TP Option Info MP/MP* : Automates

1 Automates déterministes

Type automate

On définit le type automate par:

```
type automate = { n: int ; i : int  ; t : int list ;
                  gamma : (int*char*int) list };;
```

- `n` désigne le nombre d'états de l'automate, les états de l'automate sont numérotés de 0 à $n-1$.
- `i` est l'état initial.
- `t` est la liste des états terminaux/finaux.
- `gamma` est la liste des transitions, une transition (d, c, a) est à comprendre comme $d \xrightarrow{c} a$ d est l'origine de la transition, c et son étiquette et a son arrivée.

Nous supposerons que nos automates sont déterministes , c'est à dire que si $d \xrightarrow{c} a$ et $d \xrightarrow{c} a'$ sont deux transitions alors $a = a'$.

Construisons quelques exemples

Construire les automates

1. $\mathcal{A}_1$ reconnaissant les mots écrits avec des 0 et des 1 contenant un nombre pair de 0 et un nombre impair de 1.
2. $\mathcal{A}_2$ reconnaissant le langage défini par l'expression rationnelle $a + b^*c$.
3. $\mathcal{A}_3$ reconnaissant le langage défini par l'expression rationnelle $(aa + ab)^*ba^*$.

Calculons avec nos automates

Méthode 1

1. Écrire une fonction `chercheTransition :  int -> char -> (int*char*int)list ->int` telle que si `e` est un état, `c` un caractère et `t` une liste de transitions, `chercheTransition e c t` indique l'arrivée de la transition d'origine `e` et d'étiquette `c`. Dans le cas où une telle transition n'existe pas, la fonction renverra -1.
2. Écrire une fonction `calcul :  automate -> string -> bool` telle que `calcul a m` indique si le mot `m` est reconnu par l'automate `a`.

Méthode 2

Numérotation des caractères : Caml numérote les caractères de 0 à 255, On détermine le numéro d'un caractère avec la fonction `int_of_char`, la fonction réciproque étant :`char_of_int`.

1. Écrire une fonction `table_de_transition : automate -> int array array` telle que si `a` de type **automate** est la représentation en caml de $\mathcal{A}$, alors `table_de_transition a` est une matrice de taille `a.n`$\times$ 256 et que si $e \in Q$ et $c \in X$ alors

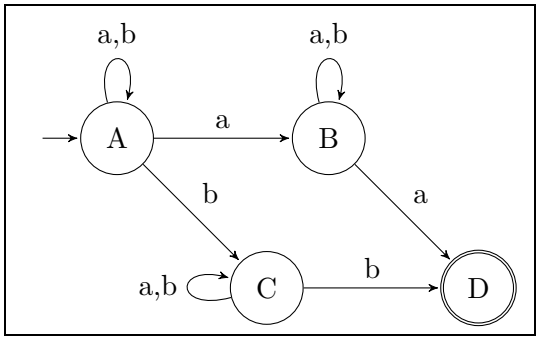
   `(table_de_transition a).(e).(int_of_char c) =` $\tilde{\gamma}(e,c)$ si $\tilde{\gamma}(e,c)$ est définie et (-1) sinon.

   Remarque: votre fonction ne devra lire qu'une seule fois la liste des transitions.

2. Réécrire la fonction `calcul : automate -> string -> bool`

   en utilisant la fonction `table_de_transition`.

2 Automates non déterministes



Remarque Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse.

Figure 1: Exemple d'automate non déterministe.

2.1 Opérations ensemblistes

Dans cette partie N désigne un entier naturel, et on va s'intéresser aux opérations dans $\mathcal{P}(\llbracket 0, N-1 \rrbracket)$.

Si $E \subset \llbracket 0, N-1 \rrbracket$ alors E représenté par un tableau de booléens de taille N suivant la règle: Si `t` représente E alors $\forall i \in \llbracket 0, N-1 \rrbracket$: `t.(i)` $= i \in E$ (c'est un booléen).

1. Écrire une fonction `appartient : int -> bool array -> bool` qui teste l'appartenance.
2. Écrire une fonction `vide : int -> bool array` qui construit la partie vide pour un N donné.
3. Écrire les fonctions d'union et d'intersection correspondants à cette représentation des ensembles.
4. Une fonction `ajoute : bool array -> bool array -> unit` telle que si `e1` et `e2` représentent deux parties de $\llbracket 0, N-1 \rrbracket$ alors `ajoute e1 e2` transforme `e1` en « `e1`$\cup$`e2`».
5. Écrire une fonction qui teste l'inclusion entre deux parties de $\llbracket 0, N-1 \rrbracket$

2.2 Calcul avec un automate non déterministe

Soit X un alphabet , un automate $\mathcal{A}$ est quadruplet (Q, I, T, Γ) où Q l'ensemble des états de $\mathcal{A}$ est un ensemble fini, $I \subset Q$ est l'ensemble des états initiaux , $T \subset Q$ est l'ensemble des états terminaux et $\Gamma \subset Q \times X \times Q$ est l'ensemble des transitions.

On remarque qu'un automate non déterministe peut avoir plusieurs états initiaux.

Si $e \in Q$ et $x \in X$ alors on note $\gamma(e,x) = \{d \in Q | (e,x,d) \in \Gamma\}$

On étend γ par si $E \subset Q$ et $x \in X$ alors $\gamma(E,x) = \bigcup_{e \in E} \gamma(e,x)$

X^* désigne l'ensemble des mots que l'on peut écrire sur l'alphabet X, ε désigne le mot vide. On étend alors γ par «fermeture transitive» à X^* par

- Si $E \subset Q$ alors $\gamma^*(E,\varepsilon) = E$
- Si $E \subset Q$, $m \in X^*$ et $x \in X$ alors $\gamma^*(E, m.x) = \gamma(\gamma^*(E,m),x)$

On définit le type automate par:

```
type automate_nd = { n_nd: int ; i_nd : bool array  ;
               t_nd : bool array ; gamma_nd : (int*char*int) list};;
```

1. Écrire une fonction `gamma : automate_nd -> int -> char -> bool array` qui réalise la fonction γ.
2. Écrire une fonction `gamma_e : automate_nd -> array bool -> char -> array bool` qui réalise l'extension de la fonction γ à $\mathcal{P}(Q)$.
3. Écrire une fonction `gamma_etoile :automate_nd -> array bool -> string -> array bool` qui réalise la fonction γ^*.
4. Écrire une fonction `calcule : automate_nd -> string -> bool` qui teste si un mot est reconnu par un automate non déterministe.

3 Déterminisation

Le calcul direct sur un automate non déterministe étant fastidieuse, on préfère remplacer $\mathcal{A}$ par un automate déterministe équivalent $\mathcal{A}'$.

$\mathcal{A}'$ est défini ainsi

$\mathcal{A}' = (\mathcal{P}(Q), I, \{E \subset Q | E \cap T \neq \emptyset\}, \{(E,x,\gamma(E,x) | E \subset Q \text{ et } x \in X\})$

Pour cela il faut numéroter les parties de Q.

Si $E \subset [\![0, N-1]\!]$ alors le numéro de E est $\sum_{i \in E} 2^i$

On remarque alors que la représentation de E sous forme de tableau de booléens est l'écriture en binaire de son numéro.

1. Écrire une fonction `numerote :bool array -> int` qui calcule le numéro d'un ensemble.
2. Écrire la fonction `partie : int -> int -> bool array` qui fait l'opération inverse, le premier paramètre est le nombre de cases que doit avoir le résultat.
3. Écrire une fonction `determinise : automate_nd -> automate` qui réalise le passage de $\mathcal{A}$ à $\mathcal{A}'$.

Remarques

- Les opérations logiques bits à bits en Caml sont `land`, `lor`, `lnot`, ...

  « Logical and, Logical or , ... »

- On peut rapidement calculer 2^i an Caml en utilisant les fonctions de décalages, pour cela on écrit `1 lsl i`

  Logical shift left