1 Hanjie et calcul de vérité

Question 1. Dans le hanjie h_0, la première colonne est nécessairement coloriée car on veut 2 cases coloriées et il n'y en a que 2.
On regarde ensuite les lignes :

- la première demande deux cases coloriées consécutives. Comme la première est coloriée, il n'y a qu'une possibilité pour la ligne.
- la deuxième ligne demande $[1, 1]$ sur 3 cases, il n'y a qu'une possibilité.

Enfin, on vérifie que l'unique possibilité convient pour les deux dernière colonnes.
La solution est unique.

Question 2. Dressons la table de vérité :

x_0	x_1	x_2	L_0
F	F	F	F
F	F	V	F
F	V	F	F
F	V	V	V
V	F	F	F
V	F	V	F
V	V	F	V
V	V	V	F

On en déduit la formule $\phi = (x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2)$

Question 3. Dressons la table de vérité :

x_1	x_4	C_1
F	F	F
F	V	V
V	F	V
V	V	F

On en déduit la formule $\psi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$

Question 4. Avec la définition de $\phi = (x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2)$ choisie, il suffit d'appliquer une règle.

$$\dfrac{\dfrac{}{(x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2) \vdash (x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2)}\,(ax)}{(x_1) \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2) \vdash x_1}\,\wedge e$$

Question 5. Avec la définition de $\psi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$ choisie, on construit un arbre de preuve :

$$\dfrac{\dfrac{\dfrac{}{\psi, x_1, x_4 \vdash \psi}\,(ax)}{\psi, x_1, x_4 \vdash \neg x_1 \vee \neg x_4}\,\wedge e \qquad \dfrac{}{\psi, x_1, x_4, \neg x_1 \vdash \neg x_1}\,(ax) \qquad \dfrac{\dfrac{}{\psi, x_1, x_4, \neg x_4 \vdash \neg x_4}\,(ax) \qquad \dfrac{}{\psi, x_1, x_4, \neg x_4 \vdash x_4}\,(ax)}{\psi, x_1, x_4, \neg x_4 \vdash \neg x_1}\,\neg e}{\psi, x_1, x_4 \vdash \neg x_1}\,\vee e$$

$$\dfrac{\dfrac{}{\psi, x_1, x_4 \vdash x_1}\,(ax) \qquad \psi, x_1, x_4 \vdash \neg x_1}{(x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_4), x_1 \vdash \neg x_4}\,\neg i$$

Question 6. On considère $\psi' = (x_2 \vee x_5) \wedge (\neg x_2 \vee \neg x_5)$.
Supposons par l'absurde qu'il existe un arbre de preuve de $\phi \wedge \psi' \to \neg x_2$.
Par correction des règles de la déduction naturelle, il s'agit d'une tautologie.
Considérons la valuation v tel que $v(x_0) = F, v(x_1) = V, v(x_2) = V, v(x_5) = F$.
D'après la table de vérité de la question 2, c'est un modèle de ϕ, c'est également un modèle de ψ' d'après la question 3.
C'est donc un modèle de $\phi \wedge \psi'$, mais pas de $\neg x_2$ donc la valuation ne satisfait pas $\phi \wedge \psi' \to \neg x_2$.
CONTRADICTION : ce n'est pas une tautologie, il n'existe pas d'arbre de preuve.

2 Cadre de résolution systématique du hanjie

2.1 Le hanjie

Question 7.

```
let est_connu c = not(c=I)
```

Question 8.

```
let get p z = p.(z/n).(z mod n)
```

Question 9.

```
let set p z c =
    let pprime = presolution_init () in
    for i = 0 to m-1 do
        for j = 0  to n-1 do
            pprime.(i).(j) <- p.(i).(j)
        done;
    done;
    pprime.(z/n).(z mod n) <- c;
    pprime
```

Question 10. Un type immuable permet de manipuler des objets dont l'état ne change pas après création. Le principal avantage est qu'il n'y a pas d'effets de bords, on peut appeler différentes fonctions sans craindre de voir l'objet modifié.

Question 11.

```
let est_complete_lig p x =
    let rec test i = match i with
        |0 -> est_connu (get p (x*n))
        |i -> (est_connu (get p (i+x*n))) && (test (i-1))
    in test (n-1)

(* Le sujet suppose qu'il existe aussi : *)
let est_complete_col p y =
    let rec test i = match i with
        |0 -> est_connu (get p y)
        |i -> (est_connu (get p (y+i*n))) && (test (i-1))
    in test (m-1)
```

Question 12.

```
let trace_lig p x =
    let rec parcours i serie =
        let c = get p (x*n+i) in
        match (i,c,serie) with
            |i,B,serie when i = n-1 && serie > 0 -> [serie]
            |i,B,serie when i = n-1 && serie = 0 -> []
            |i,N,serie when i = n-1 -> [serie+1]
            |_,B,serie when serie > 0 -> serie::(parcours (i+1) 0)
            |_,B,serie -> (parcours (i+1) 0)
            |_,N,serie -> (parcours (i+1) (serie+1))
            |_,I,_ -> failwith "Non complet"
    in parcours 0 0

(* Et pour les colonnes comme supposé écrit dans le sujet *)
let trace_col p y =
    let rec parcours i serie =
        let c = get p (y+n*i) in
```

```
        match (i,c,serie) with
            |i,B,serie when i = m-1 && serie > 0 -> [serie]
            |i,B,serie when i = m-1 && serie = 0 -> []
            |i,N,serie when i = m-1 -> [serie+1]
            |_,B,serie when serie > 0 -> serie::(parcours (i+1) 0)
            |_,B,serie -> (parcours (i+1) 0)
            |_,N,serie -> (parcours (i+1) (serie+1))
            |_,I,_ -> failwith "Non complet"
    in parcours 0 0
```

Question 13. On écrit d'abord une fonction qui vérifie l'égalité entre deux liste pour comparer les traces calculées aux résultats attendues.

```
let rec egal l1 l2 = match (l1,l2) with
    |[],[] -> true
    |[],_ -> false
    |_,[] -> false
    |t1::q1, t2::q2 -> (t1 = t2) && (egal q1 q2)
let est_admissible h p =
    let rec test_lig x = match x with
        |(-1) -> true
        | x  when est_complete_lig p x -> (egal (trace_lig p x) (h.ind_lig.(x)))
                                          && test_lig (x-1)
        | x -> test_lig (x-1) in
    let rec test_col y = match y with
        |(-1) -> true
        | y when est_complete_col p y -> (egal (trace_col p y) (h.ind_col.(y)))
                                         && test_col (y-1)
        | x -> test_col (y-1) in
    test_lig (m-1) && test_col (n-1)
```

2.2 Recherche de solutions

Question 14.

```
let etend_trivial h p z c =
    let pprime = set p z c in
    if est_admissible h pprime then Some pprime
    else None
```

Question 15. On applique une méthode de retour sur trace.

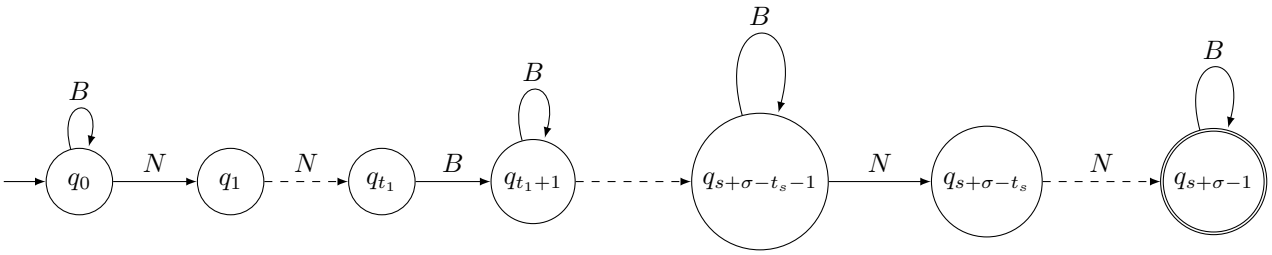
```
let resout (h:hanjie) (ext:extenseur) : presolution option =
    let p0 = presolution_init () in
    let rec explore (p:presolution) (z:int) : presolution option =
        (*début correction*)
        if z < n*m then begin
            let possibilite1 = ext h p z B in
            let possibilite2 = ext h p z N in
            match (possibilite1,possibilite2) with
                |None,None -> None
                |Some p1, None -> explore p1 (z+1)
                |None, Some p2 -> explore p2 (z+1)
                |Some p1, Some p2 -> let s1 = explore p1 (z+1) in
                                     if s1 = None then explore p2 (z+1)
                                     else s1
        end
        else if est_admissible h p then Some p
        else None
        (*fin correction*)
    in
    explore p0 0
```

3 Résolution autonome de lignes et extenseur

Question 16. Soit $w = I^n$ avec $n = 3s - 1$ et $\tau = [1; 1; ...; 1]$ avec s 1.
Les mots du langage $[w]_\tau$ sont les mots comportant s N séparés chacun par au moins un B et $2s - 1$ B.
On cherche donc à place $2s - 1$ B parmi $s + 1$ emplacement (début, entre, fin).
Cela revient à chercher $x_0, x_1, ..., x_s$ tel que $x_0 + x_1 + ... + x_s = 2s - 1$ et $x_0, x_s \in \mathbb{N}$ et $x_1, ..., x_{s-1} \in \mathbb{N}^*$.
On ajoute 1 au premier et dernier pour obtenir un problème plus facile : $x'_0, x_1, ..., s_{s-1}, x'_s$ tel que $x'_0 + x_1 + ... + x_{s-1} + x'_s = 2s + 1$ et $x'_0, x'_s \in \mathbb{N}^*$ et $x_1, ..., x_{s-1} \in \mathbb{N}^*$.
On écrit $2s+1$ B : $BBBB...BB$. Cela revient ainsi à choisir s séparateurs pour ce mot ($2s$ choix pour les places de N).
Ainsi $|[w]_\tau| = \binom{2s}{s}$.

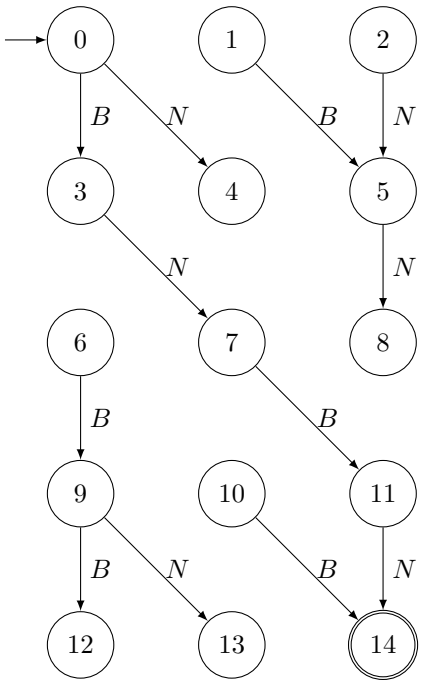
Question 17. $B^*N^{t_1}BB^*N^{t_2}BB^*...N^{t_{s-1}}BB^*N^{t_s}B^*$
Où N^{t_i} est un abus de notation pour $NN...N$ avec t_i occurences de N.

Question 18. On note $\sigma = \sum_{i=1}^{s} t_i$



Question 19. L'automate de Glushkov a autant d'états que de lettres dans l'expression régulière plus 1 pour l'état initial.
Avec l'expression choisie dans la question 17, on a $\sum_{i=1}^{s} t_i$ occurences de N et $2s$ occurences de B, soit $2s+\sum_{i=1}^{s} t_i$ états : il y en a plus, l'automate n'est pas le même.

Question 20. On obtient l'automate suivant :



Les états accessibles sont : 0,3,4,7,11,14
Les états co-accessibles sont : 0,3,7,11,10,14

Question 21. On remarque que dans le premier niveau (états entre 0 et $r - 1$) de l'automate, seul 0 est accessible.
Ensuite, on procède par récurrence sur les niveaux : toutes les transitions vont d'un niveau i (états $r \times i$ à $r \times i + r - 1$) à un niveau $i + 1$, si l'état de départ est accessible, alors l'état d'arrivée aussi.

```
1 let accessible a w =
2     let n = Array.length w in
```

```
    let b = Array.make ((n+1)*a.r) false in
    b.(0)<- true;
    let traite i c =
        for q = 0 to a.r-1 do
            if b.(q+i*a.r) then begin
                let qprime = Hashtbl.find_opt (a.transitions.(q)) c in
                match qprime with
                    |Some etat -> b.(a.r*(i+1)+etat) <- true
                    |None -> ()
            end
        done; () in
    let rec step i =
        begin match w.(i) with
            |I -> traite i B ; traite i N
            |_ -> traite i (w.(i)) end ;
        if (i<n-1) then step (i+1) in
    step 0;
    b
```

Question 22. Dans le pire des cas, on traite une fois chaque état pour ses transitions sortantes, qui sont au plus 2. On est en $O(nr)$.

Question 23. Cette fois, on fait le calcul "à l'envers". Au niveau n, on a uniquement $(n+1)r-1$ qui est coaccessible.
Un état est co-accessible si et seulement si il existe une transition vers un état co-accessible du niveau suivant.

```
let coaccessible a w =
    let n = Array.length w in
    let b = Array.make ((n+1)*a.r) false in
    b.((n+1)*a.r-1)<- true;
    let traite i c =
        for q = 0 to a.r-1 do
            let qprime = Hashtbl.find_opt (a.transitions.(q)) c in
            match qprime with
                |Some etat when b.((i+1)*a.r+etat) -> b.(i*a.r+q)<-true
                |_-> ()
        done; () in
    let rec step i =
        begin match w.(i) with
            |I -> traite i B ; traite i N
            |_ -> traite i (w.(i)) end ;
        if (i>0) then step (i-1) in
    step (n-1);
    b
```

Question 24. Question formulée de façon un peu étrange, on veut bien vérifier mais écrire quoi ?
Pour un mot w, on a $|w|+1$ niveau dans l'automate déployé avec des transitions N entre deux niveaux si $w_i = N$, B si $w_i = N$ et N et B si $w_i = N$ ou B.
Chaque mot reconnu par l'automate déployé est un mot formé de N et de B qui coïncide avec w car on avance d'une lettre et d'un niveau simultanément.
Chaque mot reconnu par l'automate déployé est un mot reconnu par l'automate original car on peut retrouver le chemin dans l'automate original en ignorant les niveaux.
Réciproquement, un mot reconnu par l'automate et coïncidant avec w est reconnu par l'automate déployé.
Ainsi, dans l'automate émondé, entre chaque niveau on a toutes les "couleurs possibles" pour w_i et seulement celle qui correspondent à au moins une solution pour l'indication.
Ainsi, on obtient en effet la plus grande extension commune.

Question 25. On commence par copier w.
Pour chaque I dans w on cherche une transition B et une transition N dans le niveau correspondant, s'il en manque, on modifie y en conséquence.

```
let projete a w =
    let n = Array.length w in
```

```
    let y = Array.copy w in
    let acc = accessible a w in
    let coacc = coaccessible a w in
    let rec cherche i c q = (*cherche la couleur c entre le niveau i et i+1 depuis q*)
        if acc.(q+i*a.r) && coacc.(q+i*a.r) then begin
            let qprime = Hashtbl.find_opt (a.transitions.(q)) c in
            match qprime with
                |Some etat when acc.(etat+(i+1)*a.r) && coacc.(etat+(i+1)*a.r) -> true
                |_ -> if q+1 = a.r then false
                      else cherche i c (q+1)
        end
        else if q+1 = a.r then false
        else cherche i c (q+1) in
    for i = 0 to n-1 do
        if y.(i) = I then
            match (cherche i B 0,cherche i N 0) with
                |False,True -> y.(i) <- B
                |True,False -> y.(i) <- N
                |_-> ()
    done;
    y
```

Question 26. Les fonctions `accessible` et `coaccessible` sont en $O(nr)$, la fonction `projete` traite à nouveau au plus une fois chaque transition donc en $O(nr)$ avec $r = s + \sum_{i=1}^{s}$.
En comparaison, un calcul direct par énumération serait bien plus grand.
Comme le suggère l'énoncé, appuyons nous sur la question 16 pour le justifier.
Dans la question 16, on a $n = 3s - 1$ et $r = 2s$ et $\binom{2s}{s}$ possibilités.
Une énumération exhaustive demanderait l'exploration d'un grand nombre de possibilités :

$$\binom{2s}{s} = \frac{(2s)!}{(s!)^2} \sim \frac{\sqrt{2\pi(2s)}(2s)^{2s}}{e^{2s}} \frac{(e^s)^2}{2\pi s \times (s^s)^2} = \frac{4^s}{\sqrt{\pi s}}$$

On aurait donc une complexité au moins exponentielle.

Question 27. Tant que c'est possible, on applique les deux fonction (lignes puis colonnes) jusqu'à stabilisation.
A chaque étape, le nombre de I diminue strictement, ce qui garantit la terminaison.

```
let egal p1 p2 =
    let rec test z =
        if z = n*m then true
        else (get p1 z = get p2 z) && test (z+1) in
    test 0

let etend_nontrivial h p z c =
    let pprime = set p z c in
    let rec etend_lig presol x =
        if x = m then Some presol
        else match (pgec_lig h presol x) with
                |None -> None
                |Some pl -> etend_lig pl (x+1) in
    let rec etend_col presol y =
        if y = n then Some presol
        else match (pgec_col h presol y) with
                |None -> None
                |Some pc -> etend_col pc (y+1) in
    let rec etend presol = match (etend_lig presol 0) with
             |None -> None
             |Some pl -> begin match (etend_col pl 0) with
                |None -> None
                |Some pc -> if egal pc presol then Some presol else etend pc
                end in
    etend pprime
```