Réduction d'automates

OPTION INFORMATIQUE - TP nº 4.x - Olivier Reynet

A Équivalence de Nérode

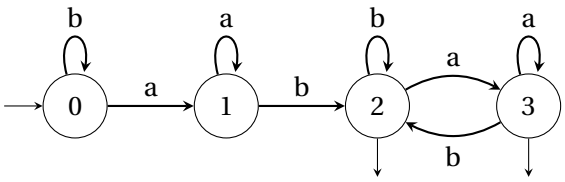
On considère l'automate fini déterministe $\mathcal{A}$ défini comme suit :

- **Alphabet** : $\Sigma = \{a, b\}$,
- **Ensemble des états** : $Q = \{0, 1, 2, 3\}$,
- **État initial** : 0,
- **Ensemble des états acceptants** : $F = \{2, 3\}$,
- **Fonction de transition** (δ) représentée par la table suivante :

État actuel	a	b
0	1	0
1	1	2
2	3	2
3	3	2

A1. Dessiner l'automate correspondant à cette représentation tabulaire.

Solution :



■ **Définition 1 — Équivalence de Nérode.** Soit q un état d'un automate A.
On note $L_q = \{w \in \Sigma^*, \delta^*(q, w) \in F\}$.
On peut alors définir une relation d'équivalence, l'équivalence de Nérode, entre états notée $\sim_A$ définie par :

$$q \sim_A p \Leftrightarrow L_q = L_p \tag{1}$$

A2. Montrer que la relation $\sim_A$ est bien une relation d'équivalence.

Solution : Cela découle du fait que = est une relation d'équivalence. Dans le détail :

- **Réflexivité** : Pour tout $q \in Q$, on a $q \sim_A q$ car $L_q = L_q$.
- **Symétrie** : Si $q \sim_A p$, alors, de même on a $L_p = L_q$ et donc $p \sim_A q$.

— **Transitivité** : Si $q \sim_A p$ et $p \sim_A r$, alors $L_q = L_p$ et $L_p = L_r$ ce qui implique $L_q = L_r$ et donc $r \sim_L r$.

L'équivalence de Nérode est donc bien une relation d'équivalence.

■ **Définition 2 — Congruence à droite.** Soit $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ un automate fini déterministe. Une relation d'équivalence $\sim$ sur l'ensemble des états Q est une congruence à droite si elle est **compatible avec la fonction de transition** δ.

Formellement, pour tout couple d'états $(p,q) \in Q^2$ et pour chaque lettre $a \in \Sigma$:

$$p \sim q \implies \delta(p,a) \sim \delta(q,a)$$

Par récurrence, cette propriété s'étend à tous les mots $w \in \Sigma^*$:

$$p \sim q \implies \forall w \in \Sigma^*, \delta^*(p,w) \sim \delta^*(q,w)$$

A3. Montrer que la relation $\sim_A$ est une congruence à droite, c'est-à-dire que pour tous états $p, q \in Q$ et pour toute lettre $x \in \Sigma$:

$$p \sim_A q \implies \delta(p,x) \sim_A \delta(q,x)$$

Solution : Soit $p \sim_A q$ et $x \in \Sigma$. Par définition, $L_p = L_q$. Un mot w appartient à $L_{\delta(p,x)}$ si et seulement si le mot xw appartient à L_p.
Comme $L_p = L_q$, $xw \in L_p \iff xw \in L_q$, ce qui équivaut à $w \in L_{\delta(q,x)}$.
On a donc $L_{\delta(p,x)} = L_{\delta(q,x)}$, d'où $\delta(p,x) \sim_A \delta(q,x)$.
La congruence à droite garantit que le résultat est le même quel que soit le choix de l'état pour représenter une classe.

A4. Montrer que si $p \sim_A q$, alors soit $\{p,q\} \subseteq F$, soit $\{p,q\} \cap F = \emptyset$.

Solution : Si $p \sim_A q$, alors $L_p = L_q$.

— Si le mot vide ϵ appartient à L_p alors $p \in F$. Or $\epsilon \in L_p \iff \epsilon \in L_q$, donc $p \in F \iff q \in F$. Donc si p est accepteur, q l'est aussi.
— Si le mot vide n'appartient pas à L_p, alors $p \notin F$. Mais alors le mot vide n'appartient pas non plus à L_q. Donc $\{p,q\} \cap F = \emptyset$

Les états d'une même classe sont donc du même type, soit accepteur, soit non accepteur.

A5. Soit $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ un automate.

(a) Déduire de la question précédente une méthode pour construire une partition la plus grossière possible permettant d'initier la recherche des classes d'équivalence dans un automate $\mathcal{A}$.

Solution : Pour que deux états soient dans la même classe, ils doivent au minimum avoir le même statut accepteur/non-accepteur. La partition initiale doit donc séparer ces états :

$$P_0 = \{F, Q \setminus F\}$$

(b) Donner la valeur de cette partition P_0 pour l'automate $\mathcal{A}$ de la question A1.

Solution : Pour l'automate de l'exercice, $F = \{2,3\}$ et $Q \setminus F = \{0,1\}$.On a donc $P_0 = \{\{2,3\},\{0,1\}\}$.

B Algorithme de Moore

On cherche maintenant à contruire l'automate minimal $\tilde{\mathcal{A}}$ associé à $\mathcal{A}$ selon l'algorithme de Moore.

1. Pour créer les classes d'équivalence, on construit d'abord une première partition P_0 comme dans la question A5.
2. Pour trouver les classes d'équivalence, on construit un tableau représentant les transistions depuis les parties. Si, pour une même partie, une lettre fait transiter dans deux parties différentes, alors on affine la partie en la scindant en deux.
3. On recommence jusqu'à obtenir des classes d'équivalence.

— Les classes d'équivalence sont les états de l'automate minimal. On les notera $\{q_j,\dots,q_k\}$, d'après les états qui les composent.
— L'état initial de $\tilde{\mathcal{A}}$ est la classe contenant l'état initial de l'automate $\mathcal{A}$.
— Une classe $\{q_j,\dots,q_k\}$ est acceptante si elle contient au moins un état accepteur de l'automate original $\{q_j,\dots,q_k\} \cap F \neq \emptyset$.

B6. Établir les classes d'équivalence de Nérode pour l'automate $\mathcal{A}$.

Solution : Les états sont divisés en deux groupes :

$$P_0 = \{\{2,3\},\{0,1\}\}.$$

P_0	$\{2,3\}$	$\{0,1\}$
a	$\{2,3\}$	$\{0,1\}$
b	$\{2,3\}$	$\{0,1\} \cup \{2,3\}$

On affine donc la partie $\{0,1\}$.

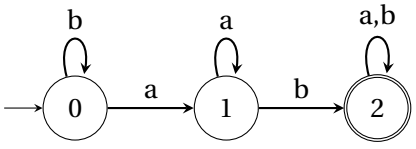
$$P_1 = \{\{2,3\},\{0\},\{1\}\}.$$

P_0	$\uparrow\{2,3\}$	$\downarrow\{0\}$	$\{1\}$
a	$\{2,3\}$	$\{1\}$	$\{1\}$
b	$\{2,3\}$	$\{0\}$	$\{2,3\}$

L'algorithme est fini, les classes sont trouvées.

B7. Dessiner l'automate minimal $\tilde{\mathcal{A}}$.

Solution :



C Programmation en OCaml

On utilise le type suivant pour représenter l'automate :

```
type automate = {
  n : int;
  initial : int;
  delta : int -> char -> int;
  final : int -> bool
}
```

Les classes sont représentées par des entiers de 0 à c. Une partitions des états en classes est un tableau de taille n : pour chaque état, on note dans la case correspondante le numéro de sa classe.

C8. Écrire une fonction de signature `p_zero : automate -> int array` qui crée la partition initiale de l'algorithme de Moore. Les accepteurs sont dans la classe 1 et les non-accepteurs dans la classe 0.

Solution :

```
let p_zero a =
  (* On crée les deux classes : les accepteurs  et les non-accepteurs *)
  Array.init a.n (fun i -> if a.final i then 1 else 0);;
```

C9. Écrire une fonction de signature `index : 'a -> 'a list -> int option` qui renvoie l'index de l'élément passé en paramètre le plus à gauche dans la liste s'il existe, `None` sinon.

Solution :

```
let index x liste =
  let rec aux lst k =
    match lst with
    | [] -> None
    | h :: t -> if h = x then Some k else aux t (k + 1)
  in
  aux liste 0
```

La signature d'un état i de l'automate est le triplet :
(classe de sa destination par a, classe de sa destination par b).

C10. Écrire une fonction de signature `signature : int -> 'a array -> automate -> 'a * 'a` qui renvoie la signature d'un état `i` de l'automate `a` dans l'état actuel de la partition.

Solution :

```
let signature i p a = (p.(a.delta i 'a'), p.(a.delta i 'b'))
```

Au cours de l'algorithme de Moore, lorsque la signature de deux états diffère, il est nécessaire de les placer dans deux classes différentes. Ainsi, à partir de la partition P_0, on va créer une fonction qui permet de créer la partition suivante comme le tableau suivante l'explique :

TABLE 1 – Exécution de la fonction `partition_suivante` depuis P_0 (première itération)

État i	$\delta(i,a)$	$\delta(i,b)$	Signature	Signatures déjà vues	Partition suivante
Initial	-	-	-	[]	`[| -1 ; -1 ; -1 ; -1 |]`
0	1	0	(1,0)	[(1,0)]	`[| 0 ; -1 ; -1 ; -1 |]`
1	1	2	(1,2)	[(1,0);(1,2)]	`[| 0 ; 1 ; -1 ; -1 |]`
2	3	2	(3,2)	[(1,0);(1,2);(3,2)]	`[| 0 ; 1 ; 2 ; -1 |]`
3	3	2	(3,2)	[(1,0);(1,2);(3,2)]	`[| 0 ; 1 ; 2 ; 2 |]`

C11. Écrire une fonction de signature `partition_suivante : automate -> 'a array -> int array` qui, à partir d'un tableau `p` représentant la partition actuelle où `p.(i)` est l'identifiant de la classe de l'état `i`, renvoie un nouveau tableau de partition plus fin.

Solution :

```
let partition_suivante a p =
  let n = a.n in
  let p_suiv = Array.make n (-1) in
  let sig_vues = ref [] in
  for i = 0 to n - 1 do
    let sig_i = signature i p a in
    let idx = index sig_i !sig_vues in
    match idx with
    | None ->
        let n_id = List.length !sig_vues in
        sig_vues := !sig_vues @ [ sig_i ];
        p_suiv.(i) <- n_id
    | Some j -> p_suiv.(i) <- j
  done;
  p_suiv
```

C12. Écrire une fonction de signature `minimser : automate -> automate` qui réduit l'automate en paramètre à l'automate minimal selon l'algorithme de Moore.

Solution :

```
let minimiser a =
  (* 1. Trouver la partition finale *)
  let p_init = Array.init a.n (fun i -> if a.final i then 1 else 0) in
  let rec calculer_partition p =
    let p_suiv = partition_suivante a p in
    if p_suiv = p then p else calculer_partition p_suiv
  in
  let partition = calculer_partition p_init in

  (* 2. Déterminer le nombre d'états de l'automate minimal *)
  let nb_etats_min =
    let max_id = ref (-1) in
    Array.iter (fun id -> if id > !max_id then max_id := id) partition;
    !max_id + 1
  in
```

```
  (* 3. Choisir un représentant pour chaque classe pour définir delta et final *)
  let repr = Array.make nb_etats_min (-1) in
  for i = 0 to a.n - 1 do
    repr.(partition.(i)) <- i
  done;

  (* 4. Construire l'automate minimal *)
  {
    n = nb_etats_min;
    initial = partition.(a.initial);
    delta =
      (fun id_classe c ->
        let q = repr.(id_classe) in
        partition.(a.delta q c));
    final = (fun id_classe -> a.final repr.(id_classe));
  }

(* Calcul de l'automate minimal *)
let a_min = minimiser ac;;

(* Affichage de la solution*)
let () =
  Printf.printf "Nombre d'états original : %d\n" a_min.n;
  Printf.printf "Nombre d'états minimal  : %d\n" a_min.n;
  for i = 0 to a_min.n - 1 do
    Printf.printf "L'état %d est acceptant ? %b\n" i (a_min.final i)
  done;
  for i = 0 to a_min.n - 1 do
    Printf.printf "Transistion de (%d,'a') -> %d\n" i (a_min.delta i 'a');
    Printf.printf "Transistion de (%d,'b') -> %d\n" i (a_min.delta i 'b')
  done
```