

1 Problème sur les automates finis

1 - Le langage choisi ici est Caml.

2 - Le graphe G est représenté par un tableau à 3 dimensions de booléens.

Nous utiliserons le fait qu'en Caml, les indices des tableaux commencent à 0,; ainsi pour avoir des indices de 1 à n , nous déclareront un tableau de taille $n + 1$ dont la première case ne sera pas utilisée.

Ainsi, pour $(i, k, j) \in [1; N_E] \times [1, N_L] \times [1, N_E]$, $G[i, k, j] = \text{true} \iff e_i \xrightarrow{\ell_k} e_j$.

```
let initialise_graphe ne nl =
  let m = make_vect (ne+1) (make_matrix (nl+1) (ne+1) false)
  in
    for i = 1 to ne do m.(i) <- (make_matrix (nl+1) (ne+1) false) done;
  m ;;
```

Remarque :

l'objectif de la boucle est de créer un nouveau tableau à deux dimensions pour chaque indice i , sans quoi tous les $m.(i)$ "pointent" sur le même tableau et alors, toute modification d'un élément $m.(i)$ se répercute sur tous les autres.

3 - Implémentation du graphe correspondant au schéma de l'énoncé.

```
let ne = 5;;
let nl = 2;;

let G = let m = initialise_graphe 5 2 in
  m.(1).(1).(2) <- true;
  m.(2).(2).(3) <- true;
  m.(3).(1).(1) <- true;
  m.(3).(2).(5) <- true;
  m.(3).(1).(4) <- true;
  m.(5).(1).(4) <- true;
m ;;
```

4 - La matrice d'adjacence est un tableau de type `bool vect vect` à $N_E + 1$ lignes et $N_E + 1$ colonnes, la première ligne et la première colonne n'étant pas utilisées.

5 - La matrice est initialisée avec `false`. Si $i \neq j$, alors $C[i, j] = \text{true} \iff \exists k \in [1, N_E] / G[i, k, j] = \text{true}$.

```
let initialise_matrice_adjacence a =
  let m = make_matrix (ne + 1) (ne + 1) false in
  for i = 1 to ne do
    for j = 1 to ne do
      for k = 1 to nl do
        if i = j then m.(i).(j) <- true
        else for k = 1 to nl do
          if a.(i).(k).(j) then m.(i).(j) <- true
        done
      done
    done
  done;
m;;

let C = initialise_matrice_adjacence G;;
```

*Remarque : on pourrait remplacer la boucle **for** portant sur **k** par une boucle **while** et s'arrêter dès que $C[i, j] = \text{true}$. La complexité est alors la même dans le pire des cas.*

- 6** - On parcourt entièrement le tableau G , d'où une complexité temporelle en $O((N_E)^2 N_L)$.
- 7** - Notons $\mathcal{P}$ la propriété de l'énoncé.
 Dans la matrice C_0 , les calculs relient directement l'origine et l'extrémité sans traverser d'autres états, donc C_0 vérifie $\mathcal{P}$ au rang 0.
- 8** - Supposons que C_{N_E} vérifie la propriété $\mathcal{P}$ au rang N_E .
 Montrons que $C_{N_E} = C$, c'est à dire $\forall(i, j), (C_{N_E}[i, j] = \text{true} \iff C[i, j] = \text{true})$.
 * il est évident d'abord que $(C_{N_E}[i, j] = \text{true} \implies C[i, j] = \text{true})$.
 * Réciproquement, si $C[i, j] = \text{true}$, alors il existe un calcul d'origine e_i et d'extrémité e_j .
 Ce calcul traverse des états qui sont dans l'automate, donc ont tous un numéro $\leq N_E$, donc $C_{N_E}[i, j] = \text{true}$.
- 9** - On suppose que C_k vérifie $\mathcal{P}$ au rang k et que C_{k+1} vérifie $\mathcal{P}$ au rang $k+1$.
 • Supposons que $C_{k+1}[i, j] = \text{true}$.
 * si $i = j$, alors $C_k[i, j] = \text{true}$.
 * si $i \neq j$, alors il existe un calcul d'origine e_i et d'extrémité e_j ne traversant que des états de numéro $\leq k+1$ (puisque C_{k+1} vérifie $\mathcal{P}$ au rang $k+1$).
 - s'il ne traverse que des états de numéro $\leq k$, alors $C_k[i, j] = \text{true}$ (puisque C_k vérifie $\mathcal{P}$ au rang k).
 - sinon, il traverse au moins une fois e_{k+1} . Si ce calcul traverse plusieurs fois e_{k+1} , on peut se ramener, en supprimant toutes les boucles d'origine et d'extrémité e_{k+1} , au cas d'un calcul ne traversant qu'une seule fois e_{k+1} .
 Ce calcul est donc constitué de la juxtaposition de deux calculs, l'un joignant e_i à e_{k+1} , l'autre joignant e_{k+1} à e_j , tous deux ne traversant que des états de numéro $\leq k$, donc $C_k[i, k+1] = \text{true}$ et $C_k[k+1, j] = \text{true}$.
 • La réciproque est immédiate.
- 10** - On suppose que C_k vérifie $\mathcal{P}$ au rang k et que C_{k+1} vérifie $\mathcal{P}$ au rang $k+1$.
a - L'implication $(C_k[i, k+1] = \text{true} \implies C_{k+1}[i, k+1] = \text{true})$ est évidente car C_k vérifie $\mathcal{P}$ au rang k et C_{k+1} vérifie $\mathcal{P}$ au rang $k+1$.
 Supposons que $C_{k+1}[i, k+1] = \text{true}$. Il existe donc un calcul joignant e_i à e_{k+1} ne traversant que des états de numéro $\leq k+1$ car C_{k+1} vérifie $\mathcal{P}$ au rang $k+1$. En supprimant de ce calcul les boucles éventuelles passant par e_{k+1} , on se ramène à un calcul joignant e_i à e_{k+1} dans lequel tous les états traversés ont un numéro $\leq k$, donc $C_k[i, k+1] = \text{true}$.
b - Analogie
- 11** - On suppose connaître la matrice C_k vérifiant $\mathcal{P}$ à l'ordre k .
 D'après **9**, pour obtenir C_{k+1} , il suffit de mettre à **true** tous les $C_k[i, j]$ tels que $C_k[i, j] = \text{false}$, $C_k[i, k+1] = \text{true}$ et $C_k[k+1, j] = \text{true}$. D'après **10**, ce sont les seuls cas à considérer.
- 12** - L'algorithme précédent nécessite de parcourir toute la matrice, d'où une complexité en $O((N_E)^2)$.
- 13** -

```

let matrice_accessible a =
  let m = initialise_matrice_adjacence a in
    for k = 1 to ne do
      for i = 1 to ne do
        for j=1 to ne do
          if (not m.(i).(j)) && m.(i).(k) && m.(k).(j)
            then m.(i).(j) <- true
        done
      done
    done;
  m;;

let C = matrice_accessible G ;;

```

14 - Le calcul de C_0 a une complexité en $O((N_L (N_E)^2)$ et chaque boucle portant sur k en $O((N_E)^2)$, d'où une complexité totale en $O((N_L (N_E)^2 + (N_E)^3)$.

15 - Dans le cas de l'exemple proposé, cela donne en initialisant tous les éléments de U avec **true** :

```
let I = [|true; true;false;false;false;false|];; (* le premier true ne compte pas *)
let F = [|true; false;false;true;true|];;
let U = [|true; true;true;true;true;true|];;
```

16 -

```
let etats_inutiles a init final =
  let m = matrice_accessible a in
  for k = 1 to ne do
    let i_acces = ref(false) and f_acces = ref(false) in
    let i = ref 1 in
    while (not ! i_acces) && (!i <= ne) do
      if init.(!i) && m.(!i).(k) then i_acces := true;
      i := !i + 1;
    done;
    let j = ref 1 in
    while (not ! f_acces) && (!j <= ne) do
      if final.(!j) && m.(k).(j) then f_acces := true;
      j := !j + 1;
    done;
    if !i_acces && !f_acces then U.(k) <- false;
  done;
  U;;

etats_inutiles G I F ;;
```

17 - En plus du calcul de la matrice d'accessibilité qui a une complexité en $O((N_L (N_E)^2 + (N_E)^3)$, chaque boucle sur k parcourt dans le pire des cas les deux vecteurs I et F , d'où une complexité en $O(N_E)$ pour chaque boucle, donc en $O((N_E)^2)$ pour l'ensemble des boucles, négligeable devant $O((N_E)^3)$.

18 - Chaque case contient un booléen, donc son traitement a un coût constant. (Ce ne serait pas le cas si le contenu était un entier long ou une chaîne alphanumérique de longueur variable).

* + * + * + * + * + *

2 Problème de complexité algorithmique

1 - Le langage choisi ici est Caml.

2 - *La méthode élémentaire*

a - let rec es x n = if n=1 then x else x * (es x (n-1)) ;;

b - Le nombre de multiplications est égal à $n - 1$.

2 - *La méthode dichotomique*

a - Supposons que $n = \overline{a_p \dots a_0}^2$ avec $a_p = 1$. Posons $n_p = 1$ et $n_k = \overline{a_p \dots a_k}^2$ pour $0 \leq k \leq p - 1$.

Au départ, on initialise la variable y avec la valeur x .

Montrons qu'à la fin de chaque boucle portant sur k , la variable y contient x^{n_k} .

A l'entrée de la première boucle, y contient $x^{n_p} = x$.

Supposons qu'à l'entrée de la boucle indexée par k , y contienne $x^{n_{k+1}}$.

Or $n_k = 2n_{k+1} + a_k$, donc $x^{n_k} = x^{2n_{k+1} + a_k} = (x^{n_{k+1}})^2 x^{a_k} = y^2 x^{a_k}$. Donc pour avoir x^{n_k} , il faut bien d'abord élever y au carré (c'est à dire remplacer y par $y * y$), puis remplacer y par $y * x$ lorsque $a_k = 1$. C'est justement ce que propose l'algorithme de Legendre, donc à la sortie de la boucle, y contient bien x^{n_k} .

- b** - Avec les notations précédentes, $\lambda(n) = p + 1$. La boucle sur k est effectuée p fois (k variant de $p - 1$ à 0), ce qui donne $p = \lambda(n) - 1$ élévations au carré. Le nombre de multiplications de y par x est égal au nombre de a_k égaux à 1 pour $0 \leq k \leq p - 1$, c'est donc $\nu(n) - 1$.

Le nombre de multiplications est donc égal à $\boxed{\lambda(n) + \nu(n) - 2}$.

3 - La méthode dichotomique

```
let rec ed x n = if n = 1 then x
                  else if pair n then ed (x*x) (n/2)
                  else x * (ed x (n-1)) ;;
```

```
let rec cd n = if n = 1 then 0
                else if pair n then 1 + cd(n/2)
                else 1 + cd (n-1) ;;
```

- c** - $\lambda(1) = 1$.

$\lambda(2k) = 1 + \lambda(k)$ car on rajoute un 0 à droite de la r.b.m. de k pour obtenir celle de $2k$.

$\lambda(2k+1) = \lambda(2k)$ car la r.b.m. de $2k$ se termine par 0 qu'on remplace par 1 pour obtenir celle de $2k+1$.

```
let rec lambda n = if n = 1 then 1
                   else if pair n then 1 + (lambda (n/2))
                   else lambda (n-1) ;;
```

- d** - $\nu(1) = 1$, $\nu(2k) = \nu(k)$, $\nu(2k+1) = 1 + \nu(k)$.

```
let rec mu n = if n = 1 then 1
                else if pair n then mu (n/2)
                else 1 + mu (n-1) ;;
```

- e** - $\pi(1) = 1 + 1 - 2 = 0$

$\pi(2k) = \lambda(2k) + \nu(2k) - 2 = 1 + \lambda(k) + \mu(k) - 2 = 1 + \pi(k)$
 $\pi(2k+1) = \lambda(2k+1) + \nu(2k+1) - 2 = \lambda(2k) + 1 + \mu(2k) - 2 = 1 + \pi(2k)$

```
let rec pi n = if n = 1 then 0
                else if pair n then 1 + pi (n/2)
                else 1 + pi (n-1) ;;
```

- f** - On observe en comparant les fonctions cd et pi qu'elles sont égales.

Donc le nombre de multiplications effectuées par ed est aussi égal à $\boxed{\pi(n) = \lambda(n) + \nu(n) - 2}$.

- g** - Lorsque $n = 2^k$, on trouve que ed nécessite exactement k multiplications.

(Deux façons : $\pi(n) = (k+1) + 1 - 2 = k$ ou bien parce que $p = k$ et que l'algorithme de Legendre n'effectue que des élévations au carré dans ce cas).

5 - La méthode des facteurs

Lorsque $n = 33 = 3 \times 11$, elle consiste à calculer :

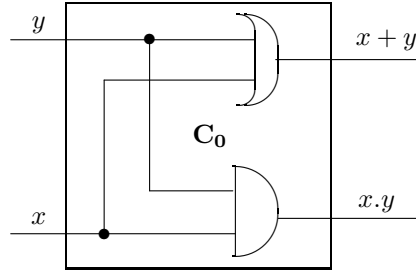
. $x^2 = x * x$, $y = x^3 = x * x^2$
 . $y^2 = y * y$, $y^4 = y^2 * y^2$, $y^5 = y * y^4$, $y^{10} = y^5 * y^5$, $y^{11} = y * y^{10}$.

On a donc effectué $\pi(3) + \pi(11) = 2 + 5 = 7$ multiplications, alors que par la méthode dichotomique nécessite seulement $\pi(33) = 6 + 2 - 2 = 6$ multiplications car $33 = \overline{100001}^2$.

3 Problème de logique des propositions

Dans ce corrigé, j'utiliserai les notations suivantes : $x + y = x \vee y$, $x.y = x \wedge y$ et $\bar{x} = \neg x$.

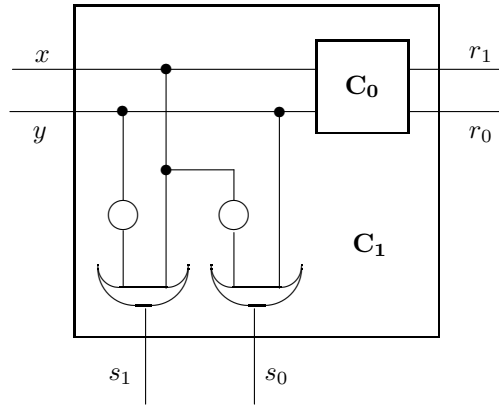
1 - On utilise que $\max(x, y) = x \vee y = x + y$ et $\min(x, y) = x \wedge y = x.y$.



2 - On a immédiatement $r_1 = \max(x, y) = x + y$ et $r_0 = \min(x, y) = x.y$.

| x | y | s_1 | s_0 |
|-----|-----|-------|-------|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

$$\text{donc, } \begin{cases} \bar{s}_1 = \bar{x} \wedge y & \text{d'où } s_1 = \overline{\bar{x} \wedge y} = x \vee \bar{y} \\ \bar{s}_0 = x \wedge \bar{y} & \text{d'où } s_0 = \overline{x \wedge \bar{y}} = \bar{x} \vee y \end{cases}$$



| e_1 | e_0 | x | y | r_1 | r_0 | s_1 | s_0 |
|-------|-------|-----|-----|-------|-------|-------|-------|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

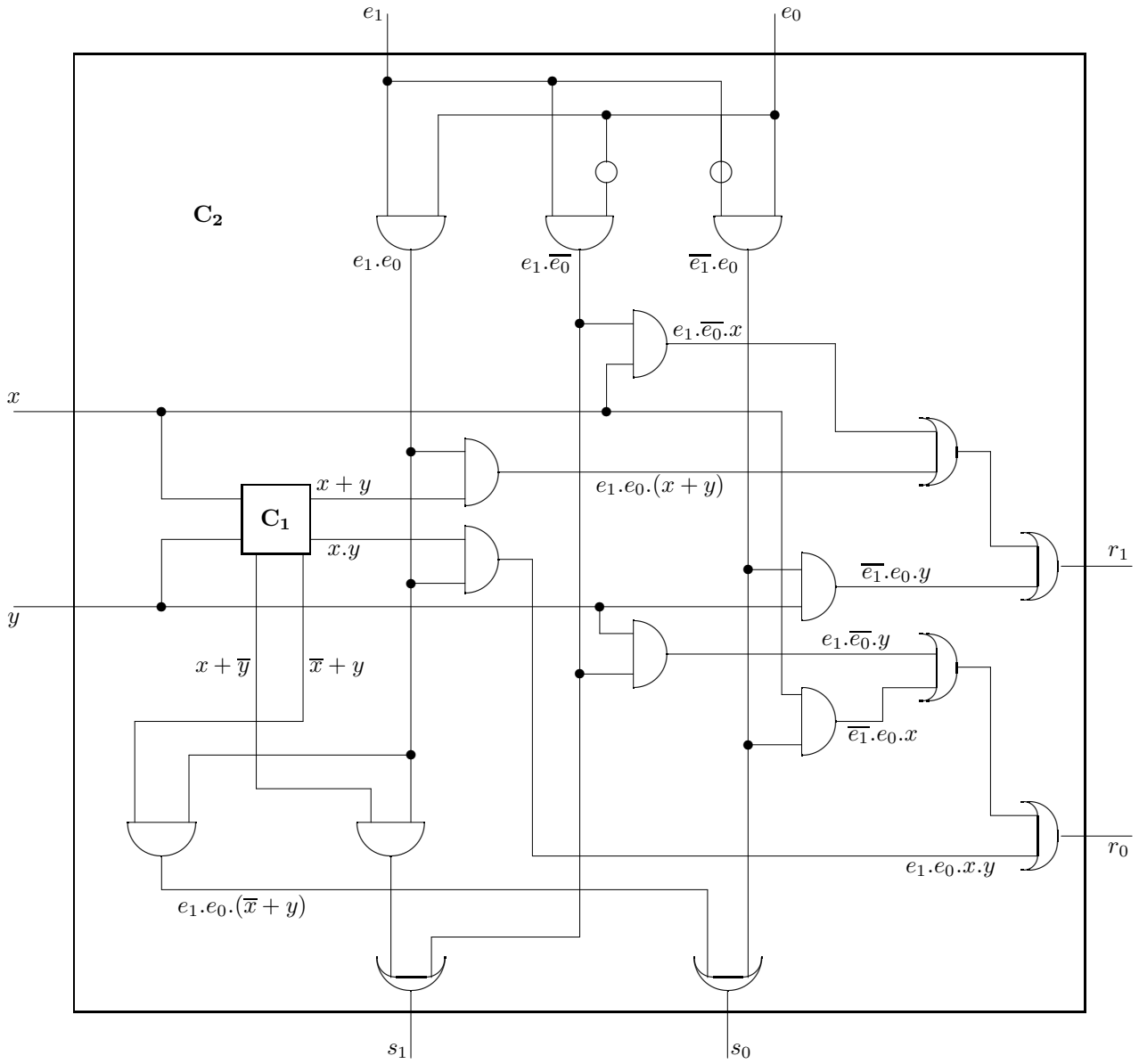
3 -

$$r_1 = e_1.\bar{e}_0.x + \bar{e}_1.e_0.y + e_1.e_0.(x + y)$$

$$r_0 = e_1.\bar{e}_0.y + \bar{e}_1.e_0.x + e_1.e_0.(x.y)$$

$$s_1 = e_1.\bar{e}_0 + e_1.e_0.(x + \bar{y})$$

$$s_0 = \bar{e}_1.e_0 + e_1.e_0.(\bar{x} + y)$$



4 - On applique d'abord le circuit C_1 au couple (a_7, b_7) .

On récupère en particulier en sortie le couple (s'_7, s_7) qui indique le résultat de la comparaison de a_7 et b_7 .

$(s'_7, s_7) = (0, 1)$ si $a_7 < b_7$, $(s'_7, s_7) = (1, 1)$ si $a_7 = b_7$, $(s'_7, s_7) = (1, 0)$ si $a_7 > b_7$.

De plus $c_7 = \max(a_7, b_7)$ et $d_7 = \min(a_7, b_7)$.

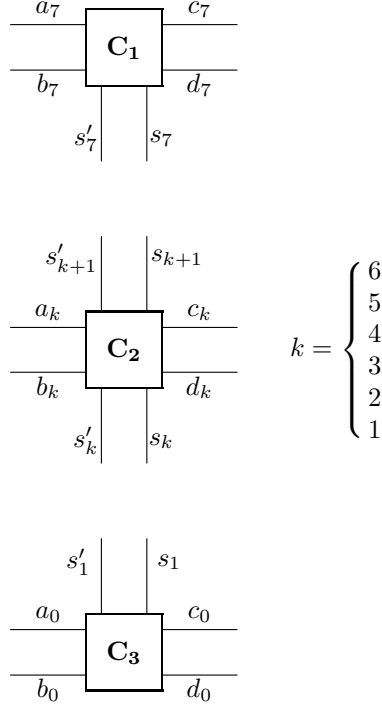
Supposons que :

$$(s'_{k+1}, s_{k+1}) = \begin{cases} (0, 1) & \text{si } \overline{a_7 \dots a_{k+1}}^2 < \overline{b_7 \dots b_{k+1}}^2 \\ (1, 1) & \text{si } \overline{a_7 \dots a_{k+1}}^2 = \overline{b_7 \dots b_{k+1}}^2 \\ (1, 0) & \text{si } \overline{a_7 \dots a_{k+1}}^2 > \overline{b_7 \dots b_{k+1}}^2 \end{cases} \text{ et que } \begin{cases} \overline{c_7 \dots c_{k+1}}^2 = \max(\overline{a_7 \dots a_{k+1}}^2, \overline{b_7 \dots b_{k+1}}^2) \\ \overline{d_7 \dots d_{k+1}}^2 = \min(\overline{a_7 \dots a_{k+1}}^2, \overline{b_7 \dots b_{k+1}}^2) \end{cases}$$

- Si $(s'_{k+1}, s_k) = (1, 0)$, alors $\overline{a_7 \dots a_{k+1}}^2 > \overline{b_7 \dots b_{k+1}}^2$, donc a fortiori $\overline{a_7 \dots a_k}^2 > \overline{b_7 \dots b_k}^2$.
On doit alors renvoyer $c_k = a_k$, $d_k = b_k$ et $(s'_k, s_k) = (1, 0)$, ce qui est bien le cas d'après la 1^e ligne définissant le circuit C_2 .
- Si $(s'_{k+1}, s_k) = (0, 1)$, alors $\overline{a_7 \dots a_{k+1}}^2 < \overline{b_7 \dots b_{k+1}}^2$, donc a fortiori $\overline{a_7 \dots a_k}^2 < \overline{b_7 \dots b_k}^2$.
On doit alors renvoyer $c_k = b_k$, $d_k = a_k$ et $(s'_k, s_k) = (0, 1)$, ce qui est bien le cas d'après la 4^e ligne définissant le circuit C_2 .
- Si $(s'_{k+1}, s_k) = (1, 1)$, alors $\overline{a_7 \dots a_{k+1}}^2 = \overline{b_7 \dots b_{k+1}}^2$.

- * Si $a_k > b_k$, alors $\overline{a_7 \dots a_k}^2 > \overline{b_7 \dots b_k}^2$.
On doit alors renvoyer $c_k = a_k$, $d_k = b_k$ et $(s'_k, s_k) = (1, 0)$,
ce qui est bien le cas d'après la 2^e ligne définissant le circuit C_2 .
- * Si $a_k < b_k$, alors $\overline{a_7 \dots a_k}^2 < \overline{b_7 \dots b_k}^2$.
On doit alors renvoyer $c_k = b_k$, $d_k = a_k$ et $(s'_k, s_k) = (0, 1)$,
ce qui est bien le cas d'après la 5^e ligne définissant le circuit C_2 .
- * Si $a_k = b_k$, alors $\overline{a_7 \dots a_k}^2 = \overline{b_7 \dots b_k}^2$.
On doit alors renvoyer $c_k = a_k$, $d_k = b_k$ et $(s'_k, s_k) = (1, 1)$,
ce qui est bien le cas d'après la 3^e ligne définissant le circuit C_2 .

On vient de montrer par récurrence que le circuit “composé” suivant est correct.



* + * + * + * + * + * + *