Final IT40

Exercice 1 : Des expressions régulières et des automates finis

Pour chacun des langages suivants, donner une expression régulière qui le décrit et dessiner l'automate fini le plus simple possible qui le reconnaît (il n'est pas demandé un automate fini déterministe mais il peut être utile d'en construire un pour la dernière question) :

1. langage des mots sur $\Sigma = a \mid b \mid c$ qui contiennent le mot aa *ou* le mot cc,
2. langage des mots sur $\Sigma = a \mid b \mid c$ qui contiennent le mot aa *et* le mot cc,
3. langage des mots sur $\Sigma = a \mid b \mid c$ qui contiennent le mot ab *et* le mot ba,
4. langage des mots sur $\Sigma = a \mid b \mid c$ ne contenant aucune occurrence du mot ab.

Exercice 2 : Automate minimal

On envisage l'automate fini défini par sa table de transitions

			a	b	c
$-$	0		0	1	1
	1		1	2	2
	2	$+$	2	3	3
	3		3	4	4
	4		4	5	5
	5	$+$	5	0	0

Dessiner cet AF et expliquer pourquoi il est déterministe et complet.
Par l'algorithme de `Hopcroft` (autrement dit avec les peper), construire l'AFdc minimal équivalent et représenter cet AFdc. Quelle propriété arithmétique explique la simplification obtenue ?

Exercice 3 : Une grammaire

Soit la grammaire

$$G = (\{a, b, c\}, \{S, X, Y, Z\}, S, \Pi)$$

où Π est l'ensemble des règles de production :

$$\begin{array}{lcl} \texttt{R1} & : & S \to aXb \\ \texttt{R2} & : & X \to aSb \\ \texttt{R3} & : & X \to bYa \\ \texttt{R4} & : & Y \to bYa \\ \texttt{R5} & : & Y \to cZ \\ \texttt{R6} & : & Z \to ccZ \\ \texttt{R7} & : & Z \to \varepsilon \end{array}$$

1. Cette grammaire est elle linéaire ? Justifiez votre réponse.
2. Si on dérive un mot en utilisant exactement k fois la règle `R2`, l fois la règle `R4` et m fois la règle `R6`, quel mot obtient on ? [On remarquera que si on utilise la règle `R2` alors on a du utiliser la règle `R1`]
3. Donner une définition pour le langage $L(G)$ des mots engendrés par cette grammaire. Ce langage est-il régulier ?
4. Montrer à l'aide du lemme de l'étoile qu'il ne peut pas exister d'automate qui reconnaisse ce langage.

Exercice 4 : Une réussite

Nous allons étudier une réussite[1] dont les règles sont les suivantes :
— Les cartes sont placées régulièrement et de manière parallèle sur la table pour former un carré (voir Figure 1).

1. un jeu de carte auquel on joue seul

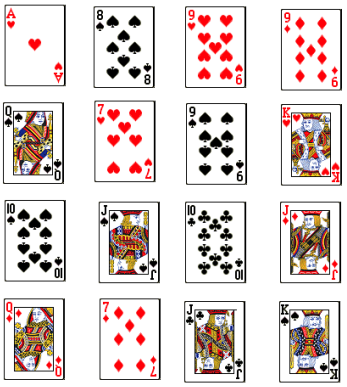

FIGURE 1 – Exemple de Réussite (gagnante).

— Les cartes placées sont tirées au hasard. On dit que la réussite est gagnante s'il est possible de passer de la carte située en haut à gauche du rectangle (sur la Figure 1 donnée en exemple, l'as de cœur) vers la carte située en bas à droite (le roi de pique sur cet exemple).
— On a le droit de passer d'une carte à l'autre si les deux cartes sont sur la même ligne ou sur la même colonne **et** si elles sont de même couleur [2] ou de même valeur.

1. Expliquer comment vous pouvez modéliser une réussite par un graphe.
2. Construire sur votre copie le graphe associé au jeu de la Figure 1.
3. Énoncer très exactement le problème que l'on cherche à résoudre sur le graphe pour savoir si la réussite est gagnante.
4. A l'aide de l'algorithme qui vous paraît le mieux adapté (dont vous donnerez le nom) montrer que la réussite de la figure 1 est gagnante : vous ferez apparaître de manière succincte le déroulement de l'algorithme sur votre feuille.

2. Nous rappelons que, quand on parle de jeu de carte, couleur ne veut pas dire rouge ou noir, mais bien pique, cœur, trèfle et carreau.

Annexes

Algorithm 1 Algorithme de parcours en Largeur d'abord d'un graphe G en partant du sommet i. Cet algorithme utilise une file F et nécessite une méthode pour marquer les sommets. Au début, tous les sommets sont non marqués. A la fin de l'algorithme $\pi(s)$ est le prédécesseur de s sur le parcours obtenu.

ParcoursLargeur(G,i)

1: $F \leftarrow \emptyset$; marquer i ; enfiler i dans F
2: **while** $F \neq \emptyset$ **do**
3: $s \leftarrow$ tête de F ; défiler F
4: **for** $w \in \Gamma^+(s)$ **do**
5: **if** w n'est pas marqué **then**
6: marquer w ; enfiler w dans F
7: **end if**
8: **end for**
9: **end while**

Algorithm 2 Algorithme de parcours en profondeur d'abord d'un graphe G en partant du sommet i (version récursive). Cet algorithme nécessite une méthode pour marquer les sommets. Au début, tous les sommets sont non marqués. A la fin de l'algorithme $\pi(s)$ est le prédécesseur de s sur le parcours obtenu.

ParcoursProfondeur(G,i)

1: **for** $s \in S$ **do**
2: $\pi(s) \leftarrow null$
3: **end for**
4: Prof(G,i)

Prof(G,i)

1: marquer le sommet i
2: **for** $v \in \Gamma^+(i)$ **do**
3: **if** v n'est pas marqué **then**
4: $\pi(v) \leftarrow i$
5: Prof(G,v)
6: **end if**
7: **end for**