DS 3

Option informatique, deuxième année

Julien Reichert

Partie 1 : Logique

Vous participez à un concours de mathématiques comportant une partie de raisonnement logique. Plusieurs orateurs font des déclarations et vous devez répondre à des questions en vous appuyant sur des informations déduites de ces déclarations. La règle suivante s'applique : « Les orateurs sont de trois natures : les véridiques, les menteurs et les changeants. Les véridiques disent toujours la vérité, les menteurs mentent toujours, et les changeants disent en alternance une vérité et un mensonge (c'est-à-dire, soit une vérité, puis un mensonge, puis une vérité, etc. ; soit un mensonge, puis une vérité, puis un mensonge, etc.). Pendant tout le concours, les orateurs ne peuvent pas changer de nature. »

Les épreuves comportent deux phases :

- Les différents orateurs font plusieurs déclarations dont l'analyse permet de déterminer la nature de chaque orateur (véridique, menteur, changeant commençant par dire la vérité, ou changeant commençant par dire un mensonge).
- Les orateurs font une seconde série de déclarations. Puis, vous devez répondre à des questions en exploitant les informations contenues dans ces déclarations.

Question 1.1 : Dans la première phase, quel est le nombre minimum de déclarations que doit faire chaque orateur pour qu'il soit possible de déterminer sa nature ? Justifier.

Question 1.2 : Soit un orateur A qui fait une suite de n déclarations A_i. Proposer des formules du calcul des propositions A_V, A_M, A_{CV} et A_{CM} qui permettent de caractériser la nature de A (respectivement véridique, menteur, changeant commençant par dire la vérité, ou changeant commençant par dire un mensonge).

Vous participez à une première épreuve avec un orateur A qui fait les déclarations suivantes :

- J'aime le rouge mais pas le bleu.
- Soit j'aime le rouge, soit j'aime le vert.
- Si j'aime le rouge et le vert, alors j'aime le bleu.

Nous noterons R, V et B les variables propositionnelles associées au fait que l'orateur aime le rouge, le vert ou le bleu. Nous noterons A_1 , A_2 et A_3 les formules propositionnelles associées aux déclarations de A.

Question 1.3 : Représenter les déclarations de l'orateur sous la forme de formules du calcul des propositions A_1, A_2 et A_3 dépendant des variables R, V et B.

Question 1.4 : Appliquer les formules permettant de caractériser la nature des orateurs proposées pour la question 2.2 pour l'orateur A dépendant des variables A_1, A_2 et A_3.

Question 1.5 : En utilisant le calcul des propositions (résolution avec les tables de vérité, par exemple), déterminer la nature de l'orateur A. Quelles sont les couleurs qu'aime A ?

Vous participez à une seconde épreuve avec trois orateurs G, H et I. Vous avez déterminé dans la première phase avec succès que G est un menteur, que H est un véridique et que I est un changeant sans savoir s'il doit dire la vérité ou un mensonge pour sa déclaration suivante. Ceux-ci font les déclarations :

— I : Le losange est visible
— G : Le cercle n'est visible que si le losange est visible.
— I : Le triangle n'est pas visible.
— H : Soit le cercle est visible, soit le triangle est visible.

Nous noterons G_1, H_1, I_1 et I_2 les formules propositionnelles associées aux déclarations des orateurs dans cette épreuve. Nous noterons C, L et T les variables propositionnelles associées au fait que le cercle, le losange ou le triangle soit visible.

Question 1.6 : Représenter les déclarations des orateurs sous la forme de formules du calcul des propositions G_1, H_1, I_1 et I_2 dépendant des variables C, L et T.

Question 1.7 : Représenter les informations sur la nature des orateurs sous la forme d'une formule du calcul des propositions dépendant des variables G_1, H_1, I_1 et I_2.

Question 1.8 : En utilisant le calcul des propositions (résolution avec les formules de De Morgan, par exemple), déterminer quelle est (ou quelles sont) la (ou les) figure(s) visible(s) ainsi que la nature exacte de l'orateur changeant I.

Partie 2 : Automates

Le but du problème est d'étudier des algorithmes permettant de rechercher efficacement des occurrences d'une ou plusieurs chaînes de caractères (appelées *motifs*) à l'intérieur d'une longue chaîne de caractères.

Définitions générales

On fixe un *alphabet* Σ, c'est-à-dire un ensemble fini, dont les éléments sont appelés *symboles*. On note $\lambda > 0$ le nombre d'éléments de cet alphabet, et on suppose que cette taille λ est disponible depuis les fonctions OCaml à implémenter sous la forme d'une constante globale `lambda`. Par exemple :

```
let lambda = 5 ;;
```

On supposera que les éléments de Σ sont ordonnés, par exemple par ordre alphabétique, et on les note dans l'ordre $\alpha_0 < \cdots < \alpha_{\lambda-1}$. On dit que le *code* d'un symbole α de l'alphabet est l'entier i tel que $\alpha = \alpha_i$ $(0 \leq i \leq \lambda - 1)$.
Une *chaîne de caractères* s est une suite *non vide* $u_1 \dots u_k$ de symboles de Σ. La *longueur* de s est son nombre de symboles, c'est à dire, ici, k. On représentera en OCaml les chaînes de caractères par des listes d'entiers (`int list`), chaque entier étant le code d'un symbole. Ainsi, si l'alphabet est $\Sigma = \{a, b, c\}$, dans cet ordre, la chaîne de caractères "*abac*" sera représentée par la liste `[0;1;0;2]`. En particulier, on n'utilisera jamais le type `string`.

Recherche naïve d'un motif

Une *occurrence* d'une chaîne de caractères $s = s_1 \dots s_k$ (aussi appelée un *motif*) dans une autre chaîne $t = \beta_1 \dots \beta_n$ est un entier naturel y avec $1 \leq y \leq n$ tel que, pour tout entier i tel que $0 \leq i \leq k - 1$, $s_{k-i} = \beta_{y-i}$. En d'autres termes, l'occurrence indique la position du dernier caractère du motif au sein de la chaîne de caractères t (les positions commençant à 1).

Par exemple, si $\Sigma = \{a, b, c, d, e\}$, s est le motif "abc" et t la chaîne de caractères "$abcabcdababcdabcdabde$", il y a quatre occurrences de s dans t à savoir $3, 6, 12$ et 16.

Question 2.1 : Soient s, t deux chaînes de caractères. Est-il possible d'avoir deux occurrences y et y' de s dans t avec $y < y'$ et $y \geq y' - k + 1$. Si oui, donner un exemple ; sinon, le prouver.

Question 2.2 : Quel est le nombre maximal d'occurrences d'un motif de longueur k dans une chaîne de caractères de longueur $n \geq k$? Prouver que cette borne est toujours respectée, et que cette borne est atteinte.

Question 2.3 : Programmer une fonction `longueur : 'a list → int` qui prend en entrée une liste et qui renvoie la longueur de cette liste. Quelle est la complexité de cette fonction, en terme du nombre n d'éléments de la liste ?

Question 2.4 : Programmer une fonction `int list → int list → bool` prenant en entrée deux listes d'entiers s et t codant des chaînes de caractères et testant si s est un préfixe de t. Quelle est la complexité de cette fonction en termes des longueurs k et n de s et t ?

Question 2.5 : À l'aide des fonctions `longueur` et `prefixe`, programmer une fonction `recherche_naive : int list → int list → int list`, la plus directe possible, telle que si s est un motif et t une chaîne de caractères, `recherche_naive s t` renvoie la liste des entiers y qui sont des occurrences de s dans t, triés par ordre croissant.

Question 2.6 : Quelle est la compelxité de la fonction `recherche_naive`, en terme des longueurs k et n de ses deux paramètres s et t ?

Automates finis déterministes à repli

Un *automate fini déterministe à repli* (ou *AFDR*) sur l'alphabet Σ est un quadruplet $\mathcal{A} = (k, F, \delta, \rho)$ tel que :

- $k \in \mathbb{N}$ est un entier strictement positif représentant le *nombre d'états* de $\mathcal{A}$; l'ensemble des *états* de $\mathcal{A}$ est $Q_{\mathcal{A}} = \{0, 1, \ldots, k-1\}$ et 0 est appelé l'*état initial*.
- $F \subset Q_{\mathcal{A}}$ est un ensemble d'états appelés *finals*.
- $\delta : Q_{\mathcal{A}} \times \Sigma \to Q_{\mathcal{A}}$ est une *fonction partielle de transition* (c'est-à-dire, une fonction dont le domaine de définition est un sous-ensemble de $Q_{\mathcal{A}} \times \Sigma$) ; on impose que $\delta(0, \alpha)$ soit défini pour tout $\alpha \in \Sigma$.
- $\rho : Q_{\mathcal{A}} \setminus \{0\} \to Q_{\mathcal{A}}$ est une application (c'est-à-dire une fonction totale), appelée *fonction de repli*, telle que pour tout $q \in Q_{\mathcal{A}}$ avec $q \neq 0$, $\rho(q) < q$. On prolonge ρ en convenant $\rho(0) = 0$.

Un AFDR est représenté en OCaml par le type enregistrement suivant :

```
type afdr={
   final : bool array ;
   transition : int array array ;
   repli : int array } ;;
```

où si `af` est de type `afdr` et représente un AFDR $\mathcal{A} = (k, F, \delta, \rho)$:

- `af.final` est un tableau de taille k de booléens tel que si $q \in Q_{\mathcal{A}}$, `af.final.(q)` contient `true` si et seulement si $q \in F$;
- `af.transition` est un tableau de taille k de tableaux de taille λ d'entiers, tel que si $q \in Q_{\mathcal{A}}$ et $\alpha_i \in \Sigma$ de code i, `af.transition.(q).(i)` contient -1 si $\delta(q, \alpha_i)$ n'est pas défini, et contient $\delta(q, \alpha_i)$ sinon ;
- `af.repli` est un tableau de taille k d'entiers tel que `af.repli.(0)` contient 0 et `af.repli.(q)` pour $q \in Q_{\mathcal{A}} \setminus \{0\}$ contient $\rho(q)$.

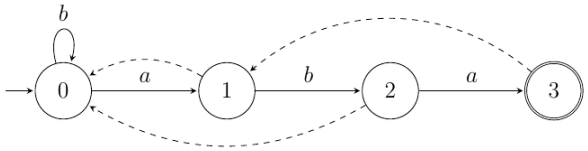
On observe que le type `afdr` ne code pas explicitement la valeur de k, mais k est par exemple la longueur du tableau `af.final`.
On dessine un AFDR de manière similaire au dessin d'un automate fini déterministe classique, en faisant figurer une flèche d'étiquette $*$ pour indiquer les replis d'un état vers un autre. Les états finals sont figurés par un double cercle.

Considérons par exemple l'automate $\mathcal{A}_1 = (k_1, F_1, \delta_1, \rho_1)$ sur l'alphabet $\Sigma = \{a, b\}$ défini par $k_1 = 4$, $F_1 = \{3\}$ et les fonctions δ_1 et ρ_1 suivantes :

	δ_1	
q	$\delta_1(q,a)$	$\delta_1(q,b)$
0	1	0
1		2
2	3	
3		

	ρ_1
q	$\rho_1(q)$
0	
1	0
2	0
3	1

$\mathcal{A}_1$ peut être dessiné comme suit



Question 2.7 : Soit $\mathcal{A} = (k, F, \delta, \rho)$ un AFDR. Pour tout $q \in Q_{\mathcal{A}}$, on note $\rho^j(q)$ pour $j \in \mathbb{N}$ l'application répétée j fois de la fonction ρ à q, c'est-à-dire $\underbrace{\rho(\rho(\ldots(\rho(q))\ldots))}_{j \text{ fois}}$. Par convention, pour tout état q, on note $\rho^0(q) = q$.
Montrer que pour tout $q \in Q_{\mathcal{A}}$, pour tout $\alpha \in \Sigma$, il existe $j \geq 0$ tel que $\delta(\rho^j(q), \alpha)$ est défini.

On dit qu'un AFDR $\mathcal{A} = (k, F, \delta, \rho)$ *accepte* un mot $u = u_1 \ldots u_p \in \Sigma^*$ s'il existe une suite finie

$$q_0, q'_1, q_1, q'_2, q_2 \ldots, q'_{p-1}, q_{p-1}, q'_p, q_p$$

d'états de $\mathcal{A}$ avec :
- $q_0 = 0$;
- pour tout $1 \leq i \leq p$, $q'_i = \rho^j(q_{i-1})$ avec $j \geq 0$ le plus petit entier tel que $\delta(\rho^j(q_{i-1}), u_i)$ est défini ;
- pour tout $1 \leq i \leq p$, $q_i = \delta(q'_i, u_i)$;
- $q_p \in F$.

Le *langage* accepté par un AFDR est l'ensemble des mots acceptés.
Ainsi, $\mathcal{A}_1$ accepte le mot "*ababa*" comme le montre la suite d'états parcourus

$$0, 0, 1, 1, 2, 2, 3, 1, 2, 2, 3$$

On remarque qu'un AFDR dont la fonction de transition δ est définie partout peut être vu comme un automate fini déterministe classique, et que cet automate fini déterministe est *complet* (ce qui signifie précisément que sa fonction de transition est définie partout). En effet, les puissances non nulles de la fonction de repli ρ ne sont utilisées que si la fonction de transition n'est pas définie. On appellera un tel automate fini déterministe complet un AFDC.

Question 2.8 : Construire (sans justification) un AFDC sur l'alphabet $\{a, b\}$ reconnaissant le même langage que $\mathcal{A}_1$ et ayant le même nombre d'états que $\mathcal{A}_1$.

Question 2.9 : Donner (sans justification) une description concise du langage recoonu par l'AFDR $\mathcal{A}_1$.

Question 2.10 : Programmer une fonction `copie_afdr : afdr → afdr` qui renvoie un nouvel AFDR identique à l'AFDR fourni en entrée, mais ne partageant aucune donnée avec lui. On pourra utiliser la fonction standard `Array.copy : 'a array → 'a array` qui renvoie un nouveau tableau contenant les mêmes éléments que les éléments d'entrée et s'exécute en un temps proportionnel au nombre d'éléments du tableau d'entrée.

Question 2.11 : En s'inspirant de la réponse aux questions 7 et 8, et en utilisant la fonction `copie_afdr`, programmer une fonction `enleve_repli : afdr → afdr` telle que si `a` repreprésente un AFDR $\mathcal{A}$, `enleve_repli a` renvoie un AFDC reconnaissant le même langage. On souhaite que cette fonction s'exécute en temps $O(k \times \lambda)$, où k est le nombre d'états de $\mathcal{A}$ et λ la taille de l'alphabet. Montrer que la fonction proposée a bien cette complexité.

Question 2.12 : Étant donné un AFDC $\mathcal{A}$ et un mot $u = u_1 \dots u_n$ sur Σ, proposer un algorithme (pas un programme OCaml) en $O(n)$ pour calculer la liste triée des entiers i avec $1 \leq i \leq n$ tels que le préfixe $u_1 \dots u_i$ de u est accepté par $\mathcal{A}$.

Question 2.13 : On veut implémenter en OCaml l'algorithme de la question précédente.
Programmer une fonction `occurrences : afdr → int list → int list` telle que quand `a` représente un AFDC $\mathcal{A}$ et `liste` est une liste d'entiers $j_1, \dots, j_n$ codant des symboles $\alpha_{j_1}, \dots, \alpha_{j_n}$ de l'alphabet Σ, `occurrences a liste` renvoie la liste des entiers i avec $1 \leq i \leq n$ tels que le mot $\alpha_{j_1}, \dots, \alpha_{j_i}$ est reconnu par $\mathcal{A}$. Quelle est la complexité de cette fonction en terme de la longueur n de la liste d'entiers, du nombre k d'états de l'automate $\mathcal{A}$ et de λ ?

Partie 3 : Le retour de l'énigme

Source : site du CNRS

Ci dessous, une liste d'assertions, certaines fausses, d'autres vraies, qui se réfèrent à un nombre positif entier (qui est écrit en base 10 et ne commence pas par 0). Si une assertion est vraie, son numéro apparaît comme chiffre du nombre à trouver sinon, il n'y apparaît pas.

- 0 : La somme des chiffres du nombre est un nombre premier.
- 1 : Le produit des chiffres du nombre est impair.
- 2 : Chacun des chiffres du nombre est inférieur au chiffre suivant (s'il existe).
- 3 : Aucun chiffre du nombre n'est égal à un autre.
- 4 : Aucun des chiffres du nombre n'est supérieur à quatre.
- 5 : Le nombre a moins de six chiffres.
- 6 : Le produit des chiffres du nombre n'est pas divisible par 6.
- 7 : Le nombre est pair.
- 8 : Aucun chiffre du nombre ne diffère de un d'un autre chiffre du nombre.
- 9 : Au moins un des chiffres du nombre est égal à la somme de deux autres chiffres du nombre. [1]

Quel est ce nombre ?

1. Les trois chiffres en question doivent être des chiffres différents du nombre à trouver. Précisément, un chiffre peut être compté deux fois, mais il faut alors qu'il apparaisse deux fois dans le nombre.