


1

 $h_0$ 

|       |   |     |     |     |
|-------|---|-----|-----|-----|
|       |   | [2] | [1] | [1] |
| [2]   |   |     |     |     |
| [1,1] |   |     |     |     |
|       | 0 | 1   | 2   |     |

- Le  $h_0$  ne comportant que 2 lignes, la colonne 0 est nécessairement totalement noircie, puisque la colonne 0 porte l'indication [2]
- Sur la ligne 1, comme  $h_0$  ne comporte que 3 colonnes, la seule façon de respecter l'indication [1,1] est d'inscrire le motif , sinon il n'y aura pas d'espace entre les blocs noirs.
- Sur la ligne 0, étant donné que l'indication est [2] et que (0,0) est noir, (0,1) est nécessairement noir.
- Cette solution vérifie les autres indications. et est nécessaire. Il n'y en a donc pas d'autres.

2

|       |       |       |
|-------|-------|-------|
| $x_0$ | $x_1$ | $x_2$ |
| $x_3$ | $x_4$ | $x_5$ |

V → noire

F → blanche

 $L_0$ 

| $x_0$ | $x_1$ | $x_2$ | $L_0$ |
|-------|-------|-------|-------|
| F     | F     | F     | V     |
| F     | F     | V     | F     |
| F     | V     | F     | V     |
| F     | V     | V     | F     |
| V     | F     | F     | V     |
| V     | F     | V     | F     |
| V     | V     | F     | V     |
| V     | V     | V     | F     |

exactement  
2 blocs noirs  
consécutifs.

Pour une forme normale conjonctive, on prend les antimodèles et la valuation opposée des variables :

$$\varphi = (\underline{x_0} \vee \underline{x_1} \vee x_2) \wedge (\underline{x_0} \vee \underline{x_1} \vee \neg x_2) \wedge (\underline{x_0} \vee \neg x_1 \vee \neg x_2)$$

$$\textcircled{1} \quad \wedge (\neg x_0 \vee \underline{x_1} \vee x_2) \wedge (\neg x_0 \vee \underline{x_1} \vee \neg x_2) \wedge (\neg x_0 \vee \neg x_1 \vee x_2)$$

$$\varphi = \underline{x_1} \wedge (\underline{x_0} \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee \neg x_1 \vee \neg x_2)$$

$$\textcircled{2} \quad \varphi = x_1 \wedge (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2) \rightarrow \text{FNC}$$

$$(\varphi = x_1 \wedge x_0 \oplus x_2 \quad \text{si } \oplus \text{ dénote})$$

le ou exclusif.

① : ~~quel~~ quels que soient  $x_0$  et  $x_2$ , si  $x_1$  est vraie, alors  $\varphi$  est vraie.

② : pour que  $\varphi$  soit vraie, il faut que  $x_1$  soit vraie et donc  $\neg x_1$  fausse.

On peut donc l'éliminer des clauses.

③

| $x_1$ | $x_4$ | $C_1$ |
|-------|-------|-------|
| F     | F     | F     |
| F     | V     | V     |
| V     | F     | V     |
| V     | V     | F     |

$\rightarrow [1] \rightarrow$  une unique case noire

V  $\rightarrow$  noire  
F  $\rightarrow$  blanche

FNC:

$$\psi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$$

$$(\varphi = x_1 \oplus x_4)$$

4) Démontrer que  $\varphi \vdash \alpha$ .

On applique deux fois l'élimination de  $\wedge$ .

$$\begin{array}{c}
 \text{--- (axiome)} \\
 \varphi \vdash \alpha_1 \wedge (\alpha_0 \vee \alpha_2) \wedge (\neg \alpha_0 \vee \neg \alpha_2) \\
 \text{--- } (\wedge_e) \\
 \varphi \vdash \alpha_1 \wedge (\alpha_0 \vee \alpha_2) \\
 \text{--- } (\wedge_e) \\
 \underbrace{\alpha_1 \wedge (\alpha_0 \vee \alpha_2) \wedge (\neg \alpha_0 \vee \neg \alpha_2)}_{\varphi} \vdash \alpha
 \end{array}$$

5) Démontrer le séquent  $\varphi, \alpha_1 \vdash \neg \alpha_4$

Rq sujet: la définition de  $\neg$  dans le sujet combine l'introduction de la contradiction  $\perp$  et le principe d'explosion  $\perp_e$  accepté en logique classique.

$$\begin{array}{c}
 \text{--- ax} \quad \text{--- ax} \quad \text{--- ax} \\
 \varphi, \alpha_1 \vdash (\alpha_1 \vee \alpha_3) \wedge (\neg \alpha_1 \vee \neg \alpha_4) \quad \varphi, \alpha_1, \neg \alpha_1 \vdash \alpha_1 \quad \varphi, \alpha_1, \neg \alpha_1 \vdash \neg \alpha_4 \\
 \text{--- } \wedge_e \quad \text{--- } \neg_e \quad \text{--- ax} \\
 \varphi, \alpha_1 \vdash \neg \alpha_1 \vee \neg \alpha_4 \quad \varphi, \alpha_1, \neg \alpha_1 \vdash \neg \alpha_4 \quad \varphi, \alpha_1, \neg \alpha_1 \vdash \neg \alpha_4 \\
 \text{--- } \vee_e \\
 \varphi, \alpha_1 \vdash \neg \alpha_4
 \end{array}$$

6)  $\varphi \wedge \varphi' \rightarrow \neg \alpha_2 \equiv \alpha_1 \wedge (\alpha_0 \vee \alpha_2) \wedge (\neg \alpha_0 \vee \neg \alpha_2) \wedge (\alpha_2 \vee \alpha_3) \wedge (\neg \alpha_2 \vee \neg \alpha_5) \rightarrow \neg \alpha_2$

Si  $v(\alpha_2) = V$  et que la prémisse est vraie, l'implication est fautive (contre-exemple). C'est possible

par  $v(\alpha_0) = F$   $v(\alpha_1) = V$   $v(\alpha_2) = V$  et  $v(\alpha_5) = F$   
 $\Rightarrow$  on ne peut pas prouver le séquent. c.q.f.d.

7

let est-connu ( $c: \text{couleur}$ ): bool =  $c \neq \perp$

Rq:  $\bar{E} = \{N, B, I\}$

let  $m = 5$  (\* constantes globales \*)

let  $n = 7$

type presolution = couleur array array

let presolution\_init() = Array.make\_matrix  $m$   $n$   $\perp$

$p.(x).(y) \leftrightarrow p[z]$

$z = y + x \cdot n$

$x = z/n$

$y = z \bmod n$

8

let get p z =

let  $x = z/n$  and  $y = z \bmod n$  in  
 $p.(x).(y)$

9

let set p z c =

let  $p2 = \text{presolution\_init}()$  in

for  $i = 0$  to  $m$  do

$p2[i] \leftarrow \text{Array.copy } p.(i)$

done;

let  $x = z/n$  and  $y = z \bmod n$

in  $p2.(x).(y) \leftarrow c;$

$p2;;$

10 Une variable immuable est une variable dont on ne peut pas modifier la valeur après initialisation.

L'utilisation de variables immuables permet d'éviter les effets de bord. La contrepartie est qu'il faut un bon ramasse-miette!

11 let est-complete-lig p x =  
 let found = ref false in  
 let j = ref 0 in  
 while not !found && !j < n do  
 if p.(x).(j) = I then found := true;  
 incr j;  
 done;  
 not !found;; (\* on cherche les I \*)

12 let trace-lig p x =  
 let c = ref 0 in let trace = ref [] in  
 for j = n-1 downto 0 do  
 let s = p.(x).(j) in  
 match s with  
 | I -> failwith "Ligne incomplète"  
 | N -> incr c  
 | B -> if !c > 0 then (trace := !c :: !trace;  
 c := 0)  
 done;  
 if !c > 0 then (trace := !c :: !trace);  
 !trace;;

```

113 let est-admissible h p =
    let ad-l = ref true in
    let i = ref 0 in
    while !i < m && !ad-l do
        match est-complete-ly p !i in
        | true -> ad-l := trace-ly p !i = h.ind-ly(!i)
        | false -> ();
        incr i;
    done;
    let ad-c = ref true in
    let j = ref 0 in
    while !j < n && !ad-c do
        match est-complete-col p !j in
        | true -> ad-c := trace-col p !j = h.ind-col(!j)
        | false -> ();
        incr j;
    done;
    !ad-l && !ad-c ;;

```

## CCMP 2023 - Option info

```

14 let extend-trivial h p z c =
    let cc = get p z in
    if c = cc || cc = I then
        let mp = set p z c in
        if est-admissible h mp
        then Some mp
        else None
    else None ;;
  
```

```

15 let recout h ext =
    let po = presolution-init() in
    let rec explore p z =
        if z = m * m
        then Some p (* condition d'arrêt, σ nouveau *)
        else let p1 = ext h p z B
              and p2 = ext h p z N in
              match p1, p2 with
              | None, None -> None
              | Some q1, None -> explore q1 (z+1)
              | None, Some q2 -> explore q2 (z+1)
              | Some q1, Some q2 ->
                  let q3 = explore q1 (z+1) in
                  match q3 with
                  | None -> explore q2 (z+1)
                  | Some -> q3
    in explore po 0 ;;
  
```

16 L'indication  $\tau = [1; 1; \dots; 1]$  avec  $s$  occurrences de 1 indique qu'il y a  $s$  cases N noires isolées dans la ligne.

Tout mot de  $[w]_\tau$  s'écrit alors :

$$w = B^{i_0} N B B^{i_1} N B B^{i_2} \dots N B^{i_s}$$

avec  $i_0 + i_1 + \dots + i_s = s$

$$(i_0, i_1, \dots, i_s) \in \mathbb{N}^{s+1}$$

On note que  $|w|_N = s$

$$|w|_B = s - 1 + s = 2s - 1$$

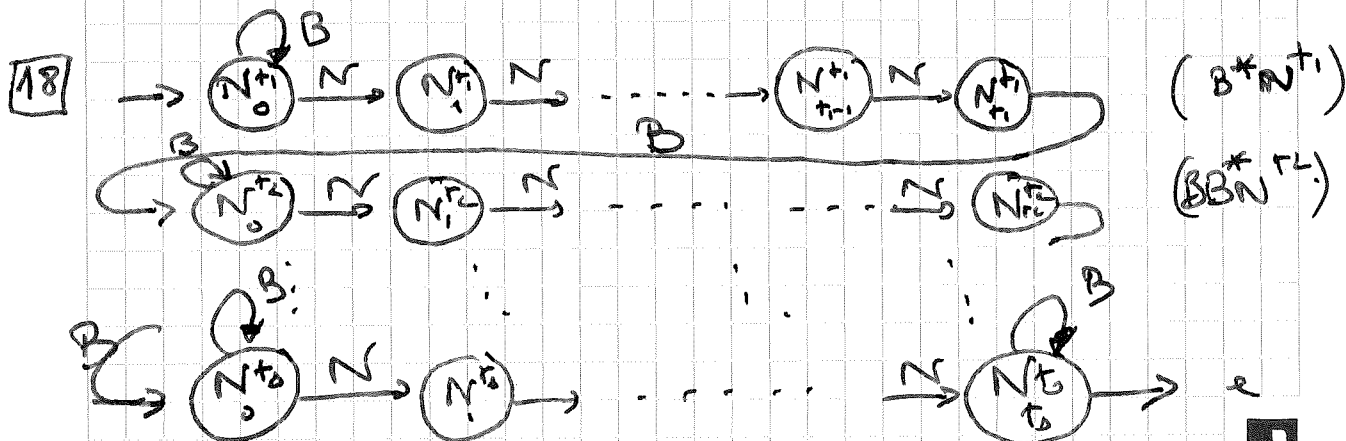
Donc  $|w| = 3s - 1$  ce qui est cohérent avec l'énoncé.

On peut regarder le mot  $w$  ainsi :

$$w = p^{i_0} q p^{i_1} q \dots q p^{i_{s-1}}$$

De ce point de vue, il s'agit de choisir  $s$  positions pour  $p$  parmi  $2s$ , soit  $\binom{2s}{s}$  mots dans  $[w]_\tau$

17  $\mathcal{L}_\tau = \mathcal{L}_{ER} (B^* N^{t_1} B B^* N^{t_2} B B^* \dots N^{t_s} B^*)$





18) Suite. Cet automate possède

$$\sum_{i=1}^n (t_i + 1) = n + \sum_{i=1}^n t_i \text{ états.}$$

ceci est un 1... GL est fini, déterministe, incomplet

19)

L'automate de Glushkov possède autant d'états que de lettres dans l'expression rationnelle, plus l'état initial. D'où

$$\underbrace{1}_{nb\ B^*} + \underbrace{n}_{nb\ B} + \underbrace{\sum_{i=1}^n t_i}_{nb\ N} = 20 + 1 + \sum_{i=1}^n t_i \text{ états.}$$

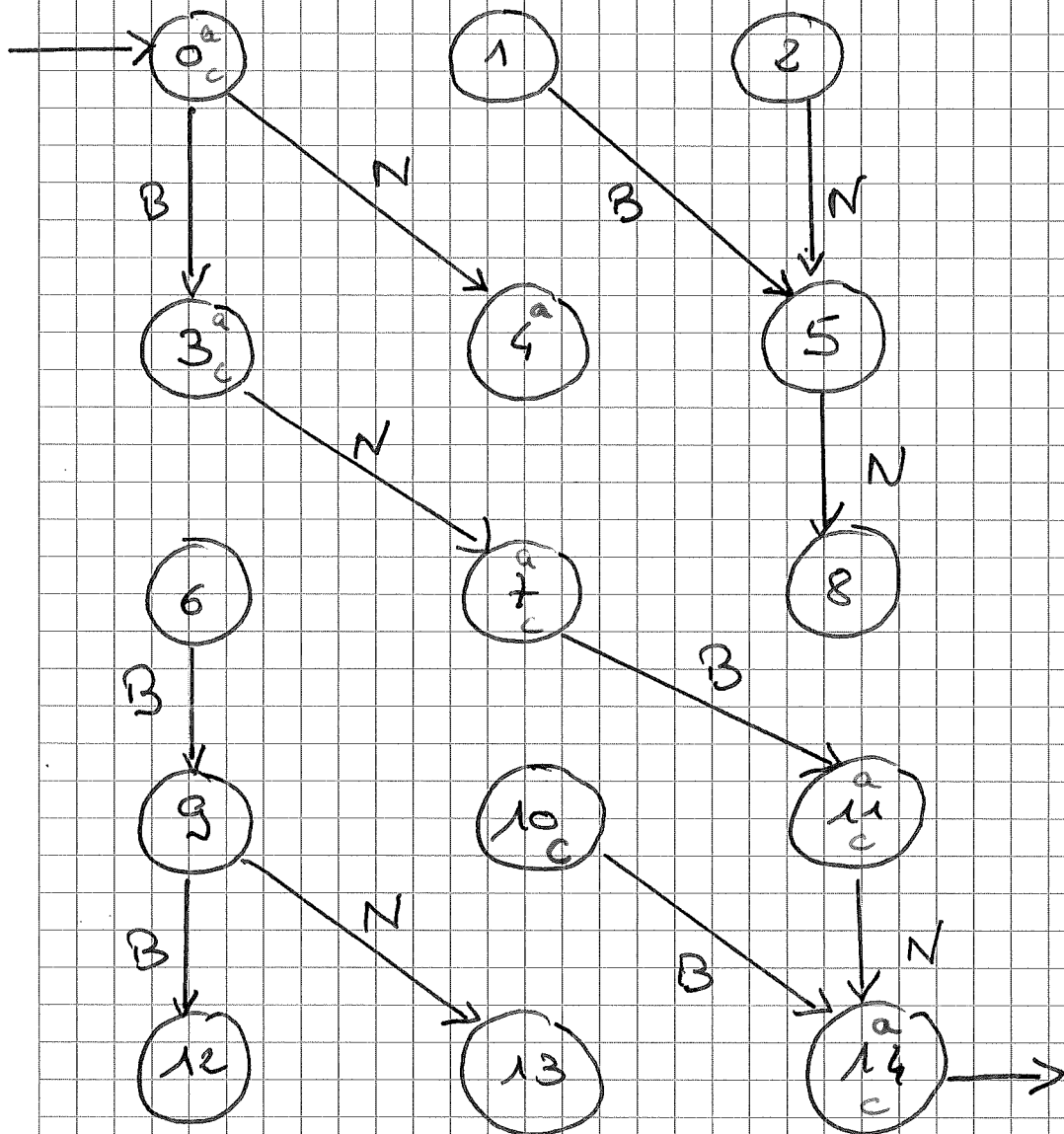
20) Procéder methodiquement, question pénible...

| $n=3$ | $i^0$ | $w_i$            | $c$ | $q < q'$   | $q < q'$   | $q < q'$    | $q < q'$    |
|-------|-------|------------------|-----|------------|------------|-------------|-------------|
|       |       |                  |     | (0, B, 0)  | (0, N, 1)  | (1, B, 2)   | (2, N, 2)   |
| 0     | I     | $c$<br>(acquies) |     | (0, B, 3)  | (0, N, 4)  | (1, B, 5)   | (2, N, 5)   |
| 1     | N     | $c=N?$           | X   |            | (3, N, 7)  | X           | (5, N, 8)   |
| 2     | B     | $c=B?$           |     | (6, B, 9)  | X          | (7, B, 11)  | X           |
| 3     | I     | $c$<br>(acquies) |     | (9, B, 12) | (9, N, 13) | (10, B, 14) | (11, N, 14) |

On peut maintenant dessiner l'automate en marquant les sommets accessibles a et les — coaccessibles c.



[20] suite...



On déduit que 14 est l'état final du fait que c'est l'aboutissement du seul chemin reconnaissant un mot de 4 lettres...  
Ceci est implicite dans le sujet

Rq: question de cours. Attention à l'usage  
des HashTbl.

Bien penser au tableau decouverts nécessaire pour  
parcourir en largeur le graphe.

```
[21] let accessible a w =  
    let n = Array.length w in  
    let decouverts = Array.make (a.r * (n+1)) false in  
    let rec parcours - largeur q i =  
        if not decouverts.(i * a.r + q)  
        then  
            begin  
                decouverts.(i * a.r + q) <- true;  
                if i < n (* le mot fait n lettres *)  
                then (* Donc on doit trouver un chemin *)  
                    adapted i < n  
                    begin  
                        let c = w.(i) in  
                        match HashTbl.find_opt a.transitions.(q) N with  
                        | None -> ()  
                        | Some s -> if c = N || c = I then  
                            parcours - largeur s (i+1);  
                        match HashTbl.find_opt a.transitions.(q) B with  
                        | None -> ()  
                        | Some s -> if c = B || c = I then  
                            parcours - largeur s (i+1);  
                    end  
                end  
            end  
    in parcours - largeur 0 0;  
    decouverts;;
```

# d'états potentiel  
de A  $\rightarrow$  w

sommet de départ

décompte de la  
longueur du  
chemin parcouru

22

Quelques observations :

- La création de découverts est en  $O(r(m+1))$
- La fonction parcours-largeur garantit qu'on n'a exploré un sommet accessible qu'une seule fois. (if not decouvert...)
- Chaque sommet accessible ne peut être découvert que :
  - par une transition B
  - ou par une transition N

donc, parcours-largeur est appelé au plus 2 fois par sommet accessible. Le nombre d'appels récurifs est donc majoré par  $2rm$  car il ne peut pas y avoir plus de  $r(m+1)$  états accessibles.

- La complexité de accessible est donc en  $O(rm)$ .

23

Question péniible par plusieurs raisons :

- 1) On ne peut pas réutiliser la fonction accessible ~~car~~ et le graphe transposé car :
  - \* HashTbl pour la f° de transition, c'est à dire un AFD : en effet pour  $(q, c) = \text{def}$  on ne peut avoir qu'une seule valeur associée dans le dictionnaire...
  - \* Le transposé d'un AFD n'est pas forcément déterministe...

2) le code à produire est long et laborieux ...

Idee: parcourir l'automate à partir d'une longueur de chemin  $n-1$ , car on cherche un mot de taille  $n$ , s'il existe une transition vers un état accepteur, c'est un état coaccessible.

(Je traite la question pour y revenir plus tard...)

[24] Pour bien comprendre les notations, on aurait déjà pu essayer sur l'automate de la question 20, etc...

Sur  $et_0 \bowtie w_0$  on vérifie que :

$$\rightarrow (0) \xrightarrow{B} (3) \xrightarrow{N} (7) \xrightarrow{B} (11) \xrightarrow{N} (14) \rightarrow$$

$$\hat{\Delta} = \{(0, B, 3), (3, N, 7), (7, B, 11), (11, N, 14)\}$$

$$H_0 = \{B\} \text{ car } \Delta \cap [0, 2] \times \{B, N\} \times [3, 5] \text{ pour } i=0 \text{ ne laisse que } (0, B, 3)$$

$$H_1 = \{N\} \text{ (idem) avec } (3, N, 7)$$

$$H_2 = \{B\} \text{ (idem) avec } (7, B, 11)$$

$$H_3 = \{N\} \text{ (idem) avec } (11, N, 14)$$

$$y = BNB N$$

On avait  $w_0 = INBI$

L'automate  $et_0$  (respectant les indications) ne permet pas d'ajouter un B après le dernier B ni d'ajouter un B avant le premier N.

Donc  $y$  est la plus grande extension commune de  $w_0 \in T^*$  par rapport aux indications  $\tau$ .

On peut maintenant essayer par l'automate de la question 18...

→ faudrait...

Il faut vérifier que  $\mathcal{L}_{A \bowtie w} = [w]_{\tau}$

dans le cadre de la question que  
 $y$  est la plus grande extension commune.  
et donc que  $H_i = E_i \quad \forall i \in [0, n-1]$

Procédons par double inclusion

( $\supset$ ) Soit  $i \in [0, n-1]$  et un ensemble  $E_i$ ;  
Soit  $c \in E_i$ , alors il existe un mot  
 $z = z_0 z_1 \dots z_{n-1} \in [w]_{\tau}$  tel que  $c = z_i$ .

Comme  $z$  respecte  $\tau$ ,  $z$  est reconnu  
par  $A$ , c'est-à-dire :

$\exists$  une transition  $(q, c, q')$  dans  $A$ .

Comme  $z$  est une extension de  $w$ , on a  
 $c = z_i = w_i$  ou bien  $c = \bar{\epsilon}$

Donc  $((i+1)\tau + q, c, (i+1)\tau + q')$   
est une transition de  $\hat{A} \bowtie w$ ,  
donc  $\in \hat{\Delta}$ , car le mot est  
accepté par  $\hat{A} \bowtie w$ .

Donc  $c \in H_i$ .

( $\subset$ ) Soit  $c \in H_i$ ; alors  $\exists q, q'$  deux états de  $A \bowtie w$   
tels que  $(q, c, q') \in \hat{\Delta} \cap [i\tau, (i+1)\tau - 1] \times (x[i\tau, (i+1)\tau - 1])$   
et  $q$  et  $q'$  sont co-accessibles.

Alors il existe un chemin étiqueté par  $z \in [w]_{\tau}$   
qui passe par cette transition  $\Rightarrow c \in E_i$



25 Let project a  $w =$   
au se cours ... cf fichier Camel.

26 En tenant compte des initialisations et de l'usage des  $f^\circ$  accessibles et coaccessibles, on atteint  $O(m)$ .

Rq : en 16 on a trouvé  $|[w]_z| = \binom{15}{0}$

$$\text{donc } |[w]_z| \underset{z \rightarrow 0}{\sim} \frac{1}{\sqrt{\pi z}} 2^z$$

Donc l'approche proposée par mot project est une approche meilleure en termes de complexité que la force brute qui serait exponentielle.

27 ... cf fichier Camel.

