Correction du DS 6

Julien Reichert

Questions de cours

Question de cours 1 :
Le symbole `>` a pour utilité principale de rediriger la sortie d'un programme dans un fichier, par exemple la commande `./programme > resultat.txt` exécute le programme en question et imprime tout ce qu'il était censé imprimer dans un fichier créé (ou effacé) pour l'occasion au lieu de la console.

Le symbole `|` redirige la sortie d'un programme dans un autre, par exemple `ps -A | grep firefox` affiche les lignes mentionnant un processus ayant `"firefox"` dans son nom parmi ceux récupérés par la commande `ps`.

Question de cours 2 :
Le propriétaire du fichier peut le lire, le modifier et l'exécuter. Sinon, un membre du groupe propriértaire peut le lire et l'exécuter. Un autre utilisateur peut simplement le lire.

Question de cours 3 :
Soient n un entier naturel non nul, P un ensemble de variables propositionnelles et φ, ψ deux formules ayant exactement les éléments de P comme variables communes. Les deux propriétés suivantes sont équivalentes :

- La formule $(\varphi \Rightarrow \psi)$ est une tautologie.
- Il existe au moins une formule θ, ne contenant aucune variable propositionnelle en dehors de P, appelée interpolante entre φ et ψ, et telle que les formules $\varphi \Rightarrow \theta$ et $\theta \Rightarrow \psi$ soient des tautologies.

Démonstration du sens direct (le sens réciproque étant trivial) : On procède par récurrence sur le nombre, noté k, de variables propositionnelles apparaissant dans φ en-dehors des n variables partagées avec ψ.

Si $k = 0$, alors on pose $\theta = \varphi$ et c'est trivial. Pour passer de k à $k+1$, on considère une formule φ dont une des $k+1$ variables non partagées est notée x. Alors $\varphi[x/\top] \cup \varphi[x/\bot]$, notée φ', n'a plus que k variables non partagées. Si $\varphi \Rightarrow \psi$ est une tautologie, alors $\varphi[x/\top] \Rightarrow \psi$ et $\varphi[x/\bot] \Rightarrow \psi$ en sont aussi (considérer pour chaque interprétation quelle implication elle concerne), d'où par une règle de déduction vue en cours $\varphi' \Rightarrow \psi$ aussi. On pose θ l'interpolante pour φ' (garantie par hypothèse de récurrence). Cette interpolante peut alors être utilisée pour φ (le même raisonnement dans l'autre sens prouve que $\varphi \Rightarrow \theta$ est une tautologie).

Question de cours 4 :
Une formule est en forme normale conjonctive si elle s'écrit comme une conjonction de disjonctions de littéraux, ces derniers étant des variables propositionnelles précédées ou non d'un symbole de négation. Par exemple, la formule $(a \vee b \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (\neg b \vee \neg c \vee d)$ est en forme normale conjonctive.

Question de cours 5 :
D'après le théorème sur la forme normale conjonctive, puis les lois de De Morgan, le système $(\neg, \wedge)$ est complet. Or la formule $\varphi \sim \varphi$ est équivalente à $\neg\varphi$, et par suite la formule $(\varphi \sim \psi) \sim (\varphi \sim \psi)$ est équivalente à $\varphi \wedge \psi$, donc, puisqu'on peut produire avec l'opérateur $\sim$ la conjonction et la négation, le système $(\sim)$ est complet.

Question de cours 6 :
On a $(\varphi \Rightarrow \psi) \Rightarrow \varphi$ implique φ, sachant que la réciproque est triviale. En guise de preuve : par contraposée, si φ est fausse, l'implication ne peut être vraie que si l'hypothèse est fausse. Or cette hypothèse est une implication dont l'hypothèse est fausse, ce qui est impossible. Ainsi, la formule de gauche ne peut pas être vraie si φ est fausse, CQFD.

Question de cours 7 :
Une clé primaire est une clé d'une table, c'est-à-dire un attribut ou un n-uplet d'attributs, pour lequel il n'y a pas deux enregistrements de la table qui ont la même valeur ou le même n-uplet de valeurs. Le fait qu'une clé soit une clé primaire est un choix conduisant à une optimisation des opérations, mais il n'y a pas de propriété supplémentaire.

Une clé étrangère est un attribut (éventuellement un n-uplet si on veut...) d'une table qui correspond à une clé (a priori primaire) d'une autre table, en vue de faire des jointures.

Par exemple, dans la base de données classique pour la gestion des notes, le couple `(Etudiant, Examen)` est une clé (primaire par principe) de la table `Notes`, constituée de deux clés étrangères vers les tables idoines.

Question de cours 8 :
Le résultat de `COUNT` ne peut différer en fonction des attributs qu'en présence de `DISTINCT`, ce qui n'est pas le cas ici, ou si l'un des attributs dans la parenthèse prend la valeur `NULL` pour certains enregistrements alors que l'autre non (et `COUNT(*)` est censé être le cardinal quoi qu'il arrive).

Question de cours 9 :
- Première requête : erreur de syntaxe (`WHERE MAX(valeur)` aurait un sens qui n'est pas celui qui est attendu, mais alors il n'y a plus qu'un agrégat et `id` n'est pas autorisé).
- Deuxième requête : idem.
- Troisième requête : idem et ce serait la même chose avec un `GROUP BY` et un `HAVING` : **on ne compare jamais un attribut et le résultat d'une fonction d'agrégation sur lui-même**.
- Quatrième requête : la syntaxe est correcte mais on ne reçoit qu'un résultat quoi qu'il arrive, même en cas d'égalités.

Version correcte : `SELECT id FROM tbl WHERE valeur = (SELECT MAX(valeur) FROM tbl);`

Question de cours 10 : [Une lettre a été retirée sur la fin pour forcer à calculer jusqu'au bout !]

Code lu	Caractères imprimés	Ajout au dictionnaire
78	"N"	256 : "NE"
69	"E"	257 : "EV"
86	"V"	258 : "VE"
69	"E"	259 : "ER"
82	"R"	260 : "R "
32	" "	261 : " G"
71	"G"	262 : "GO"
79	"O"	263 : "ON"
78	"N"	264 : "NN"
78	"N"	265 : "NA"
65	"A"	266 : "A "
271	" G"	267 : " GI"
73	"I"	268 : "IV"
258	"VE"	269 : "VE "
32	" "	270 : " Y"
89	"Y"	271 : "YO"
79	"O"	272 : "OU"
85	"U"	273 : "U "
32	" "	274 : " U"
85	"U"	275 : "UP"
80	"P"	276 : "P "
32	" "	277 : " N"
256	"NE"	278 : "NEV"
258	"VE"	279 : "VER"
260	"R "	280 : "R G"
262	"GO"	281 : "GON"
264	"NN"	282 : "NNA"
266	"A "	283 : "A L"
76	"L"	284 : "LE"
69	"E"	285 : "ET"
84	"T"	286 : "T "
270	" Y"	287 : " YO"
272	"OU"	288 : "OU "
32	" "	289 : " D"
68	"D"	290 : "DO"
263	"ON"	(plus rien)

Question de cours 11 :

Les constructeurs `Vide` n'apparaîtront pas. On commence par l'arbre vide.

FIGURE 1 – Ajout de `0`

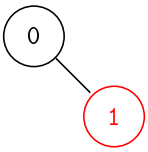

FIGURE 2 – Ajout de `1`

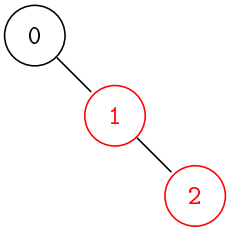

FIGURE 3 – Ajout de `2` [étape intermédiaire]

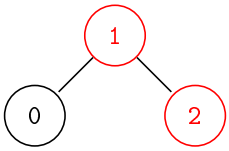

FIGURE 4 – Rotation gauche en `0`

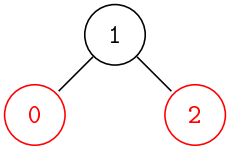

FIGURE 5 – Filtrage sur le résultat de la rotation

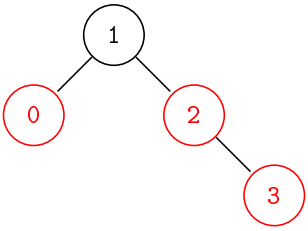

FIGURE 6 – Ajout de `3`

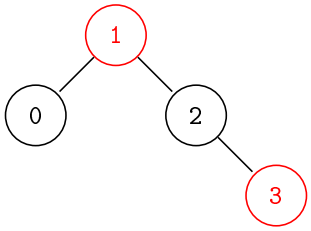

FIGURE 7 – Pas de correction sur 2, correction à la racine (changement de couleur)

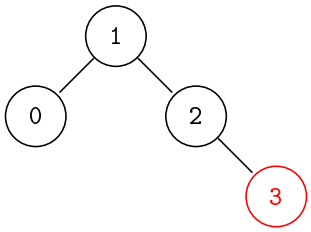

FIGURE 8 – Noircissement de la racine

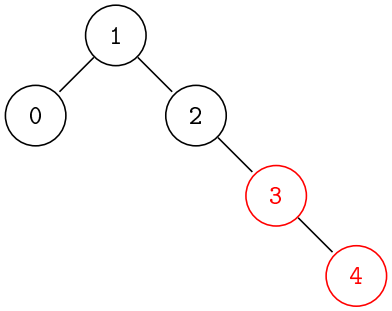

FIGURE 9 – Ajout de 4

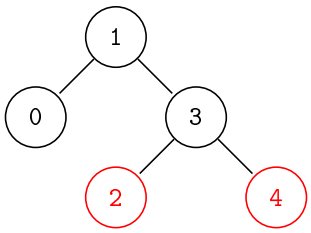

FIGURE 10 – Rotation gauche en 2 puis filtrage du résultat (cf. avant)

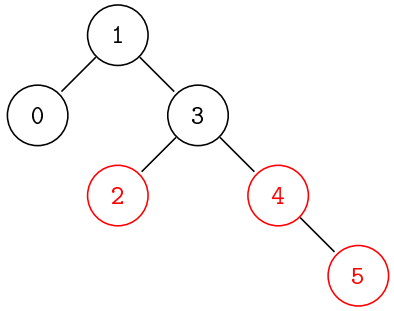

FIGURE 11 – Insertion de 5 (la racine n'a pas eu besoin de correction)

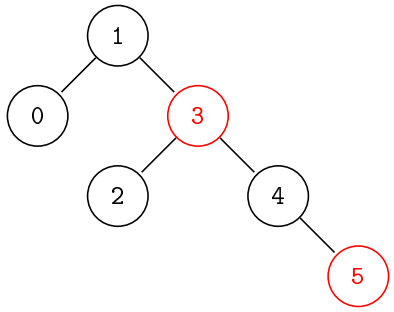

FIGURE 12 – Changement de couleur en 3

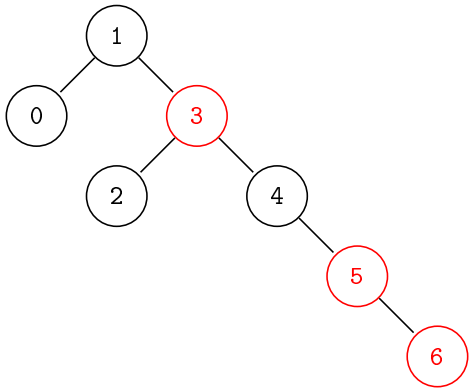

FIGURE 13 – Insertion de 6

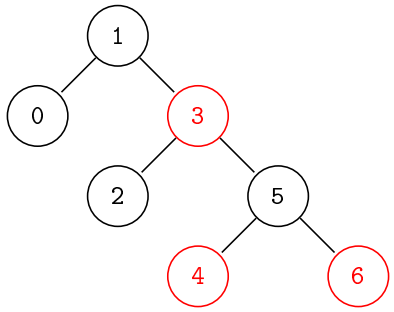

FIGURE 14 – Rotation gauche en 4 puis filtrage du résultat

On observe que l'arbre n'est pas forcément complet, malgré le fait que sa taille l'eût permis.

Exercices

Exercice 1 :

```
let ajoute dico cle valeur = match Hashtbl.find_opt dico cle with
| None -> Hashtbl.add dico cle valeur
| Some n -> Hashtbl.replace dico cle (valeur + n);;

let rec insertion (c, v) l = match l with
| [] -> [(c, v)]
| (c2, v2)::_ when v2 < v -> (c, v)::l
| cpl::q -> cpl::(insertion (c, v) q);;

let liste_scores couples =
  let dico_nombre = Hashtbl.create 8 in
  let dico_score = Hashtbl.create 8 in
  let rec aux l = match l with
  | [] -> ()
  | (qui, quoi)::q -> aux q; ajoute dico_nombre quoi 1;
  ajoute dico_score qui (Hashtbl.find dico_nombre quoi)
  in aux couples;
  Hashtbl.fold (fun cle valeur accu -> insertion (cle, valeur) accu) dico_score [];;
```

Exercice 2 :

```
cat texte.txt | grep j > ../j.txt
```

Exercice 3 :

Exceptionnellement, on fera une récursion sur un tableau. C'est lourd...

```
let rec determinant matrice =
  let n = Array.length matrice in
  if n = 1
  then matrice.(0).(0)
  else
    begin
      let rep = ref 0. in
      for i = 0 to n-1 do
        let coeff = matrice.(i).(0) *. (-.1. ** (float_of_int (i mod 2))) in
        let sousmatrice = Array.make_matrix (n-1) (n-1) 0. in
        for ligne = 0 to n-2 do
          let ligne2 = if ligne < i then ligne else ligne + 1 in
          for colonne = 0 to n-2 do
            sousmatrice.(ligne).(colonne) <- matrice.(ligne2).(colonne+1)
          done
        done;
        rep := !rep +. coeff *. determinant sousmatrice
      done;
      !rep
    end;;
```

Méthode plus simple : faire un pivot jusqu'à se ramener à une matrice triangulaire.

Si un pivot est nul, le résultat est nul, sinon on fait le produit des éléments diagonaux, en compensant les éventuelles dilatations, et on change le signe pour chaque échange de lignes qui a été effectué.

Exercice 4 :

Preuve de la formule : toute fonction induit une surjection vers son image. Si l'image est de cardinal i, alors il y a $\binom{k}{i}$ choix pour ses éléments, et donc $\binom{k}{i}u_i^{(n)}$ applications d'image un ensemble de taille i. En sommant sur toutes les valeurs de i possibles ($u_0^{(n)}$ est nul sauf si $n = 0$), on en déduit l'égalité.

```
let rec puiss a b = if b = 0 then 1 else a * puiss a (b-1);;

let rec coeff_bin n k = if n = 0 then 0 else if k = 0 then 1 else n * (coeff_bin (n-1) (k-1)) / k;;

let nombre_surjections n k =
  if n < k then 0
  else if n = 0 then 1
  else if k = 0 then 0
  else let reponses = Array.make (k+1) 0 in
    reponses.(1) <- 1;
    for kbis = 2 to k do
      reponses.(kbis) <- puiss kbis n;
      for i = 1 to kbis-1 do
        reponses.(kbis) <- reponses.(kbis) - coeff_bin kbis i * reponses.(i)
      done
    done; reponses.(k);;
```

Exercice 5 :

La fonction aux1 construit une liste correspondant à un parcours depuis le sommet en argument, sauf s'il a déjà été rencontré (« marqué ») auquel cas la liste est vide, sachant que le parcours marque tous les sommets rencontrés.

On remarque que l'ajout des sommets rencontrés dans la liste retournée se fait dans l'ordre postfixe, et la fonction principale correspond alors d'après le cours à la construction d'un tri topologique.

Pour un graphe muni d'un cycle, donc qui n'admet pas de tri topologique, le comportement est indéterminé.

Problème 1

Question P1.1 :

Si on cherche une clé primaire, il faut permettre à un client de répondre à toutes les questions d'une séance, mais aussi de participer à plusieurs séances, dont on numérotera toujours les questions de `1` à `40`, et bien entendu il faut permettre à plusieurs clients de participer à une séance.

Ceci étant, une réponse ne peut être donnée qu'une fois, ce qui impose pour la table `REPONSES` la clé primaire `(Id_seance, Id_client, Question)`, et de manière analogue pour la table `SOLUTIONS` la clé primaire `(Id_seance, Question)`.

Question P1.2 :

Pour donner les solutions aux questions d'une séance, on peut les affecter à un identifiant de client fictif (par exemple `-1`) et donc se servir de la table `REPONSES` uniquement.

Question P1.3 :

Dans l'ordre...

- `SELECT COUNT(*) FROM CLIENTS;`
- `SELECT DISTINCT Date FROM SEANCES`
  ou `SELECT Date FROM SEANCES GROUP BY Date;`
- `SELECT AVG(NbFautes) FROM SEANCES WHERE Id_client = n;`
- `SELECT MIN(NbFautes) FROM SEANCES;`
- `SELECT MAX(NbQuestions) FROM`
  `(SELECT COUNT(*) AS NbQuestions FROM SOLUTIONS GROUP BY Reponses) AS tablederivee;`
- `SELECT COUNT(*) FROM REPONSES AS R JOIN SOLUTIONS AS S`
  `ON R.Id_seance = S.Id_seance AND R.Question = S.Question`
  `WHERE Id_client = n AND R.Id_seance = s AND R.Reponses <> S.Reponses;`

Question P1.4 :

Le mieux est de construire les requêtes étape par étape.

On récupère donc les dates des trois dernières séances pour un client particulier d'identifiant noté `n` :

```
SELECT Date FROM SEANCES WHERE Id_client = n ORDER BY Date DESC LIMIT 3
```

Ensuite, pour les nombres de fautes associés, il suffit de voir si le maximum est inférieur ou égal à cinq, en vérifiant aussi qu'il y a au moins trois séances, donc on injecte :

```
SELECT COUNT(*), MAX(NbFautes) FROM
(SELECT * FROM SEANCES WHERE Id_client = n ORDER BY Date DESC LIMIT 3) AS tablederivee
```

À présent, on passe à la jointure pour se servir du nom :

```
SELECT COUNT(*) = 3 AND MAX(NbFautes) <= 5 AS Pret
FROM
(
  SELECT * FROM SEANCES
  JOIN CLIENTS ON Id = Id_client
  WHERE NomPrenom = nom
  ORDER BY Date DESC LIMIT 3
) AS tablederivee
```

En pratique, pour obtenir la liste des tels clients, le mieux est de faire une boucle dans un langage externe grâce auquel on peut faire des requêtes, et on peut même faire le traitement avec ce langage pour ne pas avoir à écrire des requêtes trop compliquées (suivant le degré de maîtrise dans les deux langages...).

Question P1.5 :

Pour obtenir le nombre de séances au total, on peut par exemple écrire

```
SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS
```

Pour filtrer les clients ayant effectué toutes les séances, on fait un regroupement et on utilise `HAVING` :

```
SELECT Id_client
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
```

De même, pour obtenir le nombre minimal de fautes au total parmi de tels clients, on récupère le nombre total de fautes pour chacun de ces clients :

```
SELECT SUM(NbFautes)
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
```

On obtient une table dérivée dont on recherche le minimum, qui doit être le nombre total de fautes du client sélectionné :

```
SELECT Id_client
FROM SEANCES
GROUP BY Id_client
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
AND SUM(NbFautes) =
(
  SELECT MIN(Total)
  FROM
  (
    SELECT SUM(NbFautes) AS Total
    FROM SEANCES
    GROUP BY Id_client
    HAVING COUNT(*) = (SELECT COUNT(DISTINCT Id_seance) FROM SOLUTIONS)
  ) AS tablederivee
)
```

Ici, on n'a pas profité de la possibilité de supposer l'existence et l'unicité, par ailleurs. Sinon c'est plus facile, au moins.

Énigmes

Énigme 1 :

$$\begin{array}{cccccccc} 8 & 1 & \times & 3 & = & 2 & 4 & 3 \\ & - & & + & & & & - \\ 4 & 0 & - & 2 & = & & 3 & 8 \\ \hline 4 & 1 & \times & 5 & = & 2 & 0 & 5 \end{array}$$

Éléments de raisonnement (dans l'ordre chronologique), en assimilant chiffres et nombres par abus et pour simplifier les formulaltions :

- $F = 0$
- $C + 1 = E$
- D est inférieur à E et A en raison des produits sur la première et la dernière ligne
- $2E = A$
- ... donc E vaut au moins 2 (deux valeurs inférieures hors 0 qui est pris) et au plus 4 (moitié de A)
- puisque la multiplication par B ne change pas le dernier chiffre dans deux cas, B, qui ne peut pas être 0 car c'est déjà pris, vaut 1 ou 6, or dans ce dernier cas, en plus de A et F qui sont déjà connus pour être pairs, on aurait G et C qui devraient l'être (les impairs multipliés par 6 changent de dernier chiffre), de même que leur différence qui est D, ce qui en fait un de trop. Donc $B = 1$.
- au passage, A étant pair, C et G sont de même parité, et D est pair pour la raison susmentionnée
- E ne peut désormais plus valoir 2 car il y a encore deux valeurs inférieures mais 1 est désormais pris, et de toute manière la troisième ligne donne une multiplication impossible (seule possibilité : $21 \times 5 = 105$ qui ne colle pas avec les lettres), et E ne peut pas valoir 3 non plus, car 31 n'a aucun facteur ayant un 0 au milieu dans la zone des possibilités, donc $E = 4$
- le reste se déduit directement par le calcul

Énigme 2 :

	15	15	28	??	13	13	3
18	10	2	■	11	4	3	■
4	2	■	4	■	11	10	3
0	3	11	10	4	2	■	■
??	■	4	3	10	■	11	2
11	4	■	11	■	3	2	10
12	11	3	■	2	10	■	4
15	■	10	2	3	■	4	11

Éléments de raisonnement :

- 28 s'obtient avec tout sauf 2, qui est sur la première ou dernière colonne, et les cases noircies sont extrémales sinon
- 18 s'obtient uniquement par $11 + 4 + 3$, au vu des deux cases noircies autour, le 2 mentionné précédemment ne peut pas être sur la première ligne, on fait alors les déductions qui s'imposent
- 12 s'obtient uniquement par $10 + 2$, d'où la localisation de l'autre cellule noircie de l'avant-dernière ligne
- 13 s'obtient uniquement par la somme de deux valeurs, d'où la localisation de l'autre cellule noircie de l'avant-dernière colonne (même raisonnement pour le 3 et la dernière colonne où il est d'ailleurs plaçable)
- dans la deuxième ligne, il n'y a plus qu'une possibilité de placer deux cases noircies avec un 4 entre
- parmi les deux possibilités pour le 15 de la deuxième colonne, il ne reste que la version avec deux chiffres $(11 + 4)$, et on place alors le 11 de la cinquième ligne et les dernières cases noircies
- le découpage du 15 de la première colonne est aussi $11 + 4$, avec le 11 forcément en bas (et le 3 à sa droite)
- des déductions plus faciles permettent de terminer...

Problème 2

Question P2.1 :

- $x_0 = a \wedge b \wedge c \wedge d \wedge e \wedge f \wedge \neg g$
- $x_1 = \neg a \wedge b \wedge c \wedge \neg d \wedge \neg e \wedge \neg f \wedge \neg g$
- $x_2 = a \wedge b \wedge \neg c \wedge d \wedge e \wedge \neg f \wedge g$
- $x_3 = a \wedge b \wedge c \wedge d \wedge \neg e \wedge \neg f \wedge g$
- $x_4 = \neg a \wedge b \wedge c \wedge \neg d \wedge \neg e \wedge f \wedge g$
- $x_5 = a \wedge \neg b \wedge c \wedge d \wedge \neg e \wedge f \wedge g$
- $x_6 = a \wedge \neg b \wedge c \wedge d \wedge e \wedge f \wedge g$
- $x_7 = a \wedge b \wedge c \wedge \neg d \wedge \neg e \wedge f \wedge \neg g$
- $x_8 = a \wedge b \wedge c \wedge d \wedge e \wedge f \wedge g$
- $x_9 = a \wedge b \wedge c \wedge d \wedge \neg e \wedge f \wedge g$

Question P2.2 :

- Code de 0 : 126
- Code de 1 : 48
- Code de 2 : 109
- Code de 3 : 121
- Code de 4 : 51
- Code de 5 : 91
- Code de 6 : 95
- Code de 7 : 114
- Code de 8 : 127
- Code de 9 : 123

Question P2.3 :

Un et un seul chiffre est affiché à la fois, d'où

$$\bigvee_{i=0}^{9} v_i \wedge \bigwedge_{\substack{j \in [\![0,9]\!], \\ j \neq i}} \neg v_j$$

et on notera Φ cette formule par la suite.

Question P2.4 :

- $v_3 : \neg b_8 \wedge \neg b_4 \wedge b_2 \wedge b_1$
- $v_6 : \neg b_8 \wedge b_4 \wedge b_2 \wedge \neg b_1$
- $v_9 : b_8 \wedge \neg b_4 \wedge \neg b_2 \wedge b_1$

Question P2.5 :

$a = (v_0 \vee v_2 \vee v_3 \vee v_5 \vee v_6 \vee v_7 \vee v_8 \vee v_9) \wedge \Phi$

... soit à partir des autres variables et en simplifiant (exemple de réponse, d'autres peuvent convenir) :

$a = (b_8 \vee b_4 \wedge (b_1 \vee b_2) \vee b_2 \vee \neg b_1) \wedge \tilde{\Phi}$

où $\tilde{\Phi}$ est une formule équivalente à Φ et utilisant les autres variables, en pratique cette formule se résume à

$\tilde{\Phi} = \neg b_8 \vee \neg b_4 \wedge \neg b_2$

(en gros, la « somme » au sens intuitif est inférieure à dix).

Question P2.6 :

De la même façon, on a :

- $b = (b_8 \vee \neg b_4 \vee (b_2 \wedge b_1) \vee (\neg b_2 \wedge \neg b_1)) \wedge \tilde{\Phi}$
- $c = (b_8 \vee b_4 \vee \neg b_2 \vee b_1) \wedge \tilde{\Phi}$
- $d = (b_8 \vee b_4 \wedge b_2 \wedge b_1 \vee b_4 \wedge \neg b_2 \wedge \neg b_1 \vee \neg b_4 \wedge (b_2 \vee \neg b_1)) \wedge \tilde{\Phi}$
- $e = ((\neg b_4 \vee b_2) \wedge \neg b_1) \wedge \tilde{\Phi}$
- $f = (b_8 \vee b_4 \vee \neg b_2 \wedge \neg b_1) \wedge \tilde{\Phi}$
- $g = (b_8 \vee b_4 \wedge (\neg b_2 \vee \neg b_1) \vee \neg b_4 \wedge b_2) \wedge \tilde{\Phi}$

Question P2.7 :

```
let tableaux =
[|
  [|true; true; true; true; true; true; false|];
  [|false; true; true; false;false ;false ;false |];
  [|true; true; false; true; true; false; true|];
  [|true; true; true; true; false; false; true|];
  [|false; true; true; false; false; true; true|];
  [|true; false; true; true; false; true; true|];
  [|true; false; true; true; true; true; true|];
  [|true; true; true; false; false; true; false|];
  [|true; true; true; true; true; true; true|];
  [|true; true; true; true; false; true; true|];
|]

let differences tableau =
  let cpt = Array.make 10 0 in
  for indice = 0 to 6 do
    for chiffre = 0 to 9 do
      if tableau.(indice) <> tableaux.(chiffre).(indice) then cpt.(chiffre) <- cpt.(chiffre) + 1
    done
  done;
  let rec minimisation chf rep valrep =
    if chf = 10 then rep
    else if cpt.(chf) < valrep then minimisation (chf+1) [chf] cpt.(chf)
    else if cpt.(chf) = valrep then minimisation (chf+1) (chf::rep) valrep
    else minimisation (chf+1) rep valrep
  in minimisation 0 [] 8;;
```

Question P2.8 :

Léger changement :

```
if chf = 10 then rep
```

devient

```
if chf = 10 then (assert (valrep = 1); List.length rep = 1)
```