TP : Automates et langages

1 Présentation de JFLAP

(*Java Formal Languages and Automata Package*) est un outil graphique pédagogique qui aide à la compréhension des concepts de base des langages formels et de la théorie des automates.
JFLAP permet de créer et de simuler un certain nombre d ?automates dont les automates à états finis et les automates à pile. De plus, JFLAP offre des passerelles vers et depuis les expressions régulières et les grammaires.
L'application originale JFLAP a été programmée en C++ et fonctionne sous un environnement X-Window (Unix, Linux, ...). En raison de son succès en tant qu'outil pédagogique en théorie des langages formels et des automates, JFLAP a été réécrit en java de manière à pouvoir fonctionner sur n'importe quelle plateforme.
Pour utiliser JFLAP, qui est une archive `jar`, il faut la télécharger sur votre machine (mettez-la sur le bureau par exemple ou dans un répertoire accessible sur votre machine).
Ensuite pour lancer l'application,
– depuis un système OS X (Mac), double-cliquez sur l'icône du fichier
– depuis une machine sous Linux utilisez un terminal et tapez `java -jar JFLAP.jar` (ceci marche aussi bien-sûr depuis un système OS X).

Au démarrage, l'application vous demande ce qui vous intéresse, et vous répondrez que c'est les automates finis (premier choix dans la liste).
Maintenant avant de faire quoi que ce soit, il faut définir un automate sur lequel travailler. Ceci se fait en dessinant cet automate à l'aide de la palette d'outils disponibles juste au dessus du canevas (zone blanche). Le principe est de sélectionner un outil (déplacement, états, transitions, détruire un objet, undo, redo) puis de cliquer sur le canevas pour utiliser l'action sélectionnée. En mode déplacement, vous disposez aussi du menu contextuel (clic droit) sur un objet pour préciser certaines choses (état initial, final, etc).
Attention, par défaut, le symbole λ correspond au mot vide, allez dans Preferences (ou Parametres) pour demander l'utilisation d'ϵ.
Quand on définit un automate, on n'a pas besoin ici de préciser le langage. Quand on définit plusieurs lettres permettant de passer d'un état à un autre, on peut factoriser sur la même transition, en indiquant les caractères les uns à la suite des autres (mais sans les séparer par une virgule, que JFLAP interprète sinon comme un caractère de la transition).

2 Automates associés à des langages

1. Dessiner l'automate $\mathcal{A} = (\Sigma, E, I, F, \Delta)$ avec $\Sigma = \{a, b\}$, $E = \{e1, e2\}$, $I = e1$, $F = \{e2\}$ et $\Delta = \{(e1, a, e1), (e1, b, e2), (e2, a, e2), (e2, b, e1)\}$ Quel langage reconnaît cet automate ? Pour vérifier, entrer des mots de ce langage dans l'automate et regardez s'ils sont acceptés systématiquement : menu *Input* → *Fast Run*.

   Quel est dans cet automate le chemin d'étiquette *aabba* ?

2. Dessiner un automate *déterministe* qui reconnaisse la langage $L = \{a, aab, aaaab, b, bbaa\}$. Enregistrez-le sous le nom `L.jff`.

Vérifiez que votre automate correspond exactement à L en testant chacun de ces mots et en vérifiant que l'automate les accepte.
Vous pouvez faire cela de plusieurs manières différentes grâce au menu `Input` :
– **Step by state** : saisissez préalablement la chaîne à tester dans la zone d'édition `Input String` ; l'action `Step by state` permet d'effectuer une reconnaissance pas à pas du mot, c'est-à-dire lettre après lettre.

– **Fast run** : saisissez préalablement la chaîne à tester dans la zone d'édition `Input String` ; l'action `Fast run` vous dit si la chaîne est acceptée ou refusée par l'automate.

– **Multiple run** : ceci permet de tester l'un après l'autre plusieurs mots et de vous présenter une liste de résultats.

3. Pouvez-vous supprimer des états (tout en continuant à reconnaître le même langage) ? Une fois les modifications faites vous pouvez le testez en utilisant l'option `Multiple Run` du menu `Input` : indiquez tous les mots de L ainsi que certains mots qui ne sont pas dans L, afin de vérifier aussi que l'automate que vous avez modifié n'accepte pas de mots supplémentaires.
4. Un automate **complet** est un automate $\mathcal{A}$ tel que pour tout état E de $\mathcal{A}$ et tout symbole $l \in \Sigma$ il existe une transition $(e, l, e') \in \Delta$. Pouvez-vous rendre votre automate complet en ajoutant au plus un état ?
5. Sans rajouter d'état, mais seulement en en supprimant (et en jouant sur les transitions) peut-on transformer l'automate donné dans le fichier `PlusieursEtatsDeSortie_A` en un automate n'ayant qu'un état de sortie (et acceptant le même langage) ? Pourquoi la même astuce ne fonctionne-t-elle pas pour les automates des fichiers `PlusieursEtatsDeSortie_B` et `PlusieursEtatsDeSortie_C` ?

3 Automate pour encoder un lexique

Dans les applications de traduction automatique, l'ensemble des mots d'un langage forme un lexique qui peut être organisé sous la forme d'un automate à états finis, où chaque arc correspond à une lettre. L'objectif est ici de construire un automate à états finis capable de représenter le langage de l'extrait de lexique suivant :

Forme fléchie	Base	Cat.	Morphèmes
ami	ami	N	ms
amis	ami	N	mp
amie	ami	N	fs
amies	ami	N	fp
amant	amant	N	ms
amants	amant	N	mp
amante	amant	N	fs
amantes	amant	N	fp

Cet automate pourra être utilisé en *génération* (il énumérera alors toutes les formes possibles) ou en *reconnaissance* (il *acceptera* alors les formes qui font partie du lexique, et *rejettera* les autres).

5. Si l'on mesure la taille d'un automate par le nombre de ses états + le nombre de ses transitions, quelle est la taille de cet automate ?
6. Quel est le gain par rapport à un stockage traditionnel du lexique sous la forme d'un tableau de caractères ?
7. Explorez la possibilité de donner un label aux états pour indiquer pour chaque état final le genre et le nombre du mot correspondant (information en colonne 4 du tableau ci-dessus), par exemple indiquez `ms` (masculin singulier) pour le mot `ami`, etc

4 Automate pour un distributeur

Une branche d'application importante des automates finis déterministes se situe en électronique et robotique, où les appareils conçus en milieu industriel sont souvent modélisés comme des

machines à états. Les automates (ou des modèles étendant les possibilités des automates, par exemple, les *réseaux de Pétri*) sont utilisés lors de la conception de ces appareils : cette phase de conception traduit par un automate sur papier le comportement que doit adopter l'appareil (ce dessin est souvent réalisé sous une forme normalisée, appelée *Grafcet*[1]). Une fois l'automate établi, l'appareil est ensuite réalisé sous forme analogique – circuit électronique avec résistances, condensateurs – ou numérique – microcontrôleur. Ces systèmes sont très répandus en raison d'une grande fiabilité et d'une réinitialisation rapide en cas de panne (voir aussi ce lien [2]).

Pour votre première machine industrielle, nous allons vous confier la conception d'un distributeur de boissons chaudes. Comme l'entreprise IG+ ne vient que fraîchement de vous recruter, nous vous proposons la modélisation associée à notre appareil le plus simple, dont les spécifications suivent :

- il ne distribue que deux types de boissons : un café expresso (30c) et un café long (50c). Chaque boisson se commande par un bouton spécifique ;
- il n'accepte que des pièces de 10c, 20c et 50c ;
- dès qu'il reçoit plus que la somme du produit le plus cher (50c donc), il rend l'argent dans le bac de récupération de la monnaie [3] ;
- l'appareil ne comporte pas de bouton pour rendre l'argent si l'utilisateur change d'avis (il ne veut plus de café) ;
- après avoir inséré l'argent, le client presse sur un des deux boutons pour indiquer quel produit il veut (café simple ou long). Si la bonne somme d'argent a été introduite, le produit lui est délivré, sinon la machine attend d'autres pièces et "oubli" la commande (mais pas l'argent déjà inséré !) ;
- une fois le produit délivré, la machine attend le prochain client.

Bien-sûr, pour modéliser ce distributeur, certains symboles étiquetant des transitions peuvent correspondre à des actions du distributeur. Nous **imposons** que les symboles 1,2,5 correspondent resp. à l'introduction de 10c,20c,50c par le client, que les symboles `S,L` (majuscules) correspondent à l'appui sur le bouton `expresso simple` et `café long` ; que les symboles `s` et `l` (lettre [el]) correspondent à la distribution d'un café simple, d'un café long ; que le symbole **r** correspondent au rendu de tout l'argent introduit quand le client dépasse la somme de 50c, que les symboles `d,v,c` correspondent au rendu d'une pièce de 10c, 20c ou 50c. Nous imposons aussi `un seul état final`, correspondant à une distribution réussie, et renvoyant à l'état initial par une transition ϵ.

1. Dans un premier temps proposez une modélisation qui ne rend pas la monnaie (mais donne du café !). Ici on gère quand même le dépassement du seuil de 50c provoquant le rendu des pièces au client.
2. Encodez votre appareil dans JFLAP et essayez les séquences d'actions suivantes pour vérifier son comportement et notez celles qui sont *acceptées* par l'automate :
   - le client introduit 50c puis 20c, se voit rendre son argent, puis appuie sur le bouton `expresso simple` (mot `52rS`)
   - le client introduit une pièce de 20c, une pièce de 10c, une pièce de 20c (dans cet ordre), appuie ensuite sur le bouton `expresso simple` et reçoit un café simple (mot `212Ss`)

1. http ://fr.wikipedia.org/wiki/Grafcet
2. partie *Analyse du comportement d'un robot* de la page `www.planete-sciences.org/robot/wikibot/index.php/Intelligence_artificielle`
3. et oui, ça implique que pour obtenir un café long, il doit avoir la monnaie – ce n'est que le distributeur le plus simpliste de notre gamme

– le client introduit 20c, puis 20c, puis presse le bouton `café long` (mot `22L`)

3. Sauvegardez votre automate (une fois les ajustements éventuels effectués) sous le fichier `cafe1.jff` ;
4. Comment modifier votre automate pour qu'il rende aussi la monnaie si l'on commande un café expresso après avoir donné plus d'argent que les 30c (mais au plus 50c, sinon l'appareil rend tout l'argent dans le bac). Une fois l'automate modifié, enregistrez-le sous le nom `café2.jff` ;