machines de Turing est égale à la classe de fonctions calculables par un algorithme quelconque (***Thèse de Church-Turing*).**

1. Trouvons des exemples de problèmes indécidables en :
   1. **Logique**
      1. **Le paradoxe du menteur**

Ce paradoxe consiste à dire « je mens » ou, sous une forme plus précise : « la présente phrase est fausse ». Si cette phrase est vraie, alors elle est fausse (puisque c'est ce qu'elle dit) ; et si elle est fausse, comme c'est précisément ce qu'elle dit, elle est vraie ! On ne peut pas lui attribuer une valeur de vérité de façon cohérente.

* + 1. **Le paradoxe de l'interrogation surprise :**

Un professeur annonce à ses élèves : « *Il y aura une interrogation surprise la semaine prochaine.* » Sachant que l’interrogation ne peut avoir lieu que soit le lundi, soit le mardi, soit le mercredi, soit le jeudi ou le vendredi ;

**Un élève *futé*** fait le raisonnement suivant : « Si jeudi soir, l'interrogation n'a pas eu lieu, alors je serai certain qu'elle est pour vendredi. Ce ne sera donc plus une surprise. L'interrogation ne peut donc avoir lieu vendredi parce que c'est le dernier jour possible. Mais puisque l'interrogation ne peut avoir lieu le dernier jour, l'avant-dernier jour devient **de** **facto**, (véritablement) le dernier jour possible. Ainsi, par récurrence, j'en déduit que l'interrogation ne peut avoir lieu. »

* 1. **Langage**
     1. **Le problème d’arrêt (HALTING-PROBLEM) :**

On se donne une paire (« M », w), où « M » est le codage d’une machine de Turing M, et w est un mot, et l’on souhaite décider si la machine M accepte le mot w.

**Objectif :** Décider si la machine M accepte le mot w.

* + 1. **Problème associé à un langage L**

On se donne un mot w. On se pose comme défi de *Décider si w ∈ L.*

Ce problème, de même que ceux ont précédé sont tous indécidables de nature.

1. **Expliquer les termes : *modèle raisonnable, effectivement calculable***

* **Modèle raisonnable :**
* **Fonction calculable :** Soient Σ et Σ’ deux alphabets. Une fonction **f : Σ∗ → Σ’∗** est calculable s’il existe une machine de Turing A, qui travaille sur l’alphabet Σ∪Σ’, telle que pour tout mot w, A avec l’entrée w, termine et accepte, avec f(w) écrit sur son ruban au moment où elle termine.

1. Donner des exemples de problèmes de décisions et trouver les problèmes d’optimisation correspondantes et inversement :

Le but d’un problème d’optimisation est de trouver une solution maximisant (resp. minimisant) une fonction objectif donnée.

A chaque problème d’optimisation on peut associer un problème de décision dont le but est de déterminer s’il existe une solution pour laquelle la fonction objective soit supérieure (resp. inférieure) ou égale à une valeur donnée. La complexité d’un problème d’optimisation est liée à celle du problème de décision qui lui est associé. En particulier, si le problème de décision est NP-complet, alors le problème d’optimisation est dit NP-difficile.

* 1. Problèmes de décisions (**D)** et d’optimisation correspondantes **(O)** :

-------------

* 1. Problèmes d’optimisation et de décisions correspondantes :

-------------

1. **D’autres de complexités avec leurs spécificités :** 
   1. **Classes L et NL** : un problème de décision qui peut être résolu par un algorithme déterministe en espace logarithmique par rapport à la taille de l'instance. L = SPACE(log(n)). La classe NL s'apparente à celle de L mais sur une machine non déterministe (NL = NSPACE(log(n)). *Par exemple savoir si un élément appartient à un tableau trié peut se faire en espace logarithmique.*
   2. **Les classes EXPTIME et NEXPTIME** : elles contiennent des problèmes dont la résolution n’est pas aisée car ce sont des tours d’exponentielle.
   3. **Les classes PSPACE et NPSPACE** : classes spatiales robustes, leur complétude se défini à l’aide d’une réduction polynomiale.
2. **Illustrer la notion de *Réduction* d’un problème**

Soient A et B deux problèmes d’alphabet respectifs MA et MB. Une **réduction de A vers B** est une **fonction f : M∗A → M∗B calculable** telle que w ∈ A ssi **f(w) ∈ B**. On note A ≤m B lorsque A se réduit à B. Quelques exemples :

* 1. Problème difficile : ***Voyager de Buenos Aires à Paris.*** Mais il n’y a de vol direct que de *Buenos Aires* à *Dakar*. Donc nous devons réduire le problème à un autre plus facile.

Problème facile : ***Prendre un vol de Dakar à Paris.*** Ainsi si l’on peut trouver une solution pour résoudre ce problème-ci, on pourra l’utiliser pour résoudre le problème difficile.

* 1. Problème difficile : ***Vivre éternellement*.** (Ce qui est impossible bien-sûr)

Problème facile : ***Arrêter de vieillir***. Si nous pouvons trouver une solution au problème de *vivre pour toujours,* on pourra également résoudre le problème du *vieillissement.* Mais cela étant impossible, on conclut donc qu’il est impossible de ne pas vieillir.