Correction du DM 1

Option informatique, deuxième année

Julien Reichert

Ce sujet reprend le sujet CCP 2015, plus précisément la première partie et la deuxième moitié de la troisième partie, qui a été adaptée aux notations du cours.

Partie 1

Exercice 1

La formule attendue est (au changement d'ordre et de notation près)

$$(A_1 \wedge \neg A_2 \wedge \neg A_3) \vee (\neg A_1 \wedge A_2 \wedge \neg A_3) \vee (\neg A_1 \wedge \neg A_2 \wedge A_3).$$

Exercice 2

Les formules attendues sont (même remarque) $A_1 = G \wedge \neg L$, $A_2 = G \Rightarrow \neg L$ (ou la formule équivalente $\neg G \vee \neg L$) et $A_3 = \neg L$.

Exercice 3

Plusieurs méthodes conviennent ici, y compris partir de la réponse à la première question après avoir remplacé les formules, ce qui a l'inconvénient de ne pas donner directement la formule vraie parmi A_1, A_2 et A_3, mais du reste l'énoncé ne l'avait pas demandé.

Les tables de vérité, relativement simples et fiables, notamment en décomposant autant que nécessaire les formules, sont un choix à privilégier, et elles ont la bienveillance du correcteur qui lira plus facilement la copie.

Autrement, un raisonnement plus verbeux peut être envisagé, comme suit :

Les propositions A_1 et A_3 impliquent la proposition A_2. Il est donc impossible que celles-ci soient vraies, car A_2 ne pourrait alors pas être fausse. Ainsi, A_1 et A_3 sont fausses et A_2 est vraie. En particulier, on a L (en tant que $\neg A_3$) et d'après A_2 on a aussi nécessairement $\neg G$.

Il faut donc consommer des lipides mais pas de glucides.

Exercice 4

La formulation « que si » ne doit pas prêter à confusion, la formule A_1 se transcrit directement $S \Rightarrow R$, soit $\neg S \vee R$ ou encore $\neg R \Rightarrow \neg S$. La formule A_2 est plus simple, il s'agit de $\neg I \Rightarrow \neg R$, soit encore $I \vee \neg R$. Quant à la troisième,

son interprétation est assez libre, elle peut aller de $R \vee I \vee S$ à $R \vee ?$ où ? peut être I ou S, voire n'importe quoi d'autre, y compris faire de l'aquaponey. En pratique, il faudra composer avec cette incertitude dans la question suivante, en espérant trouver des contradictions dans le reste des propositions, puis comprendre quelle était la seule possibilité de fin de message pertinente.

Exercice 5

La méthode peut être la même que pour la question 3.

Commençons par remarquer qu'au moins une proposition parmi A_1 et A_2 est vraie, puisque R implique A_1 et $\neg R$ implique A_2. En particulier, A_3 est fausse, donc on ne peut pas avoir R, et c'est A_2 qui est vraie.

On en déduit que A_2 n'a plus rien à nous apprendre, mais aussi que S est vrai puisque $\neg A_1$ équivaut à $S \wedge \neg R$.

Enfin, A_3 ne peut pas se compléter en « ... sportives », car elle serait alors vraie. Donc elle se complète en « ... intellectuelles », et puisque A_3 est fausse, le résumé de la situation est $\neg R \wedge S \wedge \neg I$.

Ceci étant, pour sauver l'honneur, on peut trouver une formulation de A_3 qui autorise les activités intellectuelles, en écrivant par exemple « ... aquatiques avec des équidés » (ce qui n'est pas forcément du sport) ou « ... sportives mais pas intellectuelles », donc $\neg A_3$ devient $\neg R \wedge (\neg S \vee I)$, vrai avec les hypothèses de départ dès lors qu'on a I.

Partie 2

Exercice 6

```
let rec lire_liste = fun
| _ [] -> failwith "Liste trop courte"
| 1 (a::q) -> a
| n (a::q) -> lire_liste (n-1) q;;
```

Toute autre syntaxe peut être acceptée, mais le filtrage sur les deux arguments à la fois est plus agréable ici.

Exercice 7

Dans l'ordre de gauche à droite, les arbres s'écrivent :

```
Noeud(Noeud(Vide,3,Vide),6,Noeud(Noeud(Vide,7,Noeud(Vide,8,Vide)),9,Vide))

Noeud(Noeud(Noeud(Vide,5,Vide),11,Noeud(Vide,8,Vide)),14,Noeud(Noeud(Vide,9,Vide),13,Noeud(Vide,4,Vide)))

Noeud(Noeud(Noeud(Vide,14,Vide),5,Noeud(Vide,8,Vide)),11,Noeud(Noeud(Vide,13,Vide),9,Vide))

Noeud(Noeud(Noeud(Vide,1,Vide),3,Vide),4,Noeud(Vide,2,Vide))
```

Exercice 8

La hauteur de l'arbre vide est 0 d'après la convention de l'énoncé. La hauteur de tout autre arbre binaire est un de plus que la hauteur maximale de ses deux fils.

Exercice 9

Un arbre binaire de hauteur maximale a la propriété que tous ses nœuds sauf le plus profond ont un seul fils. C'est notamment le cas des peignes gauches et droits.

Un arbre binaire de hauteur minimale a la propriété que tous ses nœuds ont deux fils sauf à un ou deux niveaux de profondeur, ces niveaux étant les deux plus profonds existants. En d'autres termes il s'agit d'un arbre binaire complet auquel on ajoute des nœuds sur un seul niveau de profondeur.

Exercice 10

Cela saute aux yeux que les files étaient nécessaires, à force.

```
let creer_file () = ([],[]);;

let est_file_vide (lp,la) = lp = [] && la = [];;

let taille (lp,la) = list_length lp + list_length la;;

let enfiler (lp,la) e = (e::lp,la);;

let rec transfert = function
|([],la) -> ([],la)
|((e::lp),la) -> transfert(lp,e::la);;

let defiler (lp,la) =
    if est_file_vide (lp,la)
    then invalid_arg "File vide"
    else if la <> []
    then (hd la,(lp,tl la))
    else let (lp2,la2) = transfert (lp,la)
    in (hd la2,(lp2,tl la2));;

let lire i arbre =
  let rec lirebis f n = let a, reste = defiler f in
    match (n, a) with
    | n, Vide -> lirebis reste n
    | 1, Noeud(_,x,_) -> x
    | ii, Noeud(g,_,d) -> lirebis (enfiler (enfiler reste g) d) (ii-1)
  in lirebis (enfiler (creer_file ()) arbre) i;;
```

Exercice 11

Aucun piège dans la question : un tas a pour enfants des tas et il suffit de vérifier la propriété pour les racines.

```
let racine = function
| Vide -> failwith "Arbre vide"
| Noeud(_,n,_) -> n;;

let rec est_tas = function
| Vide -> true
| Noeud(Vide,n,Vide) -> true
```

```
| Noeud(g,n,Vide) -> n >= racine g && est_tas g
| Noeud(Vide,n,d) -> n >= racine d && est_tas d
| Noeud(g,n,d) -> n >= racine g && n >= racine d && est_tas g && est_tas d;;
```

Exercice 12

Ce serait une injure que de faire une correction détaillée de la preuve par récurrence que le niveau k d'un arbre binaire complet est constitué de 2^k nœuds.

Attention, l'énoncé demande aussi de calculer le nombre total de nœuds, et donc la taille d'un arbre binaire complet de hauteur h est $2^h - 1$.

Exercice 13

Tout comme pour la vérification de la structure de tas, mais en moins intuitif, il s'agit de trouver une propriété héréditaire qui caractérise les arbres binaires complets ou en ordre militaire.

En pratique, un arbre complet est une racine avec deux arbres complets de même hauteur comme fils. Quant à l'ordre militaire, il est vérifié si, et seulement si :

- Le fils gauche et le fils droit sont complets, le dernier ayant une hauteur de un de moins que le premier ;
- OU le fils gauche est complet et le fils droit en ordre militaire, tous deux de même hauteur ;
- OU le fils gauche est en ordre militaire et le fils droit est complet, de hauteur un de moins (cas pouvant fusionner avec le premier).

Le programme est donc le suivant :

```
let complet_ou_OM arbre =
  let rec aux = function
  | Vide -> (2, 0) (* 2 : complet, 1 : ordre militaire, 0 : rien ; l'autre information est la hauteur *)
  | Noeud(g,_,d) -> let (rep1,h1) = aux g and (rep2,h2) = aux d in
    if rep1 = 2 && rep2 = 2 && h1 = h2 then (2, h1+1)
    else if rep1 >= 1 && rep2 = 2 && h1 = h2+1 then (1, h1+1)
    else if rep1 = 2 && rep2 = 1 && h1 = h2 then (1, h1+1)
    else (0, 42)
  in let reponses = [|"L'arbre n'est pas complet ni même en ordre militaire.";
"L'arbre n'est pas complet mais en ordre militaire.";"L'arbre est complet."|]
  in let (rep,_) = aux arbre in reponses.(rep);;
```

Exercice 14

En s'appuyant sur la question 12, on peut montrer par récurrence que le premier nœud du niveau de profondeur p porte le numéro 2^p, et donc le i-ième nœud du niveau de profondeur p porte le numéro $2^p + i - 1$.

Il s'agit en pratique de trouver la plus grande puissance parfaite de 2 inférieure à n, si on ne dispose pas de l'information de l'énoncé. En attendant, la réponse ici est $n - 2^p$ (car le nœud de numéro n est de numéro $n - (2^p - 1)$ relativement au niveau).

Exercice 15

Le nombre de nœuds à la gauche des fils du nœud numéro n est le double du nombre de nœuds à la gauche dudit nœud, d'après la propriété que l'énoncé donne à l'arbre. On a donc $2n - 2^{p+1}$.

Exercice 16

D'après ce qui précède, un nœud à la profondeur $p + 1$ ayant $2n - 2^{p+1}$ nœuds à sa gauche dans le même niveau porte le numéro $2^{p+1} + 2n - 2^{p+1}$, soit $2n$. Ceci est donc le fils gauche du nœud n, et le fils droit est alors $2n + 1$. Cette propriété est essentielle pour l'implémentation des arbres binaires en ordre militaire dans des tableaux, au passage.

Exercice 17

D'après l'énoncé, il s'agit de ne plus faire de parcours en largeur, mais de descendre directement dans la bonne branche. Pour savoir s'il faut aller à gauche ou à droite, il faut regarder chaque bit de l'écriture du numéro en binaire.

```
let rec directions accu = function
| 1 -> accu
| i -> directions (i mod 2)::accu (i / 2);;

let lirebis i a = let l = directions i in
  let rec aux liste = function
  | Vide -> failwith "Arbre trop court"
  | Noeud(_,n,_) when liste = [] -> n
  | Noeud(g,_,_) when hd liste = 0 -> aux (tl liste) g
  | Noeud(_,_,d) -> aux (tl liste) d
  in aux l a;;
```

Exercice 18

```
exception Non;;

let TBOM arbre = (* fusion des deux fonctions *)
  try
    let rec aux maximum = function
    | Vide -> (2, 0)
    | Noeud(Vide,_,Vide) -> (2, 1)
    | Noeud(g,n,d) -> if n > maximum then raise Non;
      let (rep1,h1) = aux n g and (rep2,h2) = aux n d in
      if rep1 = 2 && rep2 = 2 && h1 = h2 then (2, h1+1)
      else if rep1 >= 1 && rep2 = 2 && h1 = h2+1 then (1, h1+1)
      else if rep1 = 2 && rep2 = 1 && h1 = h2 then (1, h1+1)
      else raise Non
    in let _ = aux max_int arbre in true
  with Non -> false;;
```