Ce devoir est séparé en 4 problèmes indépendants, qui peuvent être traités dans un ordre quelconque. **Attention : ce devoir est à rendre sur deux copies séparées : une copie contenant vos réponses aux problèmes 1 et 2, et une autre copie contenant vos réponses aux problèmes 3 et 4.** Lorsque vous rendrez ces copies, veillez à les mettre dans le tas correspondant. Durée : 4 heures.

Problème 1 : Résolution du sudoku par retour sur trace

On veut écrire un algorithme qui résout le sudoku par retour sur trace.

On rappelle que le sudoku est un puzzle qui se joue dans 9 carrés de 9 cases, alignés pour former un carré de 9×9 cases. Dans chaque case on peut mettre un entier entre 1 et 9 : initialement la majorité des cases sont vides, mais certaines contiennent déjà des valeurs. Le but est de remplir chaque case afin qu'on ait pas 2 fois la même valeur dans la même ligne, colonne ou dans chacun des 9 carrés de 9 cases.

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. Écrire une fonction `init : (int * int * int) list -> sudoku` qui crée une grille à partir d'une liste l de triplets (i, j, v), indiquant que la grille contient initialement la valeur v à la case ligne i, colonne j.

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

Question 2. Exprimer le problème du sudoku comme un problème de satisfaction de contraintes.

Question 3. Écrire une fonction `possible : grille -> int -> int -> int -> bool` telle que `possible g i j n` renvoie un booléen indiquant s'il est licite de placer la valeur n à la ligne i et à la colonne j de la grille g.

Question 4. Étant donné une grille (solution partielle) remplie jusqu'à la case (i, j) qui n'est pas la dernière, comment construire une solution partielle remplie jusqu'à la prochaine case ? On parcourera la grille ligne par ligne, de gauche à droite.

Question 5. Programmer la résolution par retour sur trace de la grille. On définira une exception pour arrêter le programme dès qu'on a trouvé une solution.

```
exception Solution;;

let resoudre grille =
  ...
  let aux ...
    ...
    if ...
    then raise Solution
    ...
  try aux ... with Solution -> imprime g;;
```

Problème 2 : Logique triléenne

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 INDETERMINE (?). Une variable a pour valeur ? si on n'a pas de certitude quant à sa véritable valeur ($\top$ ou $\bot$).

$\mathcal{V}$ 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	$\neg A$
$\top$	$\bot$
$\bot$	$\top$
$?$	$?$

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

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

A	B	$A \implies B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\top$	$?$	$?$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$
$\bot$	$?$	$\top$
$?$	$\top$	$\top$
$?$	$\bot$	$?$
$?$	$?$	$\top$

Definition 1 (Tri-valuation)**.** Une trivaluation est une fonction $f : \mathcal{V} \mapsto \{\top, \bot, ?\}$

On étend alors de manière usuelle la définition de tri-valuation sur l'ensemble des formules :

Definition 2 (Tri-valuation)**.** Une trivaluation sur l'ensemble des formules est une fonction $\hat{f} : \mathcal{V} \mapsto \{\top, \bot, ?\}$

Definition 3. Une tri-valuation $\hat{f}$ satisfait une formule ϕ si $\hat{f}(\phi) = \top$.

Definition 4. Une formule ϕ est :

- une conséquence d'un ensemble de formules $\mathcal{X}$ si toute tri-valuation qui satisfait toutes les formules de $\mathcal{X}$ satisfait ϕ. On notera dans ce cas $\mathcal{X} \Vdash_3 \phi$;
- une tautologie si pour toute tri-valuation $\hat{f}$, $\hat{f}(\phi) = \top$. On notera dans ce cas $\Vdash_3 \phi$.

Question 6. Montrer que pour une formule A, $A \vee \neg A$ n'est pas une tautologie.

Question 7. Proposer alors une tautologie simple dans cette logique.

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

Question 8. Proposer un calcul simple (**utilisant les valeurs numériques définies**) permettant de trouver la table de vérité de $A \wedge B$ en fonction des valeurs de A et B. Même question avec $A \vee B$.

Question 9. En logique bi-valuée classique, $\neg A \vee B$ et $A \implies B$ sont équivalentes. Qu'en est-il dans le cadre de la logique propositionelle tri-valuée ?

Question 10. En écrivant les tables de vérité, indiquer si les propositions $\neg B \implies \neg A$ et $A \implies B$ sont équivalentes en logique trivaluée.

Question 11. Donner la table de vérité de la proposition $((A \implies B) \wedge ((\neg A) \implies B)) \implies B$. 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 12. La formule $A \rightarrow A$ est-elle une tautologie ?

Question 13. Montrer qu'il n'existe aucune tautologie en utilisant uniquement cette définition de l'implication.

Question 14. La proposition suivante est-elle une tautologie :

$$(\{A\} \vDash_3 B) \text{ est équivalent à } (\vDash_3 A \to B) \,?$$

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

```
type FormuleLogique
    |Vrai (* Constante VRAI*)
    |Faux (* Constante FAUX*)
    |Indetermine (* Constante INDETERMINE*)
    |Var of string (* Variable propositionnelle*)
    |Non of FormuleLogique (* Négation d'une formule*)
    |Et of FormuleLogique*FormuleLogique (* conjonction de deux formules*)
    |Ou of FormuleLogique*FormuleLogique (* disjonction de deux formules*)
    |Implique of FormuleLogique*FormuleLogique (* implication*)
```

Question 15. Avec la représentation précédente, écrire en OCaml la formule :

$$((A \implies B) \wedge ((\neg A) \implies B) \implies B$$

Question 16. Écrire alors une fonction récursive Ocaml `lectureFormule`, prenant en argument une formule et renvoyant une chaîne de caractères spécifiant comment un lecteur lirait la formule. Ainsi, par exemple, pour $\phi = A \wedge \neg B$, `lectureFormule` ϕ renvoie `"A et non B"`.

* * *

Attention : à partir d'ici, vous devez rédiger les problèmes qui suivent sur une nouvelle copie !

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 : $\vdash \neg\neg p \to p$

- L'axiome : $\dfrac{}{\Gamma, \varphi \vdash \varphi}$ axiome
- La règle de réduction à l'absurde : $\dfrac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}$ absurde
- La coupure : $\dfrac{\Gamma \vdash \varphi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}$ coupure
- Introduction de la négation : $\dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\neg_i$
- Élimination de la négation : $\dfrac{\Gamma \vdash \neg\varphi \quad \Gamma \vdash \varphi}{\Gamma \vdash \bot}\neg_e$
- Introdution de la conjonction : $\dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\wedge_i$
- Élimination de la conjonction : $\dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\wedge_e^g \quad \dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\wedge_e^d$
- Introduction de la disjonction : $\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\vee_i^g \quad \dfrac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\vee_i^d$
- Élimination de la disjonction : $\dfrac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \theta \quad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}\vee_e$
- Introduction de l'implication : $\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\to_i$
- Élimination de l'implication : $\dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \varphi \to \psi}{\Gamma \vdash \psi}\to_e$
- Introduction de $\top$: $\dfrac{}{\Gamma \vdash \top}\top_i$
- Règle du faux : $\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\bot_e$

Problème 4 : Autour de l'énumération des fractions positives

Ce problème comporte des questions nécessitant un **code OCaml**. Pour ces questions, **les réponses ne feront pas appel aux fonctionnalités impératives du langage** (en particulier pas de boucles, pas de références).

Notations : Dans la suite, on notera :
- $n \wedge d$ le PGCD (Plus Grand Commun Diviseur) de n et d ;
- $\lceil x \rceil$ la partie entière supérieure de x. Ainsi $\lceil 3.14 \rceil = 4$;
- $\lfloor x \rfloor$ la partie entière inférieure de x. Ainsi $\lfloor 3.14 \rfloor = 3$;
- log_2 la fonction logarithme de base 2, fonction réciproque de la fonction $i \mapsto 2^i$.

Une *fraction*, ou *nombre rationnel*, est le quotient de deux nombres entiers, le dénominateur étant par définition non nul. On notera $\mathbb{Q}^+$ l'ensemble des fractions positives.

L'objectif de cette partie est d'étudier une structure de données permettant d'énumérer l'ensemble des fractions positives. Plusieurs travaux se sont intéressés à l'énumération des nombres rationnels, le plus connu étant très certainement la méthode de Cantor. Cependant, il n'est très souvent pas possible, à moins d'un travail assez complexe, de connaître une formule générale donnant le i-ème terme de cette énumération,

ni même de donner le successeur dans l'énumération d'une fraction donnée, Nous proposons dans la suite de répondre à ces questions en construisant un arbre, dit de Calkin-Wilf, permettant de manipuler cette énumération.

Definition 5 (Arbre binaire infini)**.** Un arbre *binaire homogène* est un arbre binaire dont tous les nœuds ont 0 successeur ou 2 successeurs, appelés fils gauche et droit. La hauteur h de l'arbre est la profondeur maximale des nœuds de l'arbre, c'est-à-dire la plus grande longueur d'un chemin de la racine vers une feuille de l'arbre. Lorsque h est infinie, on parle d'*arbre infini*.

Definition 6 (Arbre de Calkin-Wilf)**.** L'*arbre de Calkin-Wilf* est un arbre binaire homogène infini dont les nœuds sont des fractions positives. La racine de l'arbre est la fraction $1/1$. Chaque sommet n/d a deux descendants : un fils gauche $n/(n+d)$ et un fils droit $(n+d)/d$. Ainsi, si $N = n/d$, les deux fils de N sont $N/(N+1)$ (gauche) et $N+1$ (droit).

Question 21. Dessiner l'arbre de Calkin-Wilf jusqu'à une profondeur de 3. Par convention, la racine de l'arbre est au niveau 0.

On propose le type enregistrement :

```
type fraction = { n: int; d: int }
```

pour définir une fraction n/d. Voilà ci-dessous un exemple de déclaration d'une telle fraction :

```
let f = { n=2; d=3 }
```

l'accès au numérateur (respectivement dénominateur) s'opérant alors par `f.n` (resp. `f.d`).

Question 22. Proposer un type OCaml récursif permettant de décrire un arbre de Calkin-Wilf fini.

Question 23. Montrer par récurrence sur le niveau d'exploration de l'arbre que si n/d est un nœud de l'arbre, alors n et d sont premiers entre eux.

Question 24. Écrire une fonction récursive OCaml de signature `pgcd : int -> int -> int` qui calcule le PGCD de deux entiers naturels.

Question 25. Écrire une fonction récursive de signature `fraction : int -> int -> fraction` qui construit une fraction positive irréductible à partir d'un numérateur n et un dénominateur d. La fonction vérifie que $n > 0$ et $d > 0$. Au besoin, elle simplifie la fraction par $n \wedge d$.

Par la suite, on ne construira des valeurs de type fraction que par l'intermédiaire de la fonction `fraction`, ce qui permettra de garantir l'invariant suivant : « pour toute valeur `f: fraction`, les entiers `f.n` et `f.d` sont premiers entre eux ».

Question 26. Soient deux nœuds k/l et n/d de l'arbre ayant des fils gauches (ou droits) identiques. Montrer qu'alors $k = n$ et $l = d$.

Question 27. Soient N un nœud et $k \in \mathbb{N}$. Montrer que le nœud provenant d'une suite de k fils gauches de N est le nœud $N/(kN+1)$. Donner sans démonstration le nœud provenant d'une suite de k fils droits de N.

On définit alors une suite $(v_i)_{i\in\mathbb{N}}$, avec $v_0 = 0$, puis en effectuant un parcours en largeur, de gauche à droite, de l'arbre de Calkin-Wilf pour définir les termes de la suite.

Question 28. Donner les huit premiers termes de la suite $(v_i)_{i\in\mathbb{N}}$.

Question 29. Soient $n, d \in \mathbb{N}^*$ premiers entre eux. Montrer par récurrence sur $n+d$ que toute fraction n/d apparaît dans l'arbre.

Question 30. Montrer qu'aucune fraction n'apparaît deux fois dans l'arbre.

Question 31. Déduire des questions précédentes que la suite $(v_i)_{i\in\mathbb{N}}$ est une bijection de $\mathbb{N}$ dans $\mathbb{Q}^+$.

La suite $(v_i)_{i\in\mathbb{N}}$ permet donc d'affirmer que $\mathbb{Q}^+$ est dénombrable. Nous allons utiliser $(v_i)_{i\in\mathbb{N}}$ pour déterminer, dans l'énumération des fractions produite par cette suite, la position de n'importe quelle fraction positive n/d. En d'autres termes, étant donnée une fraction positive n/d, on se propose de rechercher i tel que $n/d = v_i$.

Question 32. Soit $i \in \mathbb{N}^*$. Montrer que le fils gauche du nœud de valeur v_i a pour valeur v_{2i}. Montrer de même que le fils droit a pour valeur v_{2i+1}.

Pour pouvoir énoncer le i-ème terme dans cette énumération, on introduit alors une suite auxiliaire,

aux nombreuses propriétés arithmétiques et liens avec d'autres objets mathématiques. La suite diatomique de Stern (ou suite de Stern-Brocot) doit son nom à Moritz Stern (1807-1894), élève de Gauss et Achille Brocot (1817-1878), horloger qui s'intéressait aux fractions pour la fabrication d'horloges avec des engrenages comportant peu de dents, donc simples à fabriquer.

Definition 7 (Suite diatomique de Stern ou suite de Stern-Brocot)**.** La suite diatomique de Stern $(s_i)_{i\in\mathbb{N}}$ est définie par $s_0 = 0$, $s_1 = 1$ et pour tout $i \geq 1$: $\begin{cases} s_{2i} & = s_i \\ s_{2i+1} & = s_i + s_{i+1} \end{cases}$.

Question 33. Donner les dix premiers termes de la suite $(s_i)_{i\in\mathbb{N}}$.

Question 34. Écrire une fonction récursive de signature `stern : int -> int` permettant de calculer les termes de cette suite.

Question 35. Déduire par récurrence de la question 12 que : $\forall i \in \mathbb{N}, v_i = \dfrac{s_i}{s_{i+1}}$.

La suite diatomique de Stern permet d'exprimer le i-ième terme dans l'énumération des fractions positives qui est induite par le parcours en largeur de l'arbre de Calkin-Wilf décrit ci-dessus. Voyons maintenant comment obtenir de manière rapide le successeur d'une fraction v_i donnée, sans connaître i, dans cette énumération. Trois cas peuvent se présenter :

- premier cas : les nœuds de valeur v_i et v_{i+1} sont à même profondeur k, fils d'un même nœud N ;
- deuxième cas : le nœud de valeur v_i est le dernier nœud à droite à la profondeur k
- troisième cas : les nœuds de valeur v_i et v_{i+1} sont à même profondeur k, mais ne sont pas fils d'un même nœud.

On pose pour tout $x > 0$, $f(x) = \dfrac{1}{1 + 2\lfloor x \rfloor - x}$.

Question 36. Montrer que dans le premier cas, $v_{i+1} = f(v_i)$.

Question 37. Montrer, en utilisant la question 7, que dans le deuxième cas on a encore $v_{i+1} = f(v_i)$.

On étudie enfin le dernier cas : les nœuds de valeur v_i et v_{i+1} sont sur une même profondeur k, mais ne sont pas les fils d'un même nœud. On va donc passer par la recherche d'un ancêtre commun de ces deux nœuds. Dans la suite, on s'intéressera toujours au premier ancêtre commun, c'est-à-dire celui de profondeur maximale.

En partant de la racine r, il est possible d'atteindre n'importe quel nœud $N = n/d$ de l'arbre par une suite de déplacements vers la gauche (`G`) ou vers la droite (`D`). Le chemin de r vers N peut ainsi être codé par une liste d'éléments de $\{\texttt{G}, \texttt{D}\}$.

Si un déplacement est représenté en OCaml par un type :

```
type direction = G | D
```

alors un chemin est une liste `direction list`.

Question 38. Écrire une fonction OCaml de signature `chemin : fraction -> direction list` qui calcule le chemin de la racine à un nœud quelconque de l'arbre. Cette fonction fera appel à une fonction auxiliaire récursive. Ainsi `chemin n d` calcule la liste des directions à prendre pour passer de `r` au nœud `n/d`. On pourra supposer l'existence d'une fonction `rev : 'a list -> 'a list` qui renverse une liste.

Question 39. Écrire une fonction OCaml de signature `noeud : direction list -> fraction` qui détermine le nœud obtenu en effectuant une série de déplacements depuis la racine. Ainsi `noeud [D;D;G;D]` renverra un couple d'entiers `(n,d)` correspondant au nœud de valeur n/d. Cette fonction fera appel à une fonction auxiliaire récursive.

Question 40. Utiliser les deux fonctions précédentes pour écrire une fonction OCaml ancetre de signature `ancetre : fraction -> fraction -> fraction` qui détermine le premier ancêtre commun entre deux nœuds. Ainsi `ancetre (n,d) (p, q)` détermine le premier ancêtre commun à n/d et p/q. Cette fonction pourra utiliser une fonction auxiliaire récursive.

Question 41. On suppose que le nœud de valeur v_p, le premier ancêtre commun des nœuds de valeur v_i et v_{i+1}, est à k' niveaux au-dessus d'eux. Donner le chemin entre v_p et v_i sous la forme d'une liste de direction. Donner de même le chemin entre v_p et v_{i+1}.

Question 42. À partir de l'expression des fils gauche et droit de v_p, montrer que l'on a encore $v_{i+1} = f(v_i)$