Concours commun Mines-Ponts, France

Session 2023, filière MP

Informatique option MP

Durée : 3 heures

Ce que traite le sujet

- Section 1 : hanjie et calcul de vérité
- Section 2 : cadre de résolution systématique
- Section 3 : résolution autonome des lignes et extenseur

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/mines/mp/2023/info-option

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

A2023 – INFO MP

**ÉCOLE DES PONTS PARISTECH,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.**

**Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).**

CONCOURS 2023

ÉPREUVE D'INFORMATIQUE MP

Durée de l'épreuve : 3 heures

L'usage de la calculatrice ou de tout dispositif électronique est interdit.

Cette épreuve concerne uniquement les candidats de la filière MP.

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 11 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence
Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France.
Tout autre usage est soumis à une autorisation préalable du Concours commun Mines Ponts.

Préliminaires

L'épreuve est composée d'un problème unique, comportant 27 questions. Après cette section de préliminaires, qui présente le *jeu du hanjie*, le problème est divisé en trois sections qui peuvent être traitées séparément, à condition d'avoir lu les définitions introduites jusqu'à la question traitée. Dans la première section (page 2), nous résolvons le jeu par des raisonnements logiques. Dans la deuxième section (page 3), nous modélisons le jeu et le résolvons par une stratégie algorithmique de retour sur trace. Dans la troisième section (page 6), nous appliquons une méthode de parcours de graphe en théorie des automates pour accélérer la résolution du jeu.

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` et le module `Hashtbl`.

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.

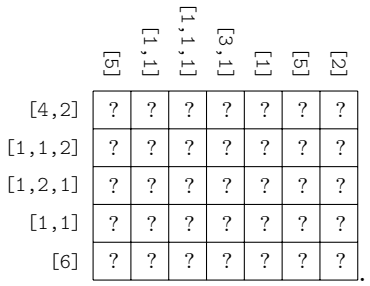
Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Description du jeu du hanjie

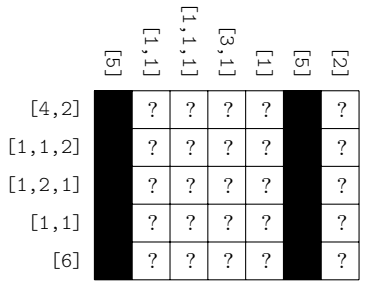
Le *hanjie* est un jeu de réflexion à l'intersection de l'art des pixels et de la tomographie discrète, technique d'imagerie courante en médecine, en géophysique ou en science des matériaux entre autres domaines.

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.

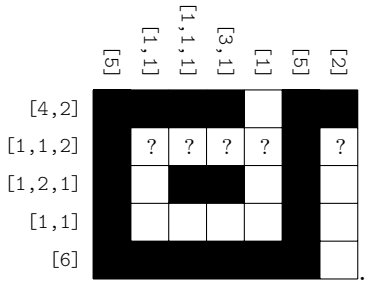
Voici un exemple, que nous résolvons à la main. Une grille vide est fournie ci-contre. Elle est de dimension 5×7. Nous marquons les cases par des symboles « ? » tant que nous ignorons leur couleur.
Nous comptons les rangées à partir de 0, de gauche à droite pour les colonnes et du haut vers le bas pour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



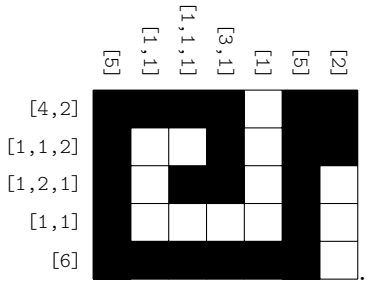
La colonne 0 et la colonne 5 sont de longueur 5. Or l'indication est [5] dans les deux cas. Elles doivent donc chacune contenir 5 blocs noirs consécutifs. Nous les noircissons en totalité.



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.

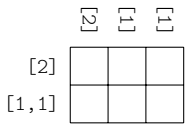


Enfin, en reprenant les indications des colonnes, nous voyons que dans les colonnes 1, 2 et 4, la dernière case inconnue est blanche. Dans la colonne 3 et 6 en revanche, nous devons noircir la dernière case inconnue. Nous avons obtenu une solution de notre hanjie. Elle est unique.

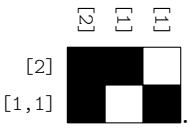


1. Hanjie et calcul de vérité

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 $\varphi \wedge \psi' \to \neg x_2$.

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.

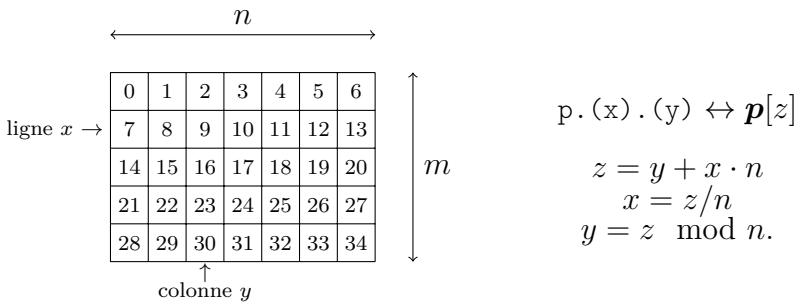
Indication OCaml : Nous déclarons des constantes globales, le type et le constructeur suivants :

```
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 $\boldsymbol{p'} = \boxed{\mathsf{B}}\boxed{\mathsf{N}}\boxed{\mathsf{I}}\boxed{\mathsf{N}}\boxed{\mathsf{B}}$ étend $\boldsymbol{p} = \boxed{\mathsf{B}}\boxed{\mathsf{I}}\boxed{\mathsf{I}}\boxed{\mathsf{N}}\boxed{\mathsf{I}}$ 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
```

3. Résolution autonome de lignes et extenseur

La stratégie d'extension de la question 14 est très naïve. Elle ignore les déductions intermédaires qui pourraient être faites grâce à une seule indication à partir d'une rangée partiellement complétée.

Soit $w \in \overline{\mathcal{C}}^n$ le mot formé des inscriptions dans une rangée dont l'indication est $\boldsymbol{\tau} = [t_1; t_2; \ldots; t_s]$ avec $s \geqslant 1$. Nous notons $[w]_{\boldsymbol{\tau}}$ l'ensemble des mots de $\mathcal{C}^n$ de trace $\boldsymbol{\tau}$ qui étendent w. Lorsque l'ensemble $[w]_{\boldsymbol{\tau}}$ n'est pas vide, nous posons, pour tout indice i compris entre 0 et $n-1$, l'ensemble des couleurs

$$E_i = \{z_i \in \mathcal{C}; \boldsymbol{z} = z_0 \cdots z_{n-1} \in [w]_{\boldsymbol{\tau}}\}.$$

Enfin nous posons, pour tout indice i compris entre 0 et $n-1$,

$$x_i = \begin{cases} \mathsf{B} & \text{si } E_i = \{\mathsf{B}\} \\ \mathsf{N} & \text{si } E_i = \{\mathsf{N}\} \\ \mathsf{I} & \text{si } E_i = \{\mathsf{B},\mathsf{N}\}. \end{cases}$$

Définition : Nous appelons *plus grande extension commune du mot w par rapport à l'indication $\boldsymbol{\tau}$* le mot $\boldsymbol{x} = x_0 \cdots x_{n-1} \in \overline{\mathcal{C}}^*$, lorsqu'il est défini.

Par exemple, si nous rencontrons la ligne | N | I | I | I | I | I | N | I | avec l'indication `[1,2,1]`, il y a deux extensions complètes possibles : | N | B | N | N | B | B | N | B | ou | N | B | B | N | N | B | N | B |. La plus grande extension commune est dans ce cas | N | B | I | N | I | B | N | B |.

☐ 16 – Dans cette question uniquement, nous supposons que n est de la forme $n = 3s-1$, où s est un entier naturel non nul, que l'indication $\boldsymbol{\tau}$ est égale à $[1;1;\ldots;1]$ avec s occurrences de 1 et que w vaut I^n. Compter le nombre de mots du langage $[w]_{\boldsymbol{\tau}}$.

Nous notons $L_{\boldsymbol{\tau}}$ le langage des mots de $\mathcal{C}^*$ de trace $\boldsymbol{\tau} = [t_1;t_2;\ldots;t_s]$ quelconque. Nous supposons toujours que l'on a $s \geqslant 1$.

☐ 17 – Proposer une expression régulière e qui dénote le langage $L_{\boldsymbol{\tau}}$. Aucune justification n'est attendue.

☐ 18 – Dessiner un automate fini déterministe incomplet à $s + \sum_{i=1}^{s} t_i$ états qui reconnaît le langage $L_{\boldsymbol{\tau}}$. Aucune justification n'est attendue.

☐ 19 – Dire combien d'états l'automate de Glushkov qui dérive de l'expression régulière e de la question 17 possède et si cet automate coïncide avec celui dessiné à la question 18.

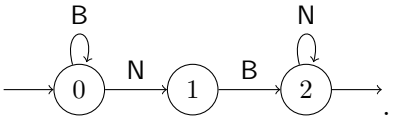
Dans ce qui suit, nous nous intéressons à des *automates finis déterministes incomplets* qui ne possèdent qu'un seul état final et dont l'alphabet est toujours $\mathcal{C}$. Nous numérotons systématiquement les états de sorte que l'état initial soit 0 et l'unique état final soit $r-1$, où r est le nombre total d'états. Le terme *transition* désigne un triplet (q,c,q') où q et q' sont des états (avec $0 \leqslant q, q' < r$) et c est une couleur (avec $c \in \mathcal{C}$). Nous représentons la fonction de transition par un tableau de longueur r ; la case q du tableau contient un dictionnaire qui, quand il existe une transition (q,c,q'), associe la couleur c à l'état q'. Nous déclarons :

```
type automate = { r : int;
                  transitions : (couleur, int) Hashtbl.t array}
```

Définition : Le *déploiement* d'un automate $\mathcal{A}$ ayant r états par un mot $\boldsymbol{w} = w_0 w_1 \cdots w_{n-1} \in \overline{\mathcal{C}}^*$ de longueur n est un nouvel automate, noté $\mathcal{A} \bowtie \boldsymbol{w}$, qui possède $r \cdot (n+1)$ états. Pour toute transition (q,c,q') dans l'automate $\mathcal{A}$ et pour tout indice i compris entre 0 et $n-1$,

pour $w_i \in \mathcal{C}$ et $c = w_i$ ou encore pour $w_i = \mathsf{I}$, il y a dans l'automate $\mathcal{A} \bowtie \boldsymbol{w}$ une transition $(i \cdot r + q, c, (i+1) \cdot r + q')$.

☐ 20 – Dans cette question uniquement, on considère l'exemple avec $n = 4$, $\boldsymbol{w_0} = \mathsf{INBI} \in \overline{\mathcal{C}}^*$ et où $\mathcal{A}_0$ est l'automate défini par



.

Dessiner le déploiement $\mathcal{A}_0 \bowtie \boldsymbol{w_0}$. Marquer les états accessibles. Marquer les états co-accessibles, c'est-à-dire les états à partir desquels il existe un chemin vers l'état final.

☐ 21 – Soient $\mathcal{A}$ un automate et $\boldsymbol{w} \in \mathcal{C}^*$ un mot. Écrire une fonction OCaml `accessible (a:automate) (w:couleur array) : bool array` dont la valeur de retour $\boldsymbol{b}$ est un tableau de booléens tels que $\boldsymbol{b}[q]$ est vrai si et seulement si l'état q de $\mathcal{A} \bowtie \boldsymbol{w}$ est accessible.

☐ 22 – Calculer la complexité en temps de la fonction `accessible` définie à la question 21.

☐ 23 – Soient $\mathcal{A}$ un automate à r états et $\boldsymbol{w} \in \mathcal{C}^*$ un mot de longueur n. Écrire une fonction `coaccessible (a:automate) (w:couleur array) : bool array` dont la valeur de retour $\boldsymbol{b}$ est un tableau de booléens tels que $\boldsymbol{b}[q]$ est vrai si et seulement si l'état q de $\mathcal{A} \bowtie \boldsymbol{w}$ est co-accessible.

Définition : Soient $\mathcal{A}$ un automate à r états et $\boldsymbol{w} \in \overline{\mathcal{C}}^*$ un mot de longueur n. Nous supposons qu'il existe un mot de longueur n reconnu par l'automate $\mathcal{A} \bowtie \boldsymbol{w}$. Nous rappelons que l'automate émondé de $\mathcal{A} \bowtie \boldsymbol{w}$ est la copie de l'automate duquel ont été retirées toutes les transitions depuis ou vers des états qui ne sont pas à la fois accessibles et co-accessibles (nous ne chassons pas les états inutiles et conservons la numérotation des états). Nous notons $\widehat{\Delta}$ l'ensemble des transitions restantes dans l'automate émondé. Nous notons, pour tout indice i compris entre 0 et $n-1$, l'ensemble des étiquettes

$$H_i = \left\{ c \in \mathcal{C};\ (q, c, q') \in \widehat{\Delta} \cap [\![i \cdot r, (i+1) \cdot r - 1]\!] \times \mathcal{C} \times [\![(i+1) \cdot r, (i+2) \cdot r - 1]\!] \right\}.$$

Nous posons, pour tout entier i compris entre 0 et $n-1$,

$$y_i = \begin{cases} \mathsf{B} & \text{si } H_i = \{\mathsf{B}\} \\ \mathsf{N} & \text{si } H_i = \{\mathsf{N}\} \\ \mathsf{I} & \text{si } H_i = \{\mathsf{B}, \mathsf{N}\}. \end{cases}$$

Nous appelons *mot projeté du déploiement* $\mathcal{A} \bowtie \boldsymbol{w}$ le mot $\boldsymbol{y} = y_0 y_1 \dots y_{n-1} \in \overline{\mathcal{C}}^*$.

☐ 24 – Avec les notations du paragraphe qui précède et lorsque l'automate $\mathcal{A}$ est l'automate dessiné à la question 18, vérifier que le mot projeté $\boldsymbol{y} \in \overline{\mathcal{C}}^*$ est la plus grande extension commune du mot $\boldsymbol{w} \in \overline{\mathcal{C}}^*$ par rapport à l'indication $\boldsymbol{\tau}$ (*cf.* définition p. 7).

☐ 25 – Écrire une fonction `projete (a:automate) (w:couleur array) : couleur array` dont la valeur de retour est le mot projeté du déploiement $\mathcal{A} \bowtie \boldsymbol{w}$.

Nous rappelons la *formule de Stirling* :

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n .$$

☐ 26 – Comparer la complexité en temps de la fonction `projete` (question 25) avec un calcul direct de la plus grande extension commune par énumération de l'ensemble $[w]_{\boldsymbol{\tau}}$. Argumenter en s'appuyant sur la question 16.

Nous nous donnons une fonction OCaml `pgec_lig (h:hanjie) (p:presolution) (x:int) : presolution option`, et respectivement `pgec_col (h:hanjie) (p:presolution) (y:int) : presolution option`, qui étend la ligne x, respectivement la colonne y, par la plus grande extension commune à l'image de la question 25 ou bien renvoie `None` si la plus grande extension commune n'est pas définie.

☐ 27 – Écrire un extenseur `etend_nontrivial (h:hanjie) (p:presolution) (z:int) (c:couleur) : presolution option` qui modifie la couleur de $\boldsymbol{p}$ au numéro z et applique les fonctions `pgec_lig` et `pgec_col` aussi longtemps que cela permet de faire progresser la présolution. Il est demandé de détailler la stratégie de l'extenseur avant d'en fournir le code.

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`

Opérations sur les tables de hachage : Le module `Hashtbl` offre les fonctions suivantes :

- `('a, 'b) Hashtbl.t`
  The type of hash tables from type $'a$ to type $'b$.
- `create : int -> ('a, 'b) t`
  `Hashtbl.create n` creates a new, empty hash table, with initial size n. For best results, n should be on the order of the expected number of elements that will be in the table. The table grows as needed, so n is just an initial guess.
- `add : ('a, 'b) t -> 'a -> 'b -> unit`
  `Hashtbl.add tbl key data` adds a binding of key to data in table tbl.
- `remove : ('a, 'b) t -> 'a -> unit`
  `Hashtbl.remove tbl x` removes the current binding of x in *tbl*, restoring the previous binding if it exists. It does nothing if x is not bound in *tbl*.
- `mem : ('a, 'b) t -> 'a -> bool`
  `Hashtbl.mem tbl x` checks if x is bound in *tbl*.
- `iter : ('a -> 'b -> unit) -> ('a, 'b) t -> unit`
  `Hashtbl.iter f tbl` applies f to all bindings in table *tbl*. f receives the key as first argument, and the associated value as second argument. Each binding is presented exactly once to f.
- `find_opt : ('a, 'b) t -> 'a -> 'b option`
  `Hashtbl.find_opt tbl x` returns the current binding of x in *tbl*, or `None` if no such binding exists.

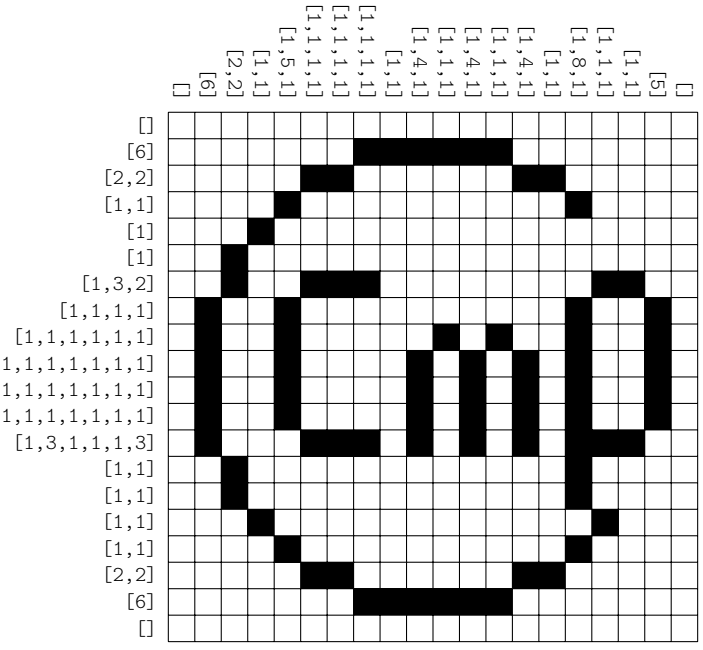
D'après `https://v2.ocaml.org/api/Hashtbl.html`

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

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



Fin de l'épreuve

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Logique, ensembles et applications
- Langage OCaml : expressions, types, filtrage, récursivité
- Récursivité
- Dictionnaires et principes du hachage
- Complexité des algorithmes itératifs et récursifs



Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/mines/mp