Concours blanc

Option informatique, deuxième année

Julien Reichert

Partie 1 : Langages formels (cours et application)

Question 1.1 : Montrer que si $\mathcal{A}$ et $\mathcal{B}$ sont deux automates, alors il existe un automate reconnaissant l'union des langages des deux automates en question. Donner le principe de la construction d'un tel automate.

On notera que les réponses aux deux questions suivantes dépendent de la construction choisie...

Question 1.2 : Déterminer la complexité de la construction présentée à la question précédente.

Question 1.3 : En considérant toujours la construction précédente, est-il vrai que si $\mathcal{A}$ et $\mathcal{B}$ sont déterministes, alors l'automate obtenu l'est aussi ? Même question pour complet et émondé.

Question 1.4 : Que dire d'un langage qui peut être reconnu par un automate à la fois émondé et complet ?

Partie 2 : Langages formels (exercice)

En plus des notations usuelles du cours, on dit / rappelle qu'un mot f sur Σ, de longueur non nulle, est un facteur d'un mot u sur Σ s'il existe deux mots x et y sur Σ avec $u = xfy$; si x est le mot vide, on précise que f est un préfixe de u, et si y est le mot vide, on dit que f est un suffixe de u.

Un mot peut posséder plusieurs occurrences d'un même facteur f. Supposons que l'on ait $\Sigma = \{a, b\}$ et $f = aabaa$. Le mot *baabaabbaabaaa* contient deux occurrences **disjointes** du facteur f, **de même que le mot** *aabaaaabaaa*. Le mot *abaabaaabaabb* contient deux occurrences non disjointes du facteur f, de même que le mot *aabaabaa* ; dans ce dernier cas, la première occurrence de f est un préfixe du mot *aabaabaa* alors que la seconde occurrence en est un suffixe.

Dans tout l'exercice, Σ désigne un alphabet et f désigne un mot de longueur non nulle sur Σ.

Question 2.1 : Montrer que le langage L_1 sur Σ des mots qui contiennent au moins une occurrence du facteur f est rationnel.

Question 2.2 : Montrer que le langage L_2 sur Σ des mots qui contiennent au moins deux occurrences disjointes du facteur f est rationnel.

Question 2.3 : Montrer que le langage L_3 sur Σ des mots qui contiennent au moins deux occurrences non disjointes du facteur f est rationnel.

Question 2.4 : Montrer que le langage L_4 sur Σ des mots qui contiennent exactement une occurrence du facteur f est rationnel.

Partie 3 : Graphes

On considère le problème concret suivant : des cours doivent avoir lieu dans un intervalle de temps précis (de 8h à 9h55,...) et on cherche à attribuer une salle à chaque cours. On souhaite qu'à tout moment une salle ne puisse être attribuée à deux cours différents et on aimerait utiliser le plus petit nombre de salles possibles. Ce problème d'allocation de ressources (ici les salles) en fonction de besoins fixes (ici les horaires des cours) intervient dans de nombreuses situations très diverses (allocation de pistes d'atterrissage aux avions, répartition de la charge de travail sur plusieurs machines,...).

Le modèle étudié ici s'appelle un graphe d'intervalles.

Représentation du problème

On modélise le problème ainsi :
- chaque besoin est représenté par un segment $[a, b]$ où $a, b \in \mathbb{N}$ et $a \leq b$;
- deux besoins I et J sont en conflit quand $I \cap J \neq \emptyset$.

La donnée du problème est une suite finie $(I_0, \ldots, I_{n-1})$ de n segments où $n \in \mathbb{N} \setminus \{0\}$.

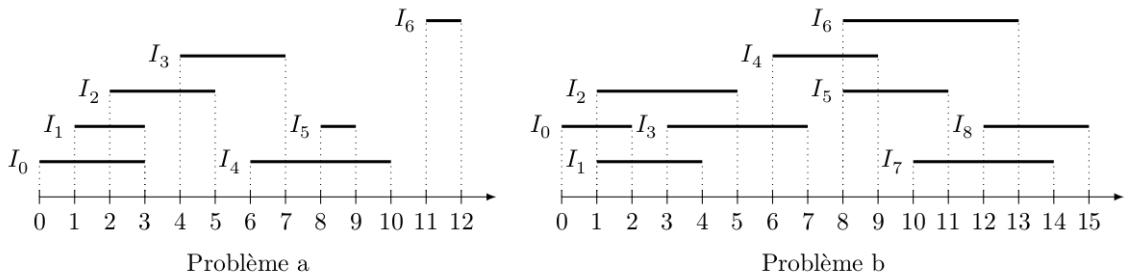


Figure 1 Deux exemples de problèmes

On représente un segment par un couple d'entiers, la donnée du problème est une valeur du type `(int * int) array`. Le problème a de la figure 1 est représenté par le tableau

```
[| (0,3); (1,3); (2,5); (4,7); (6,10); (8,9); (11,12) |].
```

Question 3.1 : Écrire une fonction ayant pour signature `conflit : int * int -> int * int -> bool` telle que `conflit i j` renvoie `true` si et seulement si `i` et `j` sont en conflit.

Graphe simple non orienté

On appelle graphe simple non orienté un couple $G = (S, A)$ où :
- S est un ensemble fini dont les éléments sont appelés les sommets du graphe ;
- A est un ensemble de paires d'éléments distincts de S. Lorsque $\{x, y\} \in A$ on dit que x et y sont reliés dans G et $\{x, y\}$ est appelée une arête de G. Les sommets reliés à un sommet x sont appelés les voisins de x.

Étant donnée une énumération de S sous la forme d'une suite finie $(x_0, \ldots, x_{n-1})$, on représente A en Caml par un élément du type `int list array` ainsi : pour `i` $\in \{0, \ldots, n-1\}$, la liste `A.(i)` contient les `j` tels que x_i soit relié à x_j dans G. On représente graphiquement le graphe G par un diagramme où les arêtes sont représentées par des traits entre les sommets.

Les arêtes du graphe dont une représentation graphique est donnée figure 2 sont représentées en Caml par le tableau :

```
[| [1;2;3]; [0;2;3]; [0;1;3;4]; [0;1;2]; [2] |]
```

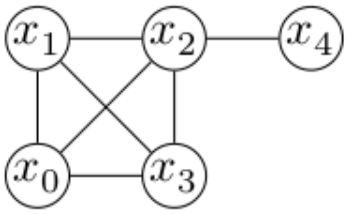


Figure 2

Une telle liste d'arêtes suffit pour déterminer un graphe lorsque l'énumération des sommets est connue car on peut alors identifier un sommet à son indice. Dans la suite de ce problème on identifiera ainsi un graphe à sa liste d'arêtes.

Graphe d'intervalles

Soit $\bar{I} = (I_0, \ldots, I_{n-1})$ une suite finie de segments, on appelle graphe d'intervalles associé à $\bar{I}$ le graphe $G(\bar{I})$

- dont les sommets sont les segments I_0 , …, I_{n-1} ;
- et où, pour $i, j \in \{0, \ldots, n-1\}$, avec $i \neq j$, les sommets I_i et I_j sont reliés si et seulement si ils sont en conflit.

Le graphe d'intervalles qui correspond au problème a de la figure 1 admet la représentation graphique de la figure 3.

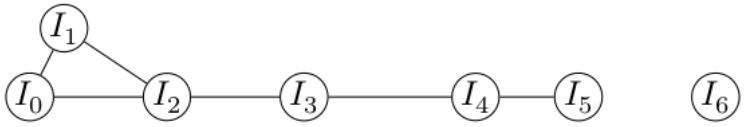


Figure 3

Question 3.2 : Donner une représentation graphique du graphe d'intervalles associé au problème b de la figure 1.

Question 3.3 : Écrire une fonction ayant pour signature `construit_graphe : (int * int) array -> int list array` qui étant donné le tableau des segments $\bar{I} = (I_0, \ldots, I_{n-1})$, énumérés dans cet ordre, renvoie la représentation des arêtes de $G(\bar{I})$.

Coloration

Soit $G = (S, A)$ un graphe simple non orienté dont les sommets sont $x_0, \ldots, x_{n-1}$. On appelle coloration de G une suite finie d'entiers naturels $(c_0, \ldots, c_{n-1})$ telle que

$$\forall i, j \in \{0, \ldots, n-1\}, \{x_i, x_j\} \in A \Rightarrow c_i \neq c_j.$$

L'entier c_i est appelé la couleur du sommet x_i et la condition se traduit ainsi : deux sommets reliés ont des couleurs distinctes. Dorénavant, le terme couleur sera synonyme d'entier naturel.

La suite finie $(0, 1, 2, 3, 0)$ est une coloration du graphe de la figure 2.

Lorsqu'une coloration utilise le plus petit nombre de couleurs distinctes possibles, on dit qu'elle est optimale. On note alors $\chi(G)$ ce nombre minimum de couleurs, appelé le nombre chromatique de G.

En associant une salle à chaque couleur, on peut répondre au problème initial à l'aide d'une coloration de son graphe d'intervalles associé.

Question 3.4 : Déterminer des colorations optimales pour les graphes d'intervalles associés aux deux problèmes de la figure 1. On attribuera à chaque fois la couleur 0 à l'intervalle I_0.

Question 3.5 : Écrire deux fonctions, l'une de signature `appartient : int list -> int -> bool`, telle que l'appel à `appartient l x` envoie `true` si et seulement si l'entier `x` est présent dans la liste `l`, et l'autre de signature `plus_petit_absent : int list -> int`, telle que l'appel à `plus_petit_absent l` renvoie le plus petit entier naturel non présent dans `l`.

Question 3.6 : On considère ici une coloration progressive des sommets d'un graphe. Pour cela, une coloration partielle est un tableau `couleurs : int array` tel que `couleurs.(i)` contient la couleur de `i` s'il est coloré et `-1` sinon, ce qui ne pose pas de problème car les couleurs sont toujours positives.
Écrire une fonction de signature `couleurs_voisins : int list array -> int array -> int -> int list` telle que l'appel à `couleurs_voisins aretes couleurs i` renvoie la liste des couleurs des voisins colorés du sommet d'indice `i` dans le graphe décrit par `aretes` où le tableau couleurs décrit une coloration partielle.

Question 3.7 : En déduire une fonction de signature

`couleur_disponible : int list array -> int array -> int -> int`

telle que l'appel à `couleur_disponible aretes couleurs i` renvoie la plus petite couleur pouvant être attribuée au sommet `i` afin qu'il n'ait la couleur d'aucun de ses voisins dans le graphe décrit par `aretes`.

Cliques

Soit $G = (S, A)$ un graphe. Un sous-ensemble $C \subseteq S$ est appelé une clique de G lorsqu'il vérifie

$$\forall x, y \in C, x \neq y \Rightarrow \{x, y\} \in A.$$

Le nombre d'éléments de C est appelé sa taille. La taille de la plus grande (celle qui possède le plus grand nombre d'éléments) clique de G est notée $\omega(G)$.

Question 3.8 : Déterminer $\chi(G)$ et $\omega(G)$ lorsque G ne possède pas d'arête (c'est à dire $A = \emptyset$), et lorsque G est un graphe complet à n sommets, c'est à dire $|S| = n$ et pour tous $u, v \in S$ distincts, $\{u, v\} \in A$.

Question 3.9 : Comparer $\chi(G)$ et $\omega(G)$ pour un graphe G quelconque.

Question 3.10 : Écrire une fonction de signature `est_clique : int list array -> int list -> bool` telle que `est_clique aretes xs` renvoie `true` si et seulement si la liste `xs` est une liste d'indices de sommets formant une clique dans le graphe décrit par `aretes`.

Algorithme glouton pour la coloration

Étant donnée une liste de segments $\bar{I} = (I_0, I_1, \ldots, I_{n-1})$ de longueur $n \geq 1$, on se propose de déterminer une coloration optimale de son graphe d'intervalles associé. On appelle coloration de $\bar{I}$ une suite finie d'entiers naturels $(c_0, \ldots, c_{n-1})$ telle que

$$\forall i, j \in \{0, \ldots, n-1\}, I_i \cap I_j \neq \emptyset \Rightarrow c_i \neq c_j.$$

On suppose dans cette partie que les segments $I_k = [a_k, b_k]$, pour $k \in \{0, \ldots, n-1\}$, sont énumérés dans l'ordre croissant de leur extrémités gauches, c'est-à-dire que $a_0 \leq a_1 \leq \cdots \leq a_{n-1}$.

On propose l'algorithme suivant :

Pour k variant de 0 à $n-1$, colorer l'intervalle I_k avec la plus petite couleur non encore utilisée dans la coloration des intervalles I_j, avec $0 \leq j < k$, qui ont une intersection non vide avec I_k.

Ainsi, l'intervalle I_0 est toujours coloré avec la couleur 0, l'intervalle I_1 reçoit la couleur 0 si $I_0 \cap I_1 = \emptyset$, et la couleur 1 sinon, etc.

Question 3.11 : Déterminer la coloration renvoyée par l'algorithme pour le problème b décrit sur la figure 1. Si par hasard elle a déjà été donnée à la question 4, on peut se contenter de le signaler tout simplement.

Question 3.12 : Écrire une fonction de signature `coloration : (int * int) array -> int list array -> int array` telle que l'appel `coloration segments aretes`, où `segments` est un tableau contenant des segments triés par ordre croissant de leurs extrémités gauches et où `aretes` représente les arêtes du graphe d'intervalles associé à ces segments, renvoie la coloration obtenue avec l'algorithme ci-dessus.

On se propose maintenant de démontrer que l'algorithme ci-dessus fournit une coloration optimale de l'ensemble de segments. Soit k un entier entre 0 et $n-1$. On suppose qu'à la k-ième étape de l'algorithme, le segment I_k reçoit la couleur c.

Question 3.13 : L'extrémité gauche du segment I_k appartient à un certain nombre de segments parmi $I_0, I_1, \ldots, I_{k-1}$. Combien au moins ?

Question 3.14 : Prouver que l'ensemble constitué de I_k et de ses voisins d'indice inférieur à k constitue une clique de taille au moins $c+1$ dans le graphe d'intervalles associé.

Question 3.15 : En déduire que le nombre de couleurs nécessaires à une coloration de l'ensemble des segments est au moins égal à $c+1$, puis conclure.

Question 3.16 : Déterminer la complexité de la fonction `coloration` en fonction du nombre m d'arêtes du graphe d'intervalles associé à la liste $\bar{I}$.

Partie 4 : Logique

Nous nous intéressons dans cet exercice à l'étude de quelques propriétés de la logique propositionnelle tri-valuée. En plus des deux valeurs classiques Vrai ($\top$) et Faux ($\bot$) que peut prendre une expression, la logique propositionnelle tri-valuée introduit une troisième valeur Indéterminé (?). $\mathcal{V}$ est l'ensemble des variables propositionnelles et $\mathcal{F}$ l'ensemble des formules construites sur $\mathcal{V}$. Pour $A, B \in \mathcal{V}$, les tables de vérité des opérateurs classiques dans cette logique propositionnelle sont les suivantes :

(a) $A \wedge B$

A	B	$A \wedge B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\top$	?	?
$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$
$\bot$	?	$\bot$
?	$\top$	?
?	$\bot$	$\bot$
?	?	?

(b) $A \vee B$

A	B	$A \vee B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\top$	?	$\top$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$
$\bot$	?	?
?	$\top$	$\top$
?	$\bot$	?
?	?	?

(c) $A \Rightarrow B$

A	B	$A \Rightarrow B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\top$	?	?
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$
$\bot$	?	$\top$
?	$\top$	$\top$
?	$\bot$	?
?	?	$\top$

(d) $\neg A$

A	$\neg A$
$\top$	$\bot$
$\bot$	$\top$
?	?

On appelle tri-valuation une fonction $f : \mathcal{V} \to \{\top, \bot, ?\}$, qu'on étend à une fonction $\hat{f}$ définie sur $\mathcal{F}$. Elle satisfait une formule ϕ si $\hat{f}(\phi) = \top$. Une formule ϕ est une conséquence d'un ensemble de formules X si toute interprétation qui satisfait toutes les formules de X satisfait ϕ, et une tautologie si toute interprétation la satisfait. On pose $\top = 1$, $\bot = 0$ et $? = 0,5$.

Question 4.1 : Montrer que $A \vee \neg A$ n'est pas une tautologie.

Question 4.2 : Proposer une tautologie simple.

Question 4.3 : Proposer un calcul simple permettant de trouver la table de vérité de $A \wedge B$ en fonction de A et B. Même question pour $A \vee B$.

Question 4.4 : En logique bi-valuée classique, les propositions $\neg A \vee B$ et $A \Rightarrow B$ sont équivalentes. Qu'en est-il dans le cadre de la logique propositionelle tri-valuée ?

Question 4.5 : En écrivant les tables de vérité, indiquer si les propositions $\neg B \Rightarrow \neg A$ et $A \Rightarrow B$ sont équivalentes.

Question 4.6 : Donner la table de vérité de la proposition $((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)) \Rightarrow B$. Cette proposition est-elle une tautologie ?

Un nouvel opérateur d'implication, noté $\rightarrow$, est alors défini, dont la table de vérité est la suivante :

A	B	$A \rightarrow B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\top$	?	?
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$
$\bot$	?	$\top$
?	$\top$	$\top$
?	$\bot$	?
?	?	?

Question 4.7 : $A \rightarrow A$ est-elle une tautologie ?

Question 4.8 : A-t-on équivalence entre le fait que B soit une conséquence de $\{A\}$ et le fait que $A \rightarrow B$ soit une tautologie ?

On définit un type `formule_logique` représentant les formules de la manière suivante :

```
type formule_logique =
|Vrai (* constante Vrai*)
|Faux (* constante Faux*)
|Indetermine (* constante Indéterminé *)
|Var of string (* variable propositionnelle *)
|Non of formule_logique (* Négation d'une formule *)
|Et of formule_logique * formule_logique (* conjonction de deux formules *)
|Ou of formule_logique * formule_logique (* disjonction de deux formules *)
|Implique of formule_logique * formule_logique (* implication *)
```

Question 4.9 : Avec la représentation précédente, écrire en Ocaml la formule : $((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)) \Rightarrow B$ [1].

1. Au passage, commenter la formule en bon mathématicien.

Partie 5 : Logique aussi

Question 5.1 : Pour chaque formule qui suit, dire si elle est satisfiable ou non, en justifiant [2] :

a) $x_1 \wedge (x_0 \vee \neg x_0) \wedge \neg x_1$
b) $(x_0 \vee \neg x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee \neg x_2)$
c) $x_0 \wedge \neg(x_0 \wedge \neg(x_1 \wedge \neg(x_1 \wedge \neg x_2)))$
d) $(x_0 \vee x_1) \wedge (\neg x_0 \vee x_1) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_0 \vee \neg x_1)$

En machine nous représenterons les formes normales conjonctives (FNC) sous la forme de listes de listes. Plus précisément, une FNC sera une liste de clauses et chaque clause sera une liste de littéraux :

```
type litteral =
|V of int (* variable *)
|NV of int;; (* négation de variable *)
type clause = litteral list;;
type fnc = clause list;;
```

Question 5.2 : Écrire une fonction `var_max` qui prend en entrée une FNC `f` et renvoie le plus grand indice de variable utilisé dans la formule. La complexité de la fonction doit être linéaire en la taille de `f`.

Partie 6 : L'énigme que tout le monde attendait !

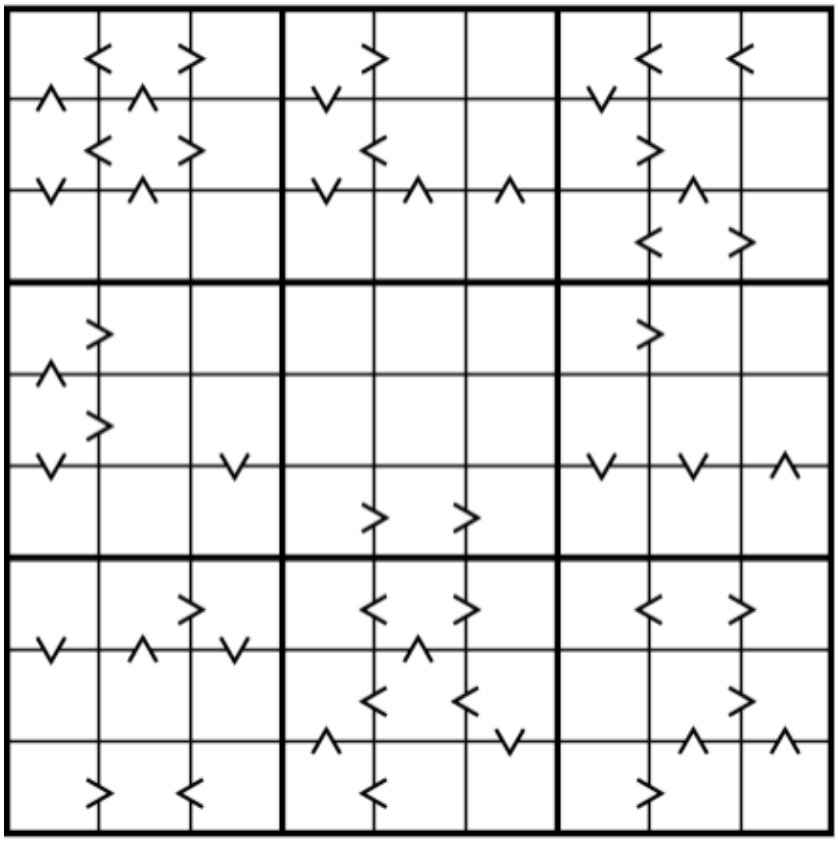
Résoudre les variantes de sudoku ci-après.

Dans la première variante, en plus des contraintes usuelles (faire apparaître une et une seule fois chaque nombre de 1 à 9 dans chaque ligne, chaque colonne et chaque carré délimité), des indices entre deux cases adjacentes indiquent lequel des deux nombres est le plus grand par l'opérateur usuel de comparaison. (La version la plus populaire, mais pas celle que j'ai connue en premier, donne des indices entre chaque couple de cases adjacentes d'un même carré.)

Dans la deuxième variante, en plus des contraintes usuelles, des zones délimitées indiquent quelle doit être la somme des valeurs dans la zone (peut-être cela rappelle-t-il le Kenken aux habitués de mes DS).

Il paraît que sur internet, des sudoku combinant ces deux variantes existent...

2. Dans le sujet original, c'était sans justification...



	1	2	3	4	5	6	7	8	9
A	13		20	22			16		
B		11			11				16
C	19		22		11	18			
D		11				12		12	
E				11					
F	4		2	13	10	13	16	13	
G	23							9	
H	6				11	15		9	
J	12		6				18		