DS 2

Option informatique, deuxième année

Julien Reichert

Exercice 1 : Logique et calcul des propositions

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau. Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons X, Y et Z. Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner. Le premier sujet abordé est la région dans laquelle se trouve le village :

— X indique : « Le village se trouve dans la vallée » ;
— Z réplique : « Non, il ne s'y trouve pas » ;
— X reprend : « Ou alors dans les collines ».

Nous noterons V et C les variables propositionnelles associées à la région dans laquelle se trouve le village. Nous noterons X_1 et Z_1 les formules propositionnelles correspondant aux affirmations de X et de Z sur le premier sujet. Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

— X dit : « Le chemin de gauche conduit au village » ;
— Z répond : « Tu as raison » ;
— X complète : « Le chemin de droite y conduit aussi » ;
— Y affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;
— Z indique : « Celui du milieu n'y conduit pas ».

Nous noterons G, M, D les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village. Nous noterons X_2, Y_2 et Z_2 les formules propositionnelles correspondant aux affirmations de X, de Y et de Z sur le second sujet.

Question 1.1 : Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_1 et Z_1.

Question 1.2 : Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions X_1 et Z_1 dépendant des variables V et C.

Question 1.3 : En utilisant la résolution avec les propriétés des opérateurs booléens et les formules de De Morgan en calcul des propositions, déterminer dans quelle région vous devez vous rendre pour rejoindre le village.

Question 1.4 : Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_2, Y_2 et Z_2.

Question 1.5 : Représenter les informations données par les participants sous la forme de trois formules du calcul

des propositions X_2, Y_2 et Z_2 dépendant des variables G, M et D.

Question 1.6 : En utilisant la résolution avec les tables de vérité en calcul des propositions, déterminer quel chemin vous devez suivre pour rejoindre le village.

Question 1.7 : En admettant que les trois participants aient menti, pouviez-vous prendre d'autres chemins ? Si oui, le ou lesquels ?

Exercice 2 : Structures de données et algorithmes

On suppose défini le type arbre de la manière suivante :

```
type arbre = | Feuille of int | Noeud of arbre * arbre ;;
```

On dit qu'un arbre est un peigne si tous les noeuds à l'exception éventuelle de la racine ont au moins une feuille pour fils. [1] On dit qu'un peigne est strict si sa racine a au moins une feuille pour fils, ou s'il est réduit à une feuille. On dit qu'un peigne est rangé si le fils droit d'un noeud est toujours une feuille. [2] Un arbre réduit à une feuille est un peigne rangé.

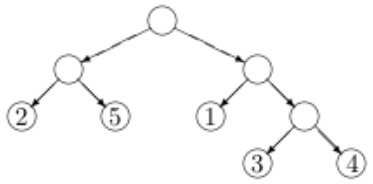

FIGURE 1 – Un peigne à cinq feuilles

Question 2.1 : Représenter un peigne rangé à 5 feuilles.

Question 2.2 : La hauteur d'un arbre est le nombre de noeuds maximal que l'on rencontre pour aller de la racine à une feuille (la hauteur d'une feuille seule est 0). Quelle est la hauteur d'un peigne rangé à n feuilles ? On justifiera la réponse.

Question 2.3 : Écrire une fonction `est_range : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne rangé.

Question 2.4 : Écrire une fonction `est_peigne_strict : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne strict. En déduire une fonction `est_peigne : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne.

On souhaite ranger un peigne donné. Supposons que le fils droit `N` de sa racine ne soit pas une feuille. Notons `A1` le sous-arbre gauche de la racine, `f` l'une des feuilles du noeud `N` et `A2` l'autre sous-arbre du noeud `N`. On va utiliser l'opération de rotation qui construit un nouveau peigne où :

- le fils droit de la racine est le sous-arbre `A2` ;
- le fils gauche de la racine est un noeud de sous-arbre gauche `A1` et de sous-arbre droit la feuille `f`.

Question 2.5 : Donner le résultat d'une rotation sur l'arbre de la figure 1.

Question 2.6 : Écrire une fonction `rotation : arbre -> arbre` qui effectue l'opération décrite ci-dessus. La fonction renverra l'arbre initial si une rotation n'est pas possible.

1. Attention, ce n'est donc pas comme dans le cours.
2. Notion de cours correspondante : peigne gauche.

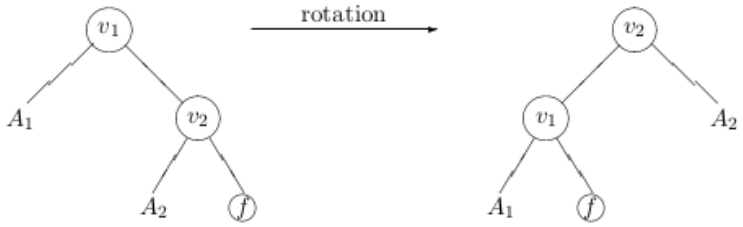

FIGURE 2 – Une rotation

Question 2.7 : Écrire une fonction `rangement : arbre -> arbre` qui range un peigne donné en argument, c'est à dire qu'il renvoie un peigne rangé ayant les mêmes feuilles que celui donné en argument. La fonction renverra l'arbre initial si celui-ci n'est pas un peigne.

Exercice 3 : Graphes

Soit G un graphe non orienté dont les sommets sont les entiers de 1 à 8 représenté par liste d'adjacence ci-dessous.

Sommets	Sommets adjacents
1	(2,3,4)
2	(1,3,4)
3	(1,2,4)
4	(1,2,3,6)
5	(6,7,8)
6	(4,5,7)
7	(5,6,8)
8	(5,7)

FIGURE 3 – Un graphe

On considérera que lors du parcours du graphe les sommets adjacents d'un sommet donné sont rencontrés dans le même ordre qu'ils sont listés dans le tableau précédent.

Question 3.1 : Représenter graphiquement le graphe correspondant à G.

Question 3.2 : Ce graphe est-il complet ? Est-il connexe ?

Question 3.3 : Donner la séquence des sommets de G obtenus lors du parcours en profondeur en commençant sur le sommet 1.

Question 3.4 : Donner la séquence des sommets de G obtenus lors du parcours en largeur en commençant sur le sommet 1.

Exercice 4 : Jeux à un joueur

Un jeu à un joueur est la donnée d'un ensemble non vide E, d'un élément $e_0 \in E$, d'une fonction $s : E \to \mathcal{P}(E)$ et d'un sous-ensemble F de E. L'ensemble E représente les états possibles du jeu. L'élément e_0 est l'état initial. Pour un état e, l'ensemble $s(e)$ représente tous les états atteignables en un coup à partir de e. Enfin, F est l'ensemble des états gagnants du jeu. On dit qu'un état e_p est à la profondeur p s'il existe une séquence finie de $p+1$ états $e_0 e_1 \dots e_p$ avec $e_{i+1} \in s(e_i)$ pour tout $0 \leq i < p$. Si par ailleurs $e_p \in F$, une telle séquence est appelée une solution du jeu, de profondeur p. Une solution optimale est une solution de profondeur minimale. On notera qu'un même état peut être à plusieurs profondeurs différentes.

Voici un exemple de jeu :

$$E = N \setminus \{0\} \; ; e_0 = 1 \; ; s(n) = \{2n, n+1\}$$

Question 4.1 : Donner une solution optimale pour ce jeu lorsque $F = \{42\}$.

Pour chercher une solution optimale pour un jeu quelconque, on peut utiliser un parcours en largeur. Un pseudo-code pour un tel parcours est donné figure 4.

```
BFS()
    A ← {e_0}
    p ← 0
    tant que A ≠ ∅
        B ← ∅
        pour tout x ∈ A
            si x ∈ F alors
                renvoyer VRAI
            B ← s(x) ∪ B
        A ← B
        p ← p + 1
    renvoyer FAUX
```

FIGURE 4 – Parcours en largeur

Question 4.2 : Montrer que le parcours en largeur renvoie `VRAI` si et seulement si une solution existe.

Question 4.3 : On se place dans le cas particulier du jeu de la question 4.1 pour un ensemble F arbitraire pour lequel le parcours en largeur de la figure 4 termine. Montrer alors que la complexité en temps et en espace est exponentielle en la profondeur p de la solution trouvée. On demande de montrer que la complexité est bornée à la fois inférieurement et supérieurement par deux fonctions exponentielles en p.

Dans la suite, on suppose donnés un type `etat` et les valeurs suivantes pour représenter un jeu en Caml :

```
initial: etat
suivants: etat -> etat list
final: etat -> bool
```

Question 4.4 : Écrire une fonction `bfs : unit -> int` qui effectue un parcours en largeur à partir de l'état initial et renvoie la profondeur de la première solution trouvée. Lorsqu'il n'y a pas de solution, le comportement de cette fonction pourra être arbitraire.

Indication : On pourra avantageusement réaliser les ensembles `A` et `B` par des listes, sans chercher à éliminer les doublons, et utiliser une fonction récursive plutôt qu'une boucle.

Question 4.5 : Montrer que la fonction `bfs` renvoie toujours une profondeur optimale lorsqu'une solution existe.

Comme on vient de le montrer, l'algorithme BFS permet de trouver une solution optimale mais il peut consommer

un espace important pour cela, comme illustré dans le cas particulier du jeu de la question 4.1 qui nécessite un espace exponentiel. On peut y remédier en utilisant plutôt un parcours en profondeur. La figure 5 contient le pseudo-code d'une fonction DFS effectuant un parcours en profondeur à partir d'un état `e` de profondeur `p`, sans dépasser une profondeur maximale `m` donnée.

```
DFS(m, e, p)
    si p > m alors
        renvoyer FAUX
    si e ∈ F alors
        renvoyer VRAI
    pour chaque x dans s(e)
        si DFS(m, x, p + 1) = VRAI alors
            renvoyer VRAI
    renvoyer FAUX
```

FIGURE 5 – Parcours en profondeur

Question 4.6 : Montrer que `DFS(m, e_0, 0)` renvoie `VRAI` si et seulement si une solution de profondeur inférieure ou égale à `m` existe.

Pour trouver une solution optimale, une idée simple consiste à effectuer un parcours en profondeur avec `m = 0`, puis avec `m = 1`, etc., jusqu'à ce que `DFS(m, e_0, 0)` renvoie `VRAI`. (Il s'agit d'une recherche itérée en profondeur.)

Question 4.7 : Écrire une fonction `ids : unit -> int` qui effectue une recherche itérée en profondeur et renvoie la profondeur d'une solution optimale. Lorsqu'il n'y a pas de solution, cette fonction ne termine pas.

Question 4.8 : Montrer que la fonction `ids` renvoie toujours une profondeur optimale lorsqu'une solution existe.

Question 4.9 : Comparer les complexités en temps et en espace du parcours en largeur et de la recherche itérée en profondeur dans les deux cas particuliers suivants : (i) il y a exactement un état à chaque profondeur p ; (ii) il y a exactement 2^p états à la profondeur p. On demande de justifier les complexités qui seront données.

Exercice 5 : Énigme de logique et d'arithmétique

[Source : magazine TV hebdomadaire allemand]

La séquence d'opérations ci-après est formée de lettres représentant chacune un seul et même chiffre, sans que deux lettres différentes ne représentent le même chiffre.

Le principe est simple : remplacer chaque lettre par le chiffre correspondant, en détaillant au maximum le raisonnement.

$$\begin{array}{cccccccc} A & B & \times & C & = & D & E & C \\ & - & & + & & & & - \\ E & F & - & D & = & & C & A \\ \hline E & B & \times & G & = & D & F & G \end{array}$$