TP2 -- L3 -- Langage et Compilation

Automate fini

Dans ce TP, on utilisera JFLAP (Java Formal Languages and Automata Package), un logiciel qui permet de manipuler les notions de base de la théorie des automates et des langages formels.

Un premier automate déterministe

On considère l'AFD `A` défini par `({a,b}, {0,1,2}, δ, 0, {2})` où `δ`, la fonction de transition, est donnée par la table

	a	**b**
0	1	0
1	2	1
2		2

Pour définir un automate avec JFLAP, il faut :

- Lancer JFLAP en tapant simplement, sur une machine du département[1] , la commande : `jflap`
- Choisir l'entrée «Finite Automaton» du menu initial
- Dessiner le graphe de transition de l'automate :
  - Créer les états (2^{e} bouton de la barre d'outil)
  - Spécifier l'état initial et les états d'acceptation (1er bouton de la barre d'outil et clic droit sur l'état voulu)
  - Définir les transitions (3^{e} bouton)

> **Question :** Construire le graphe de transition de l'AFD `A`.

Question : Effectuer le calcul de l'automate sur l'entrée `babab` (sélectionner dans le menu «Input» l'option «Step by State»).

Question : Trouver trois mots acceptés par l'automate et trois mots rejetés (sélectionner dans le menu «Input» l'option «Multiple Run»).

Question : Décrire en français le langage reconnu par cet automate. En donner une expression régulière.

Un premier automate non déterministe

Récupérer le fichier afn1.jff qui décrit un AFN `A`. L'ouvrir avec JFLAP.

Question : Pourquoi l'automate `A` n'est pas déterministe ?

Question : Effectuer le calcul de l'automate sur l'entrée `aabb`. Donner la trace des deux calculs acceptant de l'automate `A` sur cette entrée.

Question : Donner tous les mots de longueur 2 qui sont rejetés par `A`.

Déterminisation de l'automate

À partir du graphe de transition de l'automate non déterministe

- Sélectionner dans le menu «Convert» l'option «Convert to DFA».

- Utiliser la commande «State Expandeur» (3ème bouton de la barre d'outil).

> **Question :** Construire un automate fini déterministe équivalent à l'automate `A`.

D'automate fini vers expression régulière

Récupérer le fichier er1.jff qui définit un AFD `A`.

> **Question :** Donner les six mots de longueur 8 qui sont acceptés par `A`.

On veut expliciter une expression régulière qui décrit le langage reconnu par l'automate `A`.

À partir du graphe de transition de l'automate

- Ajouter un état `source` avec une transition étiquetée par le mot vide (avec JFLAP, c'est par défaut le symbole λ, qu'on obtient en tapant immédiatement sur 'entrée' lors de l'ajout d'une transition) sur l'état initial.
- Ajouter un état `destination` avec des transitions étiquetées par le mot vide des états finaux vers cet état `destination`.
- L'état `source` devient l'unique état initial et l'état `destination` l'unique état final.

Puis

- Sélectionner dans le menu «InputConvert» l'option «Convert FA to RE».

> **Question :** Donner une expression régulière qui décrit le langage reconnu par l'automate `A`.

D'expression régulière vers automate fini

JFLAP permet de construire un automate avec ε-transitions qui reconnaît le langage associé à une expression rationnelle donnée (c'est l'algorithme de Thompson qui est appliqué) :

- Ouvrir une nouvelle fenêtre en choisissant l'entrée «Regular Expression» du menu initial.
- Spécifier l'expression rationnelle voulue.
- Convertir l'expression rationnelle en automate (option «Convert to NFA» puis utiliser la commande «Do Step» pour visualiser les différentes étapes de la construction).

Question : Donner une expression régulière qui décrit l'ensemble des mots qui ont pour suffixes : `a` ou `ab`

Question : Construire un automate avec ε-transitions qui reconnaît le langage associé.

Question : Le déterminiser. Pour chacun des états de cet automate, caractériser les mots qui étiquettent les chemins menant à cet état.

Propriété de clôture

Clôture par union

On considère sur l'alphabet `{a,b}`, le langage L_1 formé de tous les mots contenant exactement deux `a` et le langage L_2 formé de tous les mots contenant exactement trois `b`.

Question : Construire un AFD A_1 qui reconnaît le langage L_1 et un AFD A_2 qui reconnaît le langage L_2 .

Question : Combiner les deux automates A_1 et A_2 pour construire un automate A_U avec ε-transitions qui reconnaît le langage $L_1 \cup L_2$.

Question : Déterminiser l'automate A_U . Que mémorise chacun des états de cet automate ? Pouvez vous trouver un automate déterministe reconnaissant le même langage avec moins d'états ?

Clôture par étoile

Question : Construire un AFD `A` (à 6 états) qui reconnaît le langage décrit par `101+001*`.

Question : À partir de l'automate `A` construire sans ajouter d'états un automate avec ε-transitions qui reconnaît le langage décrit par `(101+001*)*`.

Question : Vérifier que les mots suivants sont bien reconnus : `00100, 1010011, λ` . Et enregistrer votre automate dans le fichier `automateEtoile.jff`.

On souhaite vérifier que l'automate construit reconnaît bien le langage décrit par `(101+001*)*`. L'idée est de construire un deuxième automate avec ε-transitions à partir de l'expression rationnelle `(101+001*)*` comme vu précédemment. Puis de déterminer l'équivalence entre ces deux automates (sélection de l'option «Compare Equivalence» dans le menu «Test»).

Question : Tester si l'automate `automateEtoile` reconnaît effectivement le langage décrit par `(101+001*)*`.

Clôture par concaténation

Question : Construire un AFD A_1 (à 3 états) qui reconnaît le langage décrit par `01(11)*` et un AFD A_2 (à 2 états) qui reconnaît le langage décrit par `(00+10)*`.

Question : À partir de A_1 et A_2, construire un automate $A_\otimes$ avec ε-transitions qui reconnaît le langage $L_1 \cdot L_2$.

Question : Déterminiser l'automate $A_\otimes$.

Pump up the jam

JFLAP permet de jouer au lemme de la pompe (cliquez sur regular pumping lemma dans le menu principal). Vous pouvez en particulier jouer au lemme de la pompe pour des exemples qu'on a vu au premier TD.

Question : Pompez avec JFLAP.

---

[1] La commande `jflap` est un simple script shell qui fait :

```
#!/bin/sh
exec java -jar /usr/local/jflap/JFLAP.jar "$@"
```

Vous pouvez facilement disposer de cette commande en téléchargeant JFLAP.jar.