DS 1 : Option info MP/MP*

Consignes à lire attentivement

Ce sujet comporte 10 pages. Durée du devoir : **3 heures**.
L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

Ce sujet est divisé en 3 parties indépendantes, qui peuvent être traitées dans un ordre quelconque. **Vous devez cependant rendre 2 copies différentes : une première copie contenant vos réponses aux partie I et II, et une seconde copie contenant vos réponse aux partie III.** À la fin du devoir, vous prendrez soin de déposer vos deux copies dans les tas correspondants.

- La **partie I** est un exercice de cours, dans lequel vous devez construire des arbres de preuves pour des séquents. Cette partie se situe en bas de cette page.
- La **partie II** est un exercice de logique, portant sur l'étude du problème Horn-Sat. Cette partie se situe aux pages 2 et 3, et contient 9 questions, numérotées de **Q10** à **Q18**.
- La **partie III** est un exercice mêlant retour sur trace et logique, portant sur la résolution du jeu de réflexion *hanjie*. Cette partie se situe aux pages 4 à 9, et contient 15 questions, numérotées de **1** à **15**.
- Enfin, aux pages 9 et 10, vous trouverez une annexe contenant des rappels de programmation en OCaml (utiles pour la partie III) et les règles d'inférence de la déduction naturelle (utiles pour les parties I et III). **On utilisera exclusivement les règles d'inférence données en annexe, et pas celles vues en cours.**

⋆ ⋆ ⋆

Partie I : Déduction naturelle

Les règles d'inférence de la déduction naturelle sont rappelées en annexe, à la page 10 de ce sujet.

Pour chacun des séquents ci-dessous, donner une dérivation en déduction naturelle :

1. $p \vdash q \to (p \wedge q)$
2. $p \to q \vdash p \to (p \wedge q)$
3. $p \to \neg p \vdash \neg p$
4. $p \to r, q \to r \vdash (p \vee q) \to r$

Partie II - Logique et étude du problème Horn-Sat

Dans cette partie, $\wedge, \vee, \neg$, désignent respectivement les connecteurs de conjonction, de disjonction et de négation. Les symboles V, F désignent respectivement les valeurs de vérité VRAI et FAUX du calcul propositionnel.

Dans la suite, on s'intéresse au problème Horn-Sat, qui peut être décrit de la façon suivante : étant donnée une formule de Horn P, existe-t-il une interprétation I telle que $[P]_I = \mathrm{V}$?

L'objectif est d'expliciter un algorithme permettant de résoudre ce problème.

Définition 5 (Formule propositionnelle)
Soit X un ensemble de symboles appelés variables propositionnelles. Les **formules propositionnelles** sur X sont alors définies de façon inductive comme suit :

- V, F sont des formules propositionnelles ;
- tout élément x de X est une formule propositionnelle ;
- si P et Q sont des formules propositionnelles, alors $\neg P, (P \wedge Q), (P \vee Q)$ sont des formules propositionnelles.

Dans la suite, les variables propositionnelles et formules propositionnelles considérées sont construites à l'aide d'un ensemble X qui ne sera pas explicité.

Définition 6 (Littéral)
Soit x une variable propositionnelle. Un **littéral** est la formule x ou la formule $\neg x$. Le littéral x (respectivement $\neg x$) est un littéral positif (respectivement négatif).

Définition 7 (Clause disjonctive)
Soient $n \in \mathbb{N}$, $x_1, x_2, \ldots, x_n$ des littéraux. La formule $x_1 \vee x_2 \vee \ldots \vee x_n$ est appelée **clause disjonctive** à n littéraux (ou de taille n). Par convention, () désigne la **clause disjonctive vide**, c'est-à-dire la clause sans littéral.

Une **clause de Horn** est une clause disjonctive contenant au plus un littéral positif.

Une **clause unitaire** est une clause composée d'un unique littéral.

Définition 8 (Forme normale conjonctive)
Soit P une formule propositionnelle. On dit que P est une **forme normale conjonctive** s'il existe un entier $n \in \mathbb{N}, C_1, C_2, \ldots, C_n$ des clauses disjonctives vérifiant $P = C_1 \wedge C_2 \wedge \ldots \wedge C_n$. Par convention, une forme normale conjonctive sans clause est représentée par $\emptyset$.

Définition 9 (formule de Horn)
Soit P une formule propositionnelle. On dit que P est une **formule de Horn** s'il existe $n \in \mathbb{N}, C_1, C_2, \ldots, C_n$ des clauses de Horn vérifiant $P = C_1 \wedge C_2 \wedge \ldots \wedge C_n$.

Définition 10 (Interprétation)
Soient $n \in \mathbb{N}$ et $x_1, x_2, \ldots x_n$ des variables propositionnelles. Une **interprétation** est une application $I : \{x_1, x_2, \ldots, x_n\} \to \{\mathrm{V}, \mathrm{F}\}$.

Définition 11 (Évaluation)
Soient P une formule propositionnelle, $x_1, x_2, \ldots, x_n$ les variables propositionnelles apparaissant dans P et I une interprétation sur $\{x_1, x_2, \ldots, x_n\}$. **L'évaluation** de P suivant l'interprétation I que l'on note $[P]_I$ est définie par récurrence de la façon suivante :

- si $P \in \{\mathrm{V}, \mathrm{F}\}$, alors $[P]_I = P$;
- si $P \in \{x_1, x_2, \ldots, x_n\}$, alors $[P]_I = I(P)$;
- si $P = \neg Q$ où Q est une formule propositionnelle, alors $[P]_I = \neg[Q]_I$;
- si $P = A \circ B$ où A, B sont des formules propositionnelles et $\circ \in \{\wedge, \vee\}$, alors $[P]_I = [A]_I \circ [B]_I$.

Par convention, $[()]_I = \mathrm{F}$ et $[\emptyset]_I = \mathrm{V}$.

Définition 12 (Satisfiable)
Soit P une formule propositionnelle. On dit que P est **satisfiable** s'il existe une interprétation I telle que $[P]_I = \mathrm{V}$.

Q10. Pour chacune des formules suivantes, montrer qu'elle est satisfiable ou qu'elle est non satisfiable.

1. $P_1 = (x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y)$,
2. $P_2 = (x) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee z) \wedge (z)$,
3. $P_3 = ()$,
4. $P_4 = (x \vee \neg y \vee \neg t) \wedge (z \vee \neg t \vee \neg x \vee \neg y)$.

Q11. Parmi les fomules de la **Q10**, lesquelles sont des formules de Horn ? On ne justifiera pas la réponse.

Définition 13 (Propagation unitaire)
Soit P une forme normale conjonctive contenant une clause unitaire (x). On construit une formule P à partir de P de la façon suivante :

1. supprimer de P toutes les clauses où x apparaît ;
2. enlever toutes les occurrences du littéral $\neg x$.

Cette procédure de simplification est appelée **propagation unitaire**.

Par exemple, si $P = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x)$, on a alors $P = (y \vee z)$. Si $P = (x) \wedge (\neg x)$, on a alors $P = ()$.

Dans la suite, on note $\Pi(P)$ une formule sans clause unitaire obtenue en itérant la propagation unitaire sur P. On admet que si $\Pi(P)$ ne contient pas de clause vide $()$ alors $\Pi(P)$ est unique et que si $\Pi(P)$ contient au moins une clause vide alors toute formule sans clause unitaire obtenue par itération de la propagation unitaire sur P contient au moins une clause vide.

Q12. On pose :

$$\begin{aligned} P \;=\; & (x_1) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_1 \vee x_3) \\ & \wedge (x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_5). \end{aligned}$$

Calculer $\Pi(P)$. On pourra donner le résultat directement sans détailler les calculs.

Q13. Soit P une formule de Horn. Montrer que $\Pi(P)$ est une formule de Horn.

Q14. Soit P une forme normale conjonctive. Montrer que P est satisfiable si et seulement si $\Pi(P)$ est satisfiable.

Q15. Soit P une forme normale conjonctive. Montrer que si $()$ apparaît dans $\Pi(P)$, alors P n'est pas satisfiable.

Q16. Soit C une clause de Horn. Montrer que si C n'est ni la clause vide ni une clause unitaire positive, alors C est satisfiable.

Q17. Soit P une formule de Horn ne contenant ni de clause vide ni de clause unitaire positive. Montrer que P est satisfiable.

Q18. Soit P une formule de Horn. Montrer que P n'est pas satisfiable si et seulement si $()$ apparaît dans $\Pi(P)$.

Partie III : Résolution de Hanjie

Attention : cette partie est à rédiger sur une copie différente !

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Des rappels de logique et des extraits du manuel de documentation de OCaml sont reproduits en annexe. Ces derniers portent sur le module `Array`.

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Il faudra coder des fonctions à l'aide du langage de programmation OCaml, en reprenant l'en-tête de fonction fourni par le sujet, sans s'obliger à recopier la déclaration des types. Quand l'énoncé demande de coder une fonction, sauf demande explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de vérifier que des préconditions sont satisfaites.

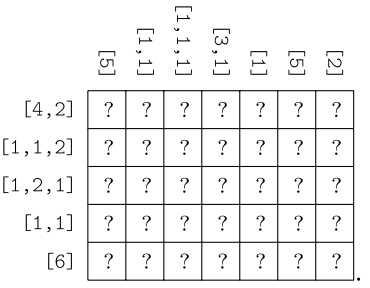
Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Description du jeu du hanjie

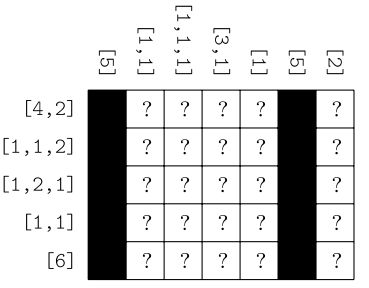
Le *hanjie* est un jeu de réflexion à l'intersection de l'art des pixels et de la tomographie discrète, technique d'imagerie courante en médecine, en géophysique ou en science des matériaux entre autres domaines.

Il consiste à retrouver une image par le noircissement de certaines cases d'une grille rectangulaire sur la base d'indications laissées sur les côtés de la grille. Pour chaque rangée, qu'elle soit horizontale ou verticale, le joueur dispose d'une suite d'entiers non nuls t_1, t_2, t_3, etc. qui indiquent que la rangée contient une série de t_1 cases noires consécutives, suivie plus loin d'une série de t_2 cases noires consécutives, et ainsi de suite. Un nombre quelconque de cases blanches peut se trouver en tête ou en queue de rangée ; au moins une case blanche sépare deux séries de cases noires.

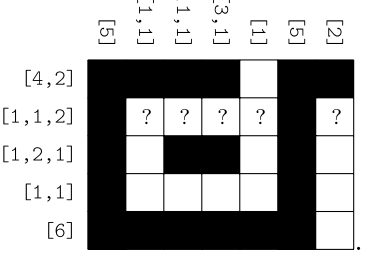
Voici un exemple, que nous résolvons à la main. Une grille vide est fournie ci-contre. Elle est de dimension 5×7. Nous marquons les cases par des symboles « ? » tant que nous ignorons leur couleur.
Nous comptons les rangées à partir de 0, de gauche à droite pour les colonnes et du haut vers le bas pour les lignes.



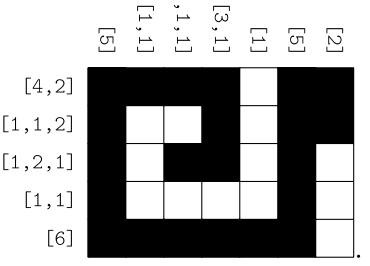
La colonne 0 et la colonne 5 sont de longueur 5. Or l'indication est [5] dans les deux cas. Elles doivent donc chacune contenir 5 blocs noirs consécutifs. Nous les noircissons en totalité.



Dans la ligne 0, il n'y a qu'une seule manière de placer un bloc de 4 cases noires puis 2 cases noires. De même, dans la ligne 2, il n'y a qu'une seule manière de positionner le bloc de longueur 2 : il doit se trouver au milieu entre les deux blocs déjà isolés. Dans la ligne 3, nous avons déjà placé deux cases noires : les autres sont donc toutes blanches. Enfin, dans la ligne 4, il n'y a plus qu'une seule manière de placer un bloc de 6 cases noires.

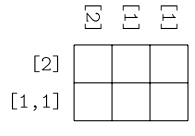


Enfin, en reprenant les indications des colonnes, nous voyons que dans les colonnes 1, 2 et 4, la dernière case inconnue est blanche. Dans la colonne 3 et 6 en revanche, nous devons noircir la dernière case inconnue. Nous avons obtenu une solution de notre hanjie. Elle est unique.

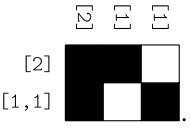


1. Hanjie et calcul de vérité

Dans cette section, nous raisonnons avec la logique propositionnelle pour déterminer la couleur de certaines cases. Nous nous concentrons sur le hanjie h_0 défini par



et dont une solution est la suivante :



☐ 1 – Établir, par raisonnement en langue française, que la solution du hanjie h_0 est unique.

Nous introduisons six variables booléennes, nommées x_0, x_1, …, x_5 et correspondant aux cases ci-contre. Nous associons la valeur de vérité V (vrai) à la couleur noire et F (faux) à la couleur blanche.

x_0	x_1	x_2
x_3	x_4	x_5

Soit L_0 le prédicat : « l'indication de la ligne zéro du hanjie h_0 est satisfaite ».

☐ 2 – Dresser la table de vérité du prédicat L_0 portant sur les variables x_0, x_1, x_2. En déduire une formule de logique φ sous forme normale conjonctive qui décrit le prédicat L_0.

Soit C_1 le prédicat : « l'indication de la colonne du milieu du hanjie h_0 est satisfaite ».

☐ 3 – Dresser la table de vérité du prédicat C_1 portant sur les variables x_1, x_4. En déduire une formule de logique ψ sous forme normale conjonctive qui décrit le prédicat C_1.

Les règles d'inférence de la déduction naturelle sont rappelées dans l'annexe B.

☐ 4 – Construire un arbre de preuve qui démontre le séquent $\varphi \vdash x_1$ à partir des règles d'inférence de la déduction naturelle.

☐ 5 – Construire de même un arbre de preuve qui démontre le séquent $\psi, x_1 \vdash \neg x_4$.

Nous notons ψ' la formule de logique obtenue à partir de ψ en remplaçant la variable x_1 par x_2 et la variable x_4 par x_5.

☐ 6 – Démontrer qu'il n'existe pas d'arbre de preuve qui démontre la formule $\varphi \wedge \psi' \to \neg x_2$.
Indication : on rappelle que la déduction naturelle est correcte : $\Gamma \vdash \varphi$ implique $\Gamma \vDash \varphi$.

2. Cadre de résolution systématique du hanjie

2.1. Le hanjie

Nous fixons un alphabet $\mathcal{C} = \{\mathsf{N}, \mathsf{B}\}$ et notons $\overline{\mathcal{C}}$ l'alphabet complété $\overline{\mathcal{C}} = \{\mathsf{N}, \mathsf{B}, \mathsf{I}\}$. Les symboles N, B et I désignent respectivement les couleurs Noir, Blanc et Inconnu. Nous déclarons en OCaml :

```
type couleur = N | B | I
```

☐ 7 – Écrire une fonction OCaml `est_connu (c:couleur) : bool` dont la valeur de retour est le booléen V (vrai) si $c \in \mathcal{C} = \{\mathsf{N}, \mathsf{B}\}$ et le booléen F (faux) si c égale la couleur I.

Pour l'ensemble du sujet, nous fixons deux entiers naturels non nuls m et n qui désignent respectivement un nombre de lignes et un nombre de colonnes.

Définition : Une *présolution* d'un hanjie est un tableau de dimension $m \times n$ à valeurs dans l'ensemble $\overline{\mathcal{C}}$.
Les figures de l'introduction (en page 2) sont des exemples de présolutions.

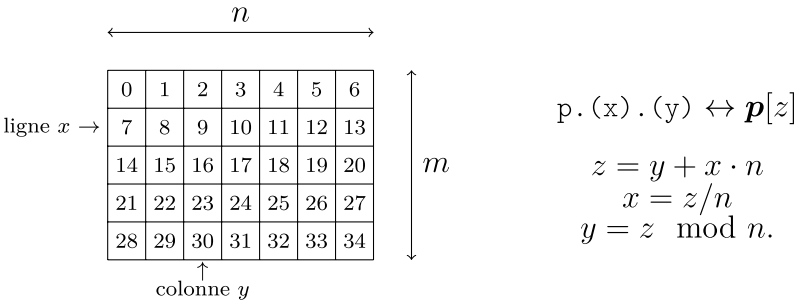
Indication OCaml : Nous déclarons des constantes globales, le type et le constructeur suivants :

```
let m = 5
let n = 7
type presolution = couleur array array

let presolution_init () : presolution =  (* cree une presolution *)
   Array.make_matrix m n I               (* toute egale a I *)
```

Lorsqu'un tableau $\boldsymbol{p}$ est de dimension $m \times n$, nous numérotons ses cases ligne après ligne avec les entiers compris entre 0 et $m \cdot n - 1$. La case en ligne x et colonne y a pour numéro $z = y + x \cdot n$. Nous notons alors $\boldsymbol{p}[z]$ la valeur de $\boldsymbol{p}$ en position z.

Dans notre exemple, nous aurions les numéros :



0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34

☐ 8 – Écrire en langage OCaml un accesseur `get (p:presolution) (z:int) : couleur` dont la valeur de retour est la couleur $\boldsymbol{p}[z]$.

☐ 9 – Écrire en langage OCaml un transformateur `set (p:presolution) (z:int) (c:couleur) : presolution` dont la valeur de retour $\boldsymbol{p'}$ est une copie profonde de $\boldsymbol{p}$ avec $\boldsymbol{p'}[z] = c$, c'est-à-dire une copie n'ayant aucun lien avec la présolution initiale.

Dans tout le sujet, on s'astreint à manipuler le type `presolution` exclusivement en employant les fonctions `presolution_init`, `set` et `get`. De la sorte, on pourra considérer que le type `presolution` est immuable.

☐ 10 – Définir le terme *immuable* et citer un avantage à utiliser des variables immuables.

La *ligne* x d'un tableau, avec $0 \leqslant x < m$, désigne l'ensemble des positions de numéro $x \cdot n$, $x \cdot n + 1$, …, $(x + 1) \cdot n - 1$ (lire n numéros au total). La *colonne* y, avec $0 \leqslant y < n$, désigne l'ensemble des positions de numéro y, $y + n$, …, $y + (m - 1) \cdot n$ (lire m numéros au total). Nous utilisons le terme *rangée* pour parler indifféremment d'une ligne ou d'une colonne.

Définition : Nous disons qu'une rangée d'une présolution est *complète* si elle est à valeurs dans $\mathcal{C}$.

☐ 11 – Écrire une fonction OCaml `est_complete_lig (p:presolution) (x:int) : bool` qui teste si la ligne x de la présolution $\boldsymbol{p}$ est complète.

Nous supposons écrite de même une fonction `est_complete_col (p:presolution) (y:int) : bool`, qui teste si la colonne y de la présolution $\boldsymbol{p}$ est complète.

Définition : Si une rangée est complète, nous appelons *trace* la suite des longueurs des sous-suites maximales de termes consécutifs égaux à Noir. Par exemple, la trace de la rangée

$$\mathsf{B},\underbrace{\mathsf{N},\mathsf{N},}\mathsf{B},\underbrace{\mathsf{N},}\mathsf{B},\underbrace{\mathsf{N},\mathsf{N},\mathsf{N},}\mathsf{B},\underbrace{\mathsf{N}}$$

est $[2;1;3;1]$.

☐ 12 – Écrire une fonction OCaml `trace_lig (p:presolution) (x:int) : int list` dont la valeur de retour est la trace de la ligne x de la présolution $\boldsymbol{p}$.

Nous supposons écrite de la même manière une fonction `trace_col (p:presolution) (y:int) : int list`, dont la valeur de retour est la trace de la colonne y de la présolution $\boldsymbol{p}$.

Définition : Le terme *indication* désigne toute suite finie d'entiers naturels non nuls. Nous appelons *hanjie de dimension* $m \times n$ la donnée d'un tableau d'indications $\boldsymbol{ind}_{lig}$, de longueur m, et d'un tableau d'indications $\boldsymbol{ind}_{col}$, de longueur n.

Indication OCaml : Nous déclarons :

```
type hanjie = { ind_lig : int list array;
                ind_col : int list array }
```

Définition : Une *solution* d'un hanjie $h = (\boldsymbol{ind}_{lig}, \boldsymbol{ind}_{col})$ de dimension $m \times n$ est un tableau $\boldsymbol{p}$ de même dimension $m \times n$ et à valeurs dans l'ensemble $\mathcal{C}$ tel que le tableau des traces des lignes de $\boldsymbol{p}$ égale $\boldsymbol{ind}_{lig}$ et le tableau des traces des colonnes de $\boldsymbol{p}$ égale $\boldsymbol{ind}_{col}$.

Par exemple, le hanjie étudié en introduction et déclaré par :

```
let h_escargot = {ind_lig = [|[4;2]; [1;1;2]; [1;2;1]; [1;1]; [6] |];
                  ind_col = [|[5]; [1;1]; [1;1;1]; [3;1]; [1] ; [5] ;
                              [2] |] }
```

admet comme solution la valeur OCaml :

```
let p_escargot = [|[|N; N; N; N; B; N; N|];
                   [|N; B; B; N; B; N; N|];
                   [|N; B; N; N; B; N; B|];
                   [|N; B; B; B; B; N; B|];
                   [|N; N; N; N; N; N; B|] |]
```

☐ 13 – Écrire une fonction OCaml `est_admissible (h:hanjie) (p:presolution) : bool` dont la valeur de retour est le booléen V (vrai) si et seulement si, pour toute rangée complète de $\boldsymbol{p}$, la trace égale l'indication du hanjie. Il n'est pas nécessaire de contrôler que la présolution et le hanjie ont même dimension.

2.2. Recherche de solutions

Définition : Nous disons qu'une présolution $\boldsymbol{p'}$ *étend* une présolution $\boldsymbol{p}$, si pour tout numéro z avec $\boldsymbol{p}[z] \in \mathcal{C}$, les couleurs $\boldsymbol{p'}[z]$ et $\boldsymbol{p}[z]$ sont égales.
Par exemple, la présolution $\boldsymbol{p'} =$ | B | N | I | N | B | étend $\boldsymbol{p} =$ | B | I | I | N | I | puisque deux cases sont passées de la couleur I à une couleur connue N ou B et les autres sont restées inchangées.

Indication OCaml : Le type paramétré `'a option` permet de distinguer l'absence d'une valeur définie, avec le constructeur `None`, de la présence d'une valeur v de type `'a`, avec la construction `Some v`. Il est défini par la déclaration :

```
type 'a option = None | Some of 'a
```

☐ 14 – Écrire une fonction OCaml `etend_trivial (h:hanjie) (p:presolution) (z:int) (c:couleur) : presolution option` qui, si la copie $\boldsymbol{p'}$ de $\boldsymbol{p}$ avec $\boldsymbol{p'}[z] = c$ est une extension de $\boldsymbol{p}$ et est admissible au sens de la question 13, renvoie `Some p'` et sinon renvoie `None`.

Plus généralement, nous appelons *extenseur* toute fonction OCaml `etend (h:hanjie) (p:presolution) (z:int) (c:couleur) : presolution option` ainsi spécifiée :
Précondition : Le numéro z est valide ($0 \leqslant z < m \cdot n$). La couleur c appartient à $\mathcal{C}$.
Postcondition : Si la valeur de retour vaut `Some` $\boldsymbol{p'}$, alors

(i) $\boldsymbol{p'}$ est une extension admissible de $\boldsymbol{p}$ avec $\boldsymbol{p'}[z] = c$

(ii) et toute présolution complète $\boldsymbol{p''}$ qui satisfait (i) est une extension de $\boldsymbol{p'}$.

Sinon, si la valeur de retour est `None`, alors il n'existe pas de présolution $\boldsymbol{p'}$ complète vérifiant (i).

Nous définissons le type

```
type extenseur = hanjie -> presolution ->
                 int -> couleur -> presolution option
```

Nous souhaitons implémenter une recherche de solutions par une stratégie de *retour sur trace* dans laquelle les cases d'une présolution sont considérées par numéro croissant.

☐ 15 – Compléter le code suivant afin que, si $\boldsymbol{p}$ est une présolution dont au moins les $z-1$ premières cases appartiennent à $\mathcal{C}$, alors `explore p z` renvoie si possible une solution qui étend $\boldsymbol{p}$ à partir du numéro z à l'aide d'itérations sur l'extenseur `ext` et sinon renvoie la valeur `None` :

```
let resout (h:hanjie) (ext:extenseur) : presolution option =
  let p0 = presolution_init () in
  let rec explore (p:presolution) (z:int) : presolution option =
    (* A COMPLETER *)
  in
  explore p0 0
```

A. Annexe : aide à la programmation en OCaml

Opérations sur les tableaux : Le module `Array` offre les fonctions suivantes :

- `length : 'a array -> int`
  Return the length (number of elements) of the given array.
- `make : int -> 'a -> 'a array`
  `Array.make n x` returns a fresh array of length n, initialized with x. All the elements of this new array are initially physically equal to x (in the sense of the == predicate). Consequently, if x is mutable, it is shared among all elements of the array, and modifying x through one of the array entries will modify all other entries at the same time.
- `make_matrix : int -> int -> 'a -> 'a array array`
  `Array.make_matrix dimx dimy e` returns a two-dimensional array (an array of arrays) with first dimension *dimx* and second dimension *dimy*. All the elements of this new matrix are initially physically equal to e. The element (x, y) of a matrix m is accessed with the notation `m.(x).(y)`.

— `init : int -> (int -> 'a) -> 'a array`
`Array.init n f` returns a fresh array of length n, with element number i initialized to the result of $f(i)$. In other terms, `init n f` tabulates the results of f applied to the integers 0 to $n-1$.
— `copy : 'a array -> 'a array`
`Array.copy a` returns a copy of a, that is, a fresh array containing the same elements as a.
— `mem : 'a -> 'a array -> bool`
`mem a l` is true if and only if a is structurally equal to an element of l (i.e. there is an x in l such that `compare a x = 0`).
— `for_all : ('a -> bool) -> 'a array -> bool`
`Array.for_all f [|a1; ...; an|]` checks if all elements of the array satisfy the predicate f. That is, it returns `(f a1) && (f a2) && ... && (f an)`.
— `exists : ('a -> bool) -> 'a array -> bool`
`Array.exists f [|a1; ...; an|]` checks if at least one element of the array satisfies the predicate f. That is, it returns `(f a1) || (f a2) || ... || (f an)`.
— `map : ('a -> 'b) -> 'a array -> 'b array`
`Array.map f a` applies function f to all the elements of a, and builds an array with the results returned by f : `[| f a.(0); f a.(1); ...; f a.(length a - 1) |]`.
— `iter : ('a -> unit) -> 'a array -> unit`
`Array.iter f a` applies function f in turn to all the elements of a. It is equivalent to `f a.(0); f a.(1); ...; f a.(length a - 1); ()`.

D'après `https://v2.ocaml.org/api/Array.html`

B. Annexe : règles de la déduction naturelle

Dans les tableaux suivants, la lettre Δ désigne un ensemble de formules de logique ; les lettres A, B et C désignent des formules de logique.

Axiome
$\dfrac{}{\Delta, A \vdash A}\ (\text{ax})$

	Introduction	Élimination
$\rightarrow$	$\dfrac{\Delta, A \vdash B}{\Delta \vdash A \rightarrow B}\ (\rightarrow\text{i})$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash A \rightarrow B}{\Delta \vdash B}\ (\rightarrow\text{e})$
$\wedge$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash B}{\Delta \vdash A \wedge B}\ (\wedge\text{i})$	$\dfrac{\Delta \vdash A \wedge B}{\Delta \vdash A}\ (\wedge\text{e})$ $\dfrac{\Delta \vdash A \wedge B}{\Delta \vdash B}\ (\wedge\text{e})$
$\vee$	$\dfrac{\Delta \vdash A}{\Delta \vdash A \vee B}\ (\vee\text{i})$ $\dfrac{\Delta \vdash B}{\Delta \vdash A \vee B}\ (\vee\text{i})$	$\dfrac{\Delta \vdash A \vee B \qquad \Delta, A \vdash C \qquad \Delta, B \vdash C}{\Delta \vdash C}\ (\vee\text{e})$
$\neg$	$\dfrac{\Delta, A \vdash B \qquad \Delta, A \vdash \neg B}{\Delta \vdash \neg A}\ (\neg\text{i})$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash \neg A}{\Delta \vdash B}\ (\neg\text{e})$