Problème 1 : Résolution du sudoku par retour sur trace

On propose un type énumération pour représenter les cases d'une grille de sudoku :

```
type contenu = Vide | Pleine of int
```

On représentera une grille de sudoku par le type `sudoku = contenu array array`.

Question 1.
```
let init l =
    let sudo = Array.make_matrix 9 9 Vide in
    let rec remplir l = match l with
        | [] -> sudo
        | (i,j,v)::q -> sudo.(i).(j) <- Pleine v
    in remplir l;;
    ;;
```

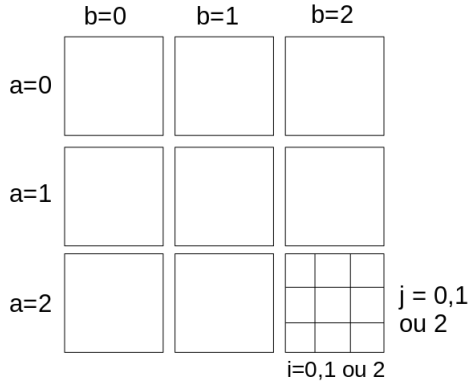
On suppose donnée une fonction `imprime : sudoku -> unit` qui affiche une grille proprement à l'écran.

Question 2. On représente le problème du sudoku par 81 variables $v_{i,j}$ pour $i,j \in [|0,8|]^2$. Ces variables ont toute le même domaine $\mathcal{D} = [|1,9|]$.

On a deux types de contraintes :

- Celles qui découlent des règles du sudoku. On a d'abord sur les lignes $\forall (i,j,k) \in [|0,9|]^3$ $v_{i,j} \neq v_{i,k}$. Pour les colonnes on a similairement $\forall (i,j,k) \in [|0,9|]^3$ $v_{j,i} \neq v_{k,i}$.

  Pour les carrés de 3*3 on a $\forall a,b \in \{0,3,6\}^2, \forall i,j,k,l \in [|0,2|]^4$, $v_{a+i,b+j} \neq v_{a+k,j+l}$. La figure suivante explique ce que représentent a, b, i (ou k) et j (ou l).



- Celles qui découlent de la grille originale. C'est à dire qu'on a un ensemble de cases $\mathcal{I} \subset [|0,8|]^2$ qui ont une valeur initiale. C'est à dire qu'on a pour tout $(i,j) \in \mathcal{I}$, une contrainte du type $v_{i,j} = constante$, la constante étant différente selon la grille.

Question 3. On vérifie que n n'apparait pas déjà dans la ligne i ou dans la colonne j. On récupère les valeurs de a et b pour savoir dans quel carré 3*3 la case se situe et on parcourt toutes les cases de ce carré, c'est à dire toutes les cases de la forme $(a+k', b+l')$, pour $k,l \in [|0,2|]$.

Dans cette version on peut avoir un problème si $g.(i).(j)$ vaut déjà n. On programmera la suite de sorte à ce que ça n'arrive jamais.

```
let possible g i j n =
  let res = ref True in
  for k=0 to 8 do (*On vérifie les lignes et les colonnes*)
      if g.(i).(k) = Pleine n then res:= False;
      if g.(k).(j) = Pleine n then res:= False
  done;
  let a = i/3*3 and b = j/3*3 and k = i mod 3 and l = j mod 3 in
  (*a et b définis comme dans la question précédente*)
  for k' = 0 to 3 do
      for l' = 0 to 3 do
          if g.(a+k').(b+l') = Pleine n then res:= False
      done;
  done; !res;;
```

Question 4. Si on a une grille remplie jusqu'à la case (i, j) qui n'est pas la dernière, alors la prochaine case est soit $(i+1, 0)$ si $j = 8$, soit $(i, j+1)$. Il est intéressant de noter que la dernière case est $(8, 8)$.

Pour trouver des solutions partielles pour cette nouvelle case, on essaie toutes les valeurs de 1 à 9 en testant s'il est licite de les placer dans cette case. Si oui on a une solution partielle (et on peut continuer la recherche par retour sur trace). Sinon on abandonne la recherche dans le sous-arbre.

Question 5.

```
exception Solution;;

let resoudre grille =
  let suivant i j =
      if j=8 then (i+1,0) (*Si i=8 et j=8 on renvoie (9,0)*)
      else (i,j+1)
  in
  let rec aux i j =
      if i = 9 then raise Solution
      if g.(i).(j) = Vide then (*On ne touche pas aux contraintes de départ*)
          for v = 1 to 9 do (*On teste toutes les valeurs*)
              if possible grille i j v then begin g.(i).(j)<- Pleine v; (*On met v*)
                                                aux (suivant i j);
                                                g.(i).(j)<-Vide end; (*On nettoie*)
          done;
  in
  try aux 0 0 with Solution -> imprime g;;
```

Problème 2 : Logique triléenne

Tiré de l'épreuve informatique option MP CCINP 2016.

Question 6. Soit $A \in \mathcal{V}$ et f une tri-valuation définie sur $\mathcal{V}$ telle que $f(A) =?$, alors $\hat{f}(\neg A) =?$ et d'après la table de $\vee$, $\hat{f}(A \vee \neg A) =? \neq \top$. D'où $\not\models_3 (A \vee \neg A)$.

Question 7. En regardant la table de vérité de $\implies$, on réalise que $A \implies A$ est une tautologie.

Posons $\top = 1, \bot = 0$ et $? = 0,5$.

Question 8. On peut proposer pour tout couple $(A, B) \in \mathcal{V}^2$ et toute tri-valuation f : $\hat{f}(A \wedge B) = \min(f(A), f(B))$ et $\hat{f}(A \vee B) = \max(f(A), f(B))$

Question 9. Il est clair en observant la table de vérité de $A \vee B$ que les propositions $A \vee B$ et $B \vee A$ sont équivalentes. Par suite, si on avait $\neg A \vee B$ équivalente à $A \Rightarrow B$ on aurait (par transitivité et spécialisation) équivalence entre les propositions $A \vee \neg A$ et $A \Rightarrow A$. Mais on a vu en question **1** que $\not\models_3 (A \vee \neg A)$ alors que dans la question **7** on a établi que $\models_3 (A \Rightarrow A)$. En conclusion, les propositions $\neg A \vee B$ et $A \Rightarrow B$ ne sont pas équivalentes en logique tri-valuée.

Question 10. Comme demandé, on construit la table de vérité conjointe des deux propositions pour les comparer :

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

On en déduit que : Les propositions $\neg B \Rightarrow \neg A$ et $A \Rightarrow B$ sont équivalentes.

Question 11.

Soit $\psi = ((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)) \Rightarrow B$. On en construit la table de vérité :

A	B	$A \Rightarrow B$	$\neg A \Rightarrow B$	$((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)$	B	ψ
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\top$	?	?	$\top$	?	?	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	?	$\top$	?	?	?	$\top$
?	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
?	$\bot$	?	?	?	$\bot$	?
?	?	$\top$	$\top$	$\top$	?	?

On constate en observant les deux dernières lignes que ψ n'est pas une tautologie.

Question 12. Pour une valuation qui à A associe ?, $A \to A$ prend pour valeur ?. Donc cette formule n'est pas une tautologie (contrairement à $A \implies A$).

Question 13. Dans cette question il s'agit de montrer que la logique triléenne construite avec les opérateurs $\neg, \wedge, \vee, \to$ et des variables n'admet pas de tautologie, dont on note $\mathcal{F}$ l'ensemble des formules. Remarque : contrairement à la logique propositionelle vue en cours $\top$ ne fait pas partie des constructeurs (sinon ce serait une tautologie gratuite).

On note q la tri-valuation de $\mathcal{V}$ dans $\{\top, \bot, ?\}$ définie par : $\forall A \in \mathcal{V}, q(A) =?$. Il suffit d'établir que : $\forall \phi \in \mathcal{F}, \hat{q}(\phi) =?$.

Montrons cela par induction structurelle. Soit $\phi \in \mathcal{F}$.

- si $\phi \in \mathcal{V}$ alors $\hat{q}(\phi) = q(\phi) =?$
- supposons $\phi = \neg\phi_0$ et $\hat{q}(\phi_0) =?$. La table de vérité de l'opérateur $\neg$ montre que dans ce cas $\hat{q}(\phi) = \hat{q}(\neg\phi_0) =?$
- supposons $\phi = \phi_1 \diamondsuit \phi_2$, où ϕ_1, ϕ_2 vérifient l'hypothèse d'induction et $\diamondsuit$ représente l'un quelconque des opérateurs binaires $\wedge, \vee$ ou $\to$. En remarquant qu'une tri-valuation indéterminée à la fois de ϕ_1 et ϕ_2 correspond à la dernière ligne du tableau de vérité de tous ces opérateurs binaires, et que le résultat est dans les trois cas égal à ?, on en déduit que $\hat{q}(\phi) = \hat{q}\,(\phi_1 \diamondsuit \phi_2) =?$

Question 14. Cette question a été recopiée telle qu'elle était dans le sujet de concours. Son énoncé est très bizarre puisque la proposition est exprimée en logique mathématique (booléenne) et pas en logique propositionelle triléenne. Essayons d'en faire ce qu'on peut...

D'après la question 12 on a pas $\Vdash_3 A \to A$. Pourtant on a de manière évidente $\{A\} \Vdash_3 A$. Donc l'implication directe est fausse.

Pour l'implication réciproque, soit A, B des formules trivaluées. On suppose que toute trivaluation q rend $A \to B$ vraie. On a donc pour toute trivaluation q soit $q(A) = \top$ et $q(B) = \top$, soit $q(A) = \bot$ et $q(B)$ quelconque, soit $q(A) =?$ et $q(B) = \top$.

Soit maintenant q une valuation telle que $q(A) = \top$. Alors d'après l'étude précédente on a $q(B) = \top$. Donc on a bien $\{A\} \Vdash_3 B$. L'implication réciproque est vraie.

Question 15.

```
let formule = Implique (Et (Implique (Var "A", Var "B")
,Implique (Non (Var "A"), Var "B")), Var "B");;
```

Question 16. L'énoncé ne précise pas de donner un parenthésage correct, donc on va faire une version simple où on parcourt juste l'arbre de la formule de manière infixe.

```
let affiche formule =
 let rec aux f = match f with
   | Vrai -> "vrai"
   | Faux -> "faux"
   | Indetermine -> "indéterminé"
```

```
    | Var s -> s
    | Non g -> "non " ^ (aux g)
    | Et (g,d) -> (aux g) ^ " et " ^ (aux d)
    | Ou (g,d) -> (aux g) ^ " ou " ^ (aux d)
    | Implique (g,d) -> (aux g) ^ " implique " ^ (aux d)) in
  Printf.printf "%s" (aux formule);;
```

* * *

Problème 3 : Déduction naturelle

Les règles de la déduction naturelle vues en cours sont rappelées à la page suivante.

Question 17. Donner une dérivation en déduction naturelle du séquent : $(p \vee q) \to r \vdash p \to r$.

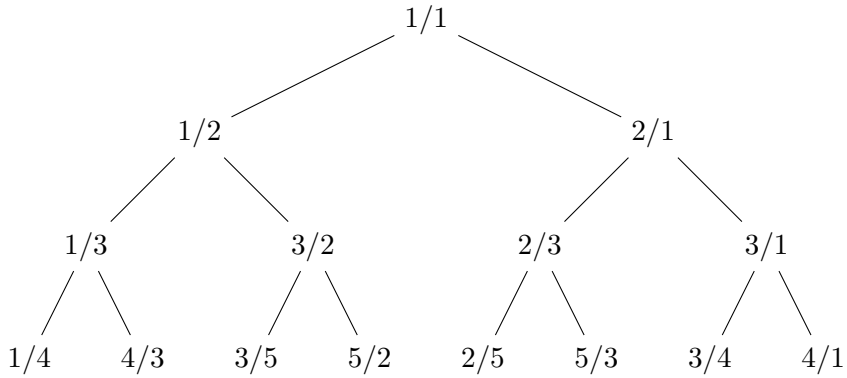
Question 18. Donner une dérivation en déduction naturelle du séquent : $(p \wedge q) \vdash (p \wedge (\neg p \vee q))$

Question 19. Donner une dérivation en déduction naturelle du séquent : $\vdash p \to \neg\neg p$

Question 20. Donner une dérivation en déduction naturelle du séquent : $\vdash \neg\neg p \to p$

Problème 4 : Autour de l'énumération des fractions positives

21. La représentation de l'arbre de Calkin-Wilf jusqu'à la profondeur 3 est :



22. Pour pouvoir représenter des arbres de profondeur finie, on distingue les nœuds des feuilles et on propose le type suivant :

```
type acw = F of fraction | N of acw * fraction * acw ;;
```

23. Montrons par récurrence sur le niveau d'exploration que si n/d est un nœud de l'arbre alors $n \wedge d = 1$.
    - Au niveau 0, la seule fraction représentée est $1/1$ et le résultat est immédiat.
    - $k \in$ et on suppose que tous les nœuds du niveau k sont de la forme n/d avec $n \wedge d = 1$.
      On peut donc considérer un couple de Bézout $(u, v) \in^2$ associé au couple (n, d) :

      $$un + vd = 1$$

      On a alors :

      $$(u - v)n + v(n + d) = 1, \qquad u(n + d) + (v - u)d = 1$$

      Ces deux égalités assurent que $d \wedge n + d = n \wedge (n + d) = 1$. Les couples $(n + d, d)$, $(n, n + d)$ sont donc des couples de nombres entiers premiers entre eux. Ce qui permet de conclure.

24. On s'appuie sur l'algorithme d'Euclide :

```
let rec pgcd m n  =
  if m mod n = 0 then n else pgcd n (m mod n) ;;
```

25. On propose un code qui tient compte de la phrase "Au besoin, elle simplifie la fraction par $n \wedge d$". On pourrait faire la simplification dans tous les cas.

```
let fraction n d =
  if n <= 0 || d <= 0 then failwith "fracirreduc" else
   begin
    let delta = pgcd n d in match delta with
     | 1 -> {n=n;d=d}
     | _ -> {n=n/delta;d=d/delta}
   end;;
```

26. Avec les notations de l'énoncé, on pose :

$$N_1\frac{k}{l}, \quad N_2 = \frac{n}{d}$$

On suppose que les fils gauches sont égaux, alors :

$$\frac{N_1}{N_1+1} = \frac{N_2}{N_2+1}$$

Le résultat de la question 23 assure que les fractions $\frac{N_1}{N_1+1}$ et $\frac{N_2}{N_2+1}$ sont irréductibles.
Par unicité du représentant irréductible d'un nombre rationnel positif, on en déduit que $N_1 = N_2$.
Puisque les fractions $\frac{k}{l}$ et $\frac{n}{d}$ sont irréductibles et égales on en déduit que :

$$k = n, \quad l = d$$

On démontre de la même façon le résultat si les fils droits sont égaux.

27. Le résultat se démontre par récurrence sur $k \in$ en s'appuyant sur le fait que le fils gauche de N est $\frac{N}{1+N}$ (*Les détails ne sont pas donnés dans cette correction.*)
On montre de même que le nœud provenant de d'une suite de k fils droits de N est $N + k$.

28. En s'appuyant sur l'arbre représenté dans la question 21, on obtient :

i	0	1	2	3	4	5	6	7
v_i	0	1	1/2	2	1/3	3/2	2/3	3/1

29. - Pour $n + d = 2$, on a $n = d = 1$, puisque $v_1 = 1$ le résultat est vrai pour $n + d = 2$.
- On considère $k \in \backslash 0, 1$ et on suppose que pour tout $(n, d) \in {}^{*2}$, tel que $n \wedge d = 1$ et $n + d \leq k$, la fraction $\frac{n}{d}$ apparaît dans l'arbre.
alors deux nombres entiers naturels non-nuls n et d tels que $n \wedge d = 1$ et $n + d = k + 1$.

— Si $\frac{n}{d} < 1$, alors n et $d - n$ sont deux nombres entiers naturels non-nuls premiers entre eux et :

$$n + (d - n) \leq k$$

l'hypothèse de récurrence assure alors que la fraction rationnelle $\frac{n}{d-n}$ apparaît dans l'arbre. Or $\frac{n}{d}$ est le fils gauche de ce nœud, la fraction $\frac{n}{d}$ apparaît donc dans l'arbre.

— Si $\frac{n}{d} > 1$, alors $n - d$ et d sont deux nombres entiers naturels non-nuls premiers entre eux et :

$$(n - d) + d \leq k$$

l'hypothèse de récurrence assure alors que la fraction rationnelle $\frac{n-d}{n}$ apparaît dans l'arbre. Or $\frac{n}{d}$ est le fils droit de ce nœud, la fraction $\frac{n}{d}$ apparaît donc dans l'arbre.

On peut donc conclure que pour tout couple de nombres entiers naturels non-nuls (n, d) premiers entre eux, la fraction n/d apparaît dans l'arbre.

30. On va montrer par récurrence que pour tout $k \in$, l'arbre de Calkin-Wilf de profondeur k, noté CW_k ne répète aucun nœud.

- Pour $k = 0$, le résultat est immédiat.
- On considère $k \in$ et on suppose le résultat vrai pour CW_k.
  Supposons que l'arbre CW_{k+1} admet deux nœuds distincts de même valeur N.

  — Si $N < 1$, il s'agit nécessairement de deux fils gauches. Or l'application :
  $$^{+*}x\frac{x}{x+1}$$
  est injective. Les deux nœuds parents sont donc égaux et de profondeur strictement inférieure à k.
  Cela contredit l'hypothèse de récurrence.
  — Si $N > 1$, on conclut de la même façon en raisonnant sur les fils droit et en utilisant le caractère injectif de l'application :
  $$^{+*}xx+1$$

  Pour tout $k \in$, l'arbre CW_k est donc sans répétition, ce qui permet de conclure.

31. Le résultat de la question 29 assure que l'application :
$$^{*+}iv_i$$
est surjective. Le résultat de la question 30 assure qu'elle est injective.
$v_0 = 0$ permet de conclure que la suite $v_{ii\in}$ définit une bijection de dans $^{+}$.

32. On montre classiquement qu'à la profondeur k, il y a 2^k nœuds d'indices variant de 2^k à $2^{k+1} - 1$.
Pour $i \in 2^k, 2^{k+1} - 1$, les 2^k fils gauches sont les nœuds d'indices pairs obtenus en multipliant l'indice du nœud père par 2.
Puisque chaque fils droit est d'indice consécutif de l'indice du fils gauche, on obtient le résultat souhaité.

33. On a :

i	0	1	2	3	4	5	6	7	8	9
s_i	0	1	1	2	1	3	2	3	1	4

34. Le code suivant s'appuie sur la définition récursive de la fonction.

```
let rec stern i = match i with
 | 0                      -> 0
 | 1                      -> 1
 | n  when n mod 2 = 0 -> stern (n/2)
 | n                      -> let i = (n-1)/2 in (stern i) + (stern (i+1));;
```

35. Montrons par récurrence sur la profondeur k des nœuds, que pour tout $i \in$:
$$v_i = \frac{s_i}{s_{i+1}}$$

- Pour $k = 0$, le résultat découle immédiatement de la définition de s_0 et s_1.
- $k \in$ et on suppose le résultat vrai pour k.
  Les nœuds de profondeurs $k + 1$ sont des nœuds fils des nœuds de profondeur k. En notant i l'indice d'un nœud de profondeur k, il s'agit donc de montrer que :
  $$v_{2i} = \frac{s_{2i}}{s_{2i+1}}, \quad v_{2i+1} = \frac{s_{2i+1}}{s_{2i+2}}$$
  Or :
  $$v_{2i} = \frac{v_i}{v_i + 1}$$

l'hypothèse de récurrence assure alors que :

$$v_{2i} = \frac{\frac{s_i}{s_{i+1}}}{\frac{s_i}{s_{i+1}} + 1} = \frac{s_i}{s_i + s_{i+1}} = \frac{s_{2i}}{s_{2i+1}}$$

De même :

$$v_{2i+1} = v_i + 1 = \frac{s_i}{s_{i+1}} + 1 = \frac{s_i + s_{i+1}}{s_i} = \frac{s_{2i+1}}{s_{2i}}$$

ce qui permet de conclure.

36. Dans le premier cas, on peut considérer $j \in^*$ tel que $i = 2j$ et $i + 1 = 2j + 1$.
Le résultat de la question 35 assure alors que :

$$v_i = \frac{s_{2j}}{s_{2j+1}}, \quad v_{i+1} = \frac{s_{2j+1}}{s_{2j+2}}$$

Soit encore :

$$v_i = \frac{s_j}{s_j + s_{j+1}}, \quad v_{i+1} = \frac{s_j + s_{j+1}}{s_{j+1}}$$

Or :

$$\frac{s_j}{s_j + s_{j+1}} = 1 - \underbrace{\frac{s_{j+1}}{s_j + s_{j+1}}}_{\in]0,1[}$$

Donc $\left\lfloor \frac{s_j}{s_j+s_{j+1}} \right\rfloor = 0$ et :

$$f\frac{s_j}{s_j + s_{j+1}} = \frac{1}{1 + 2\left\lfloor \frac{s_j}{s_j+s_{j+1}} \right\rfloor - \frac{s_j}{s_j+s_{j+1}}} = \frac{1}{1 - \frac{s_j}{s_j+s_{j+1}}} = \frac{s_j + s_{j+1}}{s_{j+1}} = v_{i+1}$$

ce qui permet de conclure.

37. Le résultat de la question 27 permet d'établir par une récurrence rapide que la dernier nœud à droite à la profondeur k, correspond à

$$v_{2^{k+1}-1} = k + 1$$

On montre de même que le nœud suivant, qui est le nœud le plus à gauche de la profondeur $k + 1$ vérifie :

$$v_{2^{k+1}} = \frac{1}{k+2}$$

Par ailleurs :

$$fk + 1 = \frac{1}{1 + 2\lfloor k + 1 \rfloor - k - 1} = \frac{1}{k+2}$$

ce qui permet de conclure.

38. Pour construire le chemin, on part de $\frac{n}{d}$ et on remonte jusqu'à la racine.

- Si $\frac{n}{d} < 1$ le parent du nœud est $\frac{n}{d-n}$.
- Si $\frac{n}{d} > 1$ le parent du nœud est $\frac{n-d}{d}$.

On réitère le procédé jusqu'à obtenir la racine.
On peut noter que l'on n'a pas besoin de la fonction `rev` pour obtenir le résultat souhaité.

```
let chemin frac =
 let rec cons res n d = match n,d with
  | 1,1               -> res
  | n,d  when n/d<1 -> cons (G::res) n (d-n)
  | n,d               -> cons (D::res) (n-d) d
 in cons [] frac.n frac.d;;
```

39. La fonction `direction` est construite sur le même principe :

```
let direction liste =
 let rec cons n d liste = match liste with
  | []      -> {n=n;d=d}
  | D::fin  -> cons (n+d) d fin
  | G::fin  -> cons n (n+d) fin
 in cons 1 1 liste;;
```

40. On obtient un ancêtre commun en atteignant le nœud obtenu en parcourant le plus long chemin commun depuis la racine vers les deux nœuds.
Le code de la fonction `ancetre` en découle.
On utilise la fonction auxiliaire `commun` qui détermine le plus long préfixe commun à deux listes.

```
let ancetre f1 f2 =
 let rec commun res l1 l2 = match l1,l2 with
  | [],l2                                -> res
  | l1,[]                                -> res
  | d1::fin1,d2::fin2 when d1 = d2 -> commun (d1::res) fin1 fin2
  | _,_                                  -> res
 in let ch1,ch2 = chemin f1, chemin f2 in
 direction (commun [] ch1 ch2);;
```

41. Puisque v_i et v_{i+1} ont des indices consécutifs, si v_p est le premier ancêtre commun k' niveaux au-dessus des deux nœuds, les chemins de v_p à v_i et de v_p à v_{i+1} sont respectivement de la forme :

$$[\texttt{G}, \texttt{D}, \cdots, \texttt{D}], \quad [\texttt{D}, \texttt{G}, \cdots, \texttt{G}]$$

Où les deux fins de listes sont de longueur $k'-1$ et sont composées respectivement de `D` uniquement, de `G` uniquement.

42. Les résultats des questions 27 et 41 assurent que si $v_p = \dfrac{n}{d}$, alors :

$$v_i = \frac{n}{n+d} + k' - 1, \quad v_{i+1} = \frac{N}{1+(k'-1)N} \text{ où } N = \frac{n+d}{d}$$

Puisque $\dfrac{n}{n+d} \in]0,1[$, on a :

$$\lfloor v_i \rfloor = \left\lfloor \frac{n}{n+d} + k' - 1 \right\rfloor = k' - 1$$

Donc :

$$fv_i = \frac{1}{k' - \frac{n}{n+d}} = \frac{n+d}{k'(n+d) - n}$$

Par ailleurs :

$$\frac{N}{1+(k'-1)N} = \frac{\frac{n+d}{d}}{1+(k'-1)\frac{n+d}{d}} = \frac{n+d}{d+(k'-1)(n+d)} = \frac{n+d}{k'(n+d)-n}$$

ce qui permet de conclure.