Automates

d'après un sujet de Nathanaël Fijalkow

Étienne Miquey
etienne.miquey@ens-lyon.fr

Nous allons étudier dans ce TP l'implémentation d'algorithmes classique sur les automates finis. Vous êtes censés maîtriser la plupart de ces algorithmes de façon théorique, voici l'occasion de voir si vous êtes aussi capable de les mettres en pratique. Par ailleurs, si vous n'êtes pas complétement au point avec les fonctions `map`, `do_list`, `init_vect`, `assoc`, etc ; c'est une bonne occasion de clarifier tout ça.

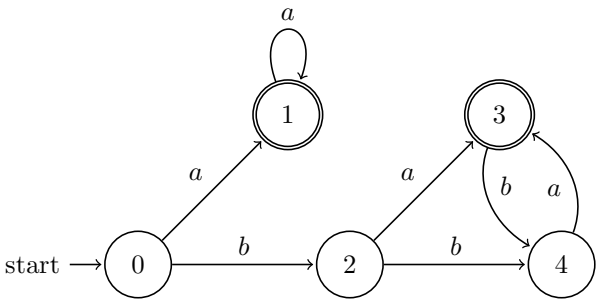
Un automate sera représenté par un enregistrement contenant la taille, l'état initial, les états finaux et la table de transition.

```
type automate = {
   taille : int ;
   initial : int ;
   transitions : (char * int) list vect ;
   final : bool vect } ;;
```

1 Automates déterministes

Les automates que nous allons considérer ici sont déterministes.

Question 1. Écrire une fonction `calcul_det` qui étant donné un mot et un automate supposé déterministe, détermine si l'automate accepte ce mot. Quelle est sa complexité ?



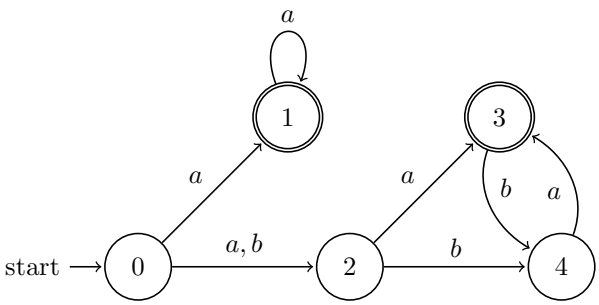
Question 2. Définir l'automate ci-dessus, et vérifier sur les exemples *aa*, *aba* et *bab* que la fonction `calcul_det` est correcte.

Question 3. Écrire une fonction accessible qui supprime les états inaccessibles d'un automate. Pour cela, on doit renuméroter les états, on pourra maintenir deux tableaux `tab_conv` et `tab_inv` qui gèreront la correspondance entre nouveaux et anciens états.

2 Automates non déterministes

Considérons à présent des automates non-déterministes, toujours avec le type `automate`.

Question 4. En vous appuyant sur la question 1, écrire une fonction `calcul_nondet` qui étant donné un mot et un automate, détermine si l'automate accepte le mot. Quelle est sa complexité ?



Question 5. Définir l'automate représenté ci-dessus, et vérifier sur les exemples *aa*, *aba* et *bab* que la fonction une fonction `calcul_nondet` est correcte.

3 Déterminisation

Pour déterminiser un automate, on passe par l'automate des parties : étant donné un automate non-déterministe $\mathcal{A} = (Q = \{0, \dots, n-1\}, 0, \delta, F)$, son déterminisé est l'automate $\bar{\mathcal{A}} = (2Q, \{0\}, \delta', F')$, où $\delta'(S, a) = \{q' \in Q \mid \exists q \in S, (q, a, q') \in \delta\}$ et $F' = \{S \subseteq Q \mid S \cap F = \}$.

Pour représenter les états de $\bar{\mathcal{A}}$, on va coder les sous-ensembles $S \subseteq \{0, \dots, n-1\}$ par des entiers. Par exemple, $\{0, 2, 3\}$ sera représenté par le tableau de booléens `[| true ; false ; true ; true |]`, et de manière équivalente par l'entier $2^0 + 2^2 + 2^3 = 13$.

Question 6. Écrire les fonctions de conversion `tab2int` et `int2tab`. La fonction `int2tab` prend en argument l'entier k à convertir et la taille n du tableau attendu.

Écrire également une fonction `list2tab` qui prendra en argument une liste d'entiers distincts et un entier n et renverra le tableau de booléen de taille n correspondant. En déduire la fonction `list2int` qui correspond.

Question 7. Écrire une fonction `determinise` qui calcule l'automate déterminisé.

Le nombre d'états de $\bar{\mathcal{A}}$ est exponentiel en le nombre d'états de $\mathcal{A}$; ceci rend impraticable la déterminisation dès que A est gros. En pratique, on préfère calculer seulement la partie accessible de $\bar{\mathcal{A}}$. En effet, souvent, le déterminisé a (beaucoup) moins que 2^n états. Cependant, dans certains cas cette borne est atteinte :

Question 8. Construire un automate non-déterministe reconnaissant $L_n = \mathcal{A}^* a \mathcal{A}^{n-1}$. Montrer que tout automate déterministe reconnaissant L_n possède au moins 2^n états.

4 Minimisation

Pour minimiser un automate (déterministe et complet), on va utiliser l'algorithme de Moore qui se base sur l'équivalence de Nérode. Étant donné $\mathcal{A} = (Q = \{0, \dots, n-1\}, 0, \delta, F)$, c'est la relation d'équivalence sur Q définie par

$$p \sim q \leftrightarrow \forall w \in A^*, (p \cdot w \in L(A) \leftrightarrow q \cdot w \in L(A)).$$

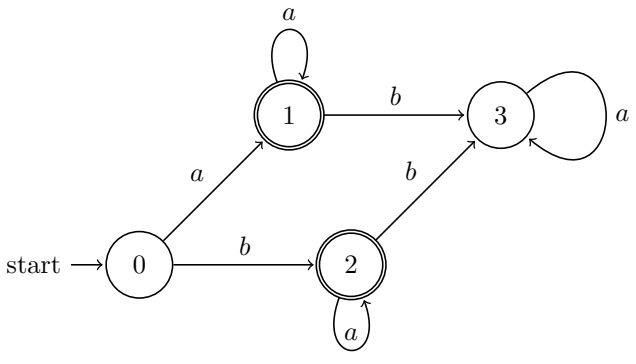
On la calcule par approximations successives : on définit les relations $\sim_k$ pour $k \in \mathbb{N}$ par

$$p \sim_k q \leftrightarrow \forall w \in A^{\leq k}, (p \cdot w \in L(A) \leftrightarrow q \cdot w \in L(A))$$

où $\mathcal{A}^{\leq k}$ est l'ensemble des mots de longueur au plus k. Vous vérifierez que $\sim = \sim_{n-2}$.

On va utiliser un tableau annexe `partition`, dont la valeur `partition.(i)` sera un état $j \leq i$ en relation avec i. Le but va être de séparer les états dits *distinguables*, en calculant successivement les représentants pour les relations $\sim_k$.

On se servira pour cela d'une fonction `classe` qui calcule le représentant minimal de la classe de i, et met à jour les valeurs des états considérés. Pour tester si deux états sont en relation, il suffira de calculer les représentants minimaux de leurs deux classes, et de les comparer.



Question 9. Écrire une fonction `minimise` qui calcule l'automate minimal. Tester-la sur l'automate ci-dessus.

5 Si vous vous ennuyez...

Vous pouvez vous entraîner à coder les automates obtenus par la méthode de Thompson qui font le lien avec les expressions rationnelles. Ou me demander, je devrais pouvoir vous indiquer tout un tas d'autres choses à coder sur le thème.