Travaux pratiques de mathématiques appliquées

Automates finis

Durée prévue : 3 H

1. Introduction

1.1. Objectifs

Les objectifs principaux de ce laboratoire sont de :

- manipuler les notions de langages formels, grammaires et automates finis au moyen de l'outil JFLAP ;
- explorer l'équivalence entre grammaires régulières et automates finis ;
- convertir un automate non déterministe en un automate déterministe équivalent ;
- mettre en œuvre la notion d'automate dans un programme qui implémente un fonctionnement d'un feu tricolore .

Ce laboratoire utilise et met en pratique les notions expliquées dans les chapitres 8 et 9 du cours théorique.
L'étudiant est supposé avoir révisé la matière de ces deux chapitres avant le début du laboratoire !

1.2. Evaluation

Les séances de travaux pratiques ont un caractère essentiellement formatif. Les exercices proposés dans cet énoncé ne seront pas notés. Néanmoins, il est fondamental que vous tentiez d'en résoudre un maximum par vous-même, ceci afin d'acquérir une bonne compréhension et une bonne maîtrise de la matière du cours.

Si vous éprouvez des difficultés, n'hésitez pas à solliciter l'aide de votre responsable de laboratoire. Vous pouvez également collaborer avec vos condisciples pour concevoir les solutions. Assurez-vous cependant que vous avez compris les notions mises en pratique. Ne vous contentez pas de recopier une solution sans la comprendre : vous devez être capable de la recréer par vous-même.

1.3. JFLAP et JFLAP Book

Durant ce laboratoire, nous utiliserons l'outil JFLAP « Java Formal Languages and Automata Package » présenté lors du cours théorique pour explorer de manière interactive les notions de langages formels, grammaires et automates finis.

Les exercices proposés dans la section 2 sont tirés de l'édition 2006 du livre « JFLAP – An Interactive Formal Languages and Automata Package » de Susan H. Rodger et Thomas W. Finley. Nous nous référerons à ce livre en parlant du « JFLAP Book ».

Toutes les informations et ressources nécessaires concernant JFLAP sont disponibles sur l'espace HELMo Learn du cours.

2. Exercices avec JFLAP - Automates finis

2.1. Introduction

Cette partie du laboratoire est basée sur les chapitres 1 et 2 du livre « JFLAP Book ». Vous trouverez dans ces chapitres une description des outils proposés par JFLAP pour manipuler les automates finis.

- Section 1.1 – Création/modification d'un automate fini.
- Section 1.2 – Simulation de l'automate pour une ou plusieurs entrées.
- Section 1.3 – Création/simulation d'un automate fini non déterministe.
- Section 1.4 – Analyse des propriétés d'un automate (équivalence, non-déterminisme, etc.).
- Section 2.1 – Conversion d'un NFA vers un DFA (« Subset Construction Algorithm »).

Vous pouvez également regarder les vidéos introductives sur HELMo Learn.

2.2. Concevoir un automate qui reconnaît un langage donné

Alors que les grammaires permettent de générer les mots d'un langage (*système générateur*), les automates finis permettent quant à eux de déterminer l'appartenance d'un mot à un langage (*système reconnaisseur*).

Le but de l'exercice qui suit est vous exercer à construire un automate capable de reconnaître un langage régulier donné.

Exercice 1.1 du livre « JFLAP Book »

- Pour chaque langage décrit ci-dessous, construisez un automate fini <u>déterministe</u> qui reconnaît ce langage.
  a) Le langage sur $\Sigma = \{a\}$ de tous les mots contenant un nombre impair de 'a'.
  b) Le langage sur $\Sigma = \{a\}$ de tous les mots contenant un nombre pair de 'a'.
  c) Le langage sur $\Sigma = \{a, b\}$ de tous les mots contenant un nombre pair de 'a' et un nombre impair de 'b'.
- Validez que votre automate est correct en simulant son fonctionnement pour un ensemble de mots d'entrée. Vérifiez que les mots qui font partie du langage sont acceptés et que ceux qui n'en font pas partie sont rejetés.
  [Suggestion : utilisez l'outil « Multiple Run » proposé par JFLAP.]

Exercices supplémentaires :

d) Le langage sur $\Sigma = \{a, b\}$ de tous les mots contenant un nombre pair de 'a' et au moins trois 'b'.

e) Le langage sur $\Sigma = \{a, b\}$ de tous les mots dans lesquels les nombres de 'a' et de 'b' sont soit tous les deux pairs, soit tous les deux impairs.

2.3. Valider et modifier un automate

Exercice inverse du précédent : êtes-vous capable de déterminer le langage reconnu par un automate donné ? Si le langage est connu, l'automate reconnaît-il effectivement ce langage et pas un autre ? Si tel n'est pas le cas, pouvez-vous modifier l'automate pour qu'il remplisse correctement sa fonction ?

Exercice 1.2 du livre « JFLAP Book »

- Soit des automates finis pour lesquels l'auteur de chaque automate donne une description du langage qu'il est supposé reconnaître, ce qui n'est pas le cas !
- Pour chaque automate,
  (1) déterminez 6 chaînes d'entrée qui :
  - soit font partie du langage, mais ne sont pas acceptées par l'automate,
  - soit sont acceptées par l'automate alors qu'elles ne font pas partie du langage ;
  (2) modifiez l'automate afin qu'il reconnaisse effectivement le langage décrit ;
  (3) vérifiez votre solution en vous assurant que les chaînes d'entrée identifiées à l'étape (1) sont à présent correctement acceptées ou rejetées par l'automate.

  a) L'automate du fichier « `ex1.6a` » reconnaît le langage des mots sur l'alphabet $\Sigma = \{a, b\}$ contenant exactement deux 'b'.

  b) L'automate du fichier « `ex1.6b` » reconnaît le langage des mots contenant un nombre de 'a' divisible par 2 ou par 3.

Exercices supplémentaires :

c) L'automate du fichier « `ex1.6c` » reconnaît le langage des mots sur l'alphabet $\Sigma = \{a, b, c\}$ contenant au moins trois 'b' ou au moins trois 'c'.

d) L'automate du fichier « `ex1.6d` » reconnaît le langage des mots sur l'alphabet $\Sigma = \{a, b, c\}$ contenant au moins deux 'b' ou au moins trois 'c'.

Exercice 1.3 du livre « JFLAP Book »

- Soit un automate fini pour lequel on décrit le langage qu'il doit reconnaître.
- Déterminez les modifications à effectuer dans l'automate fourni pour qu'il reconnaisse effectivement le langage décrit. Effectuez les modifications dans JFLAP.
- Validez le fonctionnement correct de l'automate au moyen d'une série de chaînes d'entrée. Vérifiez que les mots qui font partie du langage sont acceptés et ceux qui n'en font pas partie sont rejetés.

  a) L'automate du fichier « `ex1.6e` » reconnaît le langage des mots contenant un nombre impair de 'a'. Nous souhaitons qu'il reconnaisse le langage des mots contenant un nombre pair de 'a'.

  b) L'automate du fichier « `ex1.1a` » reconnaît le langage des mots contenant un nombre quelconque de 'a', suivis par un nombre impair de 'b'. Nous souhaitons qu'il reconnaisse le langage des mots contenant un nombre de 'a' non nul, suivis par un nombre impair de 'b'.

c) L'automate du fichier « `ex1.6f` » reconnaît le langage des mots sur l'alphabet $\Sigma = \{a, b, c\}$ constitués par la répétition un nombre quelconque de fois d'une séquence contenant un 'a', suivi par un nombre impair de 'b', suivis par un 'c'. Nous souhaitons qu'il reconnaisse les mots où, à la place d'un seul 'c', une séquence peut contenir un nombre pair de 'c'.
Note : pour cet exercice, il peut être plus facile de créer en premier lieu un automate non déterministe.

2.4. Équivalence entre grammaires régulières et automates finis

Nous avons vu au cours théorique qu'il y a une équivalence entre les langages réguliers, les grammaires régulières, les expressions régulières, les maîtres-diagrammes syntaxiques et les automates finis. Tous ces formalismes permettent de décrire, générer ou reconnaître la même classe de langages.

Dans cet exercice, nous allons explorer l'équivalence entre les grammaires régulières et les automates finis et déterminer comment on passe de l'un à l'autre.

Exercice 3.5 du livre « JFLAP Book »

- Convertissez chaque grammaire régulière définie dans un des différents fichiers listés ci-dessous vers un automate fini.
  a) `ex3.rg2fa-a`
  b) `ex3.rg2fa-b`
  c) `ex3.rg2fa-c`
- L'outil « Convert Right-Linear Grammar to FA » de JFLAP permet de réaliser la construction de l'automate qui correspond à la grammaire de deux manières :
  - **Méthode manuelle** : choisir une production dans la grammaire puis créer la transition correspondante de l'automate. JFLAP valide que chaque transition créée est correcte.
  - **Méthode automatisée** : « Create Selected » crée la transition qui correspond à la production choisie, « Show All » crée toutes les transitions manquantes.

  La méthode automatisée doit uniquement servir à soutenir votre apprentissage. Elle vous permet de vous familiariser avec le processus de conversion et de valider que l'automate construit est correct. Vous devez être capable d'effectuer la conversion manuellement !

Exercices supplémentaires :

d) `ex3.rg2fa-d`
e) `ex3.rg2fa-e`
f) `ex3.rg2fa-f`

Ces trois grammaires contiennent des productions avec des symboles terminaux multiples.

Exercice 3.6 du livre « JFLAP Book »

- Convertissez chaque automate fini défini dans un des différents fichiers listés ci-dessous vers une grammaire régulière.
  a) `ex3.fa2rg-a`
  b) `ex3.fa2rg-b`
  c) `ex3.fa2rg-c`
- L'outil « Convert to Grammar » de JFLAP automatise complètement le processus de génération de la grammaire. Il n'est pas possible d'introduire les productions manuellement. Dès lors, nous vous conseillons de réaliser la conversion « à la main » sur papier ou dans un éditeur de texte, puis de vérifier si votre solution est correcte avec JFLAP.

Exercices supplémentaires :

d) `ex3.fa2rg-d`
e) `ex3.fa2rg-e`
f) `ex3.fa2rg-f`
g) `ex3.fa2rg-g`

Les deux derniers automates contiennent des transitions sur des symboles terminaux multiples.

2.5. Équivalence de langages – Exercice facultatif

Il n'est pas évident de déterminer si deux langages sont équivalents à partir d'une description, d'une grammaire, d'une expression régulière ou d'un automate caractérisant chacun des deux langages.

En revanche, il existe un algorithme général qui permet de déterminer l'équivalence de deux automates. Cet algorithme est une des fonctionnalités proposées par JFLAP (« Test > Compare Equivalence »).

Par conséquent, pour déterminer si deux langages sont équivalents, on peut construire un automate qui reconnaît chacun des deux langages, puis tester l'équivalence des deux automates. Si les deux automates sont équivalents, cela signifie qu'ils reconnaissent le même langage !

Exercice 1.4 du livre « JFLAP Book »

- Considérons les cinq langages réguliers sur l'alphabet $\Sigma = \{a, b\}$ définis comme étant l'ensemble des mots pour lesquels :
  a) 'aa' n'est pas une sous-chaîne ;
  b) 'aa' et 'aaa' ne sont pas des sous-chaînes ;
  c) 'aa' et 'aba' ne sont pas des sous-chaînes ;
  d) 'aa' et 'abaa' ne sont pas des sous-chaînes ;
  e) 'aaa' et 'aab' ne sont pas des sous-chaînes ;
- Séparez ces cinq langages en groupes qui correspondent selon vous au même langage.
- Construisez les cinq automates qui reconnaissent chacun de ces cinq langages, puis vérifiez si vos suppositions de regroupement étaient correctes en testant l'équivalence des automates.

2.6. Automates non déterministes

Nous terminons cette section consacrée aux automates par une mise en pratique de l'algorithme SCA « Subset Construction Algorithm » qui permet de transformer un automate non déterministe en un automate déterministe équivalent (mais pas forcément minimal).

Exercice 2.1 du livre « JFLAP Book »

- Convertissez l'automate non déterministe défini dans chacun des fichiers suivants en un automate déterministe équivalent au moyen de l'algorithme SCA.
  a) `ex2-nfa2dfa-a`
  b) `ex2-nfa2dfa-b`
  c) `ex2-nfa2dfa-c`
- L'outil « Convert to DFA » de JFLAP permet de valider la construction manuelle de l'automate déterministe étape par étape, ou de réaliser la conversion de manière partiellement (« State Expander ») ou entièrement automatisée (« Complete »).
  La méthode automatisée doit uniquement servir à soutenir votre apprentissage. Elle vous permet de vous familiariser avec le processus de conversion et de valider que l'automate construit est correct. Vous devez être capable d'effectuer la conversion manuellement !

Exercices supplémentaires :

a) `ex2-nfa2dfa-d`
b) `ex2-nfa2dfa-e`
c) `ex2-nfa2dfa-f`

Facultatif - Exercice 2.3 du livre « JFLAP Book »

- Considérons l'automate déterministe défini dans le fichier « `ex2-dfa2nfa` ».
  Cet automate est le résultat de la conversion d'un automate non déterministe au moyen de l'algorithme SCA. Les labels associés aux différents états indiquent les états du NFA de départ. Par ailleurs, le NFA initial ne possédait pas de transitions sur ε (transitions vides).
- Reconstruisez le NFA initial.

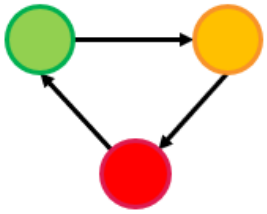
3. Feu tricolore

3.1. Introduction

Lors du cours théorique, nous avons illustré la définition d'un mécanisme à l'aide du feu tricolore.

En Belgique, le fonctionnement d'un feu de signalisation tricolore répond de base à la séquence :

vert → orange → rouge → vert → orange → rouge → …



Feu tricolore (= mécanisme à trois états)

Dans ce premier exercice, nous allons commencer par implémenter ce mécanisme.

Exercice 3.1 – Feu tricolore - Mécanisme

- Dans le projet « 2425_B1MATH_LF_GRM_FSM » importé dans Eclipse lors du laboratoire 4 – Langages formels et grammaires, créez le package « feuTricolore » dans le dossier source `src`. Ajoutez-y le fichier « FeuTricoloreMecanisme.java ».
- Dans la classe « FeuTricoloreMecanisme » du package « feuTricolore », complétez la méthode suivante :

  `public void changerEtat()`

  → Change l'état du feu tricolore en respectant l'ordre prévu.
- Testez que votre méthode fonctionne correctement en exécutant la fonction principale fournie.

Dans certaines situations, la séquence de fonctionnement d'un feu de signalisation tricolore peut être modifiée pour passer en mode « orange clignotant » (= orange → éteint → orange → éteint …)

Dans notre modélisation, le passage du mode normal au mode orange clignotant se fera à l'aide d'un interrupteur « Maintenance (on/off) ».

Nous allons modéliser le fonctionnement de ce feu tricolore au moyen d'un automate fini.

L'automate possédera 5 états :

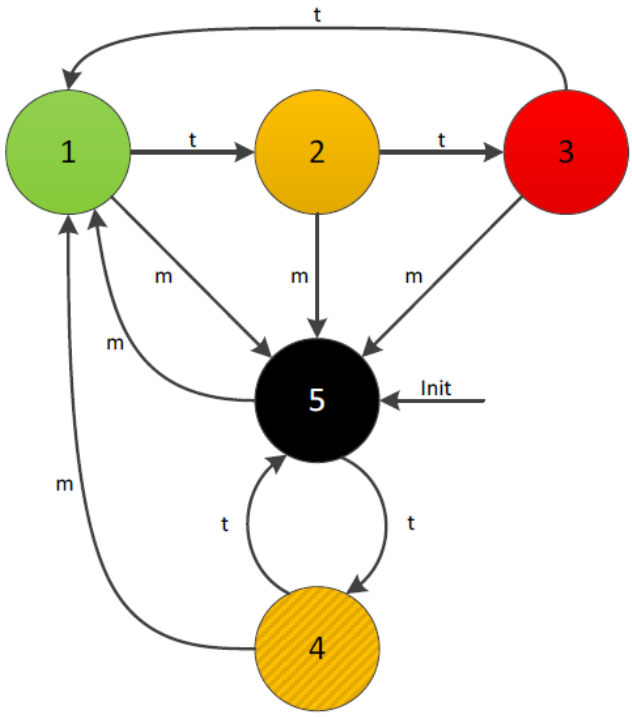
- Etat 1 : la lampe verte est allumée ;
- Etat 2 : la lampe orange est allumée, en mode continu ;
- Etat 3 : la lampe rouge est allumée ;
- Etat 4 : la lampe orange est allumée, en mode clignotant ;
- Etat 5 : toutes les lampes sont éteintes.

Cet automate réagira à deux inputs :

- L'input 't' (time tick) indique qu'il faut passer à l'état suivant de la séquence ;
- L'input 'm' (maintenance) permet d'entrer ou de sortir du mode clignotant.

L'automate démarre dans l'état 5.

Voici le schéma correspondant :



Exercice 3.2 – Feu tricolore – Automate fini

- Au package « feuTricolore » que vous venez de créer dans le projet « 2425_B1MATH_LF_GRM_FSM », ajoutez le fichier « FeuTricoloreFSM.java ».
- Dans la classe « FeuTricoloreFSM » du package « feuTricolore », complétez les méthodes suivantes :

  `public void changerEtat()`

  → Change l'état du feu tricolore en respectant l'ordre prévu, sous l'effet de l'input 't'.

  `public void basculerMaintenance()`

  → Change le mode de fonctionnement du feu tricolore ,en réponse à l'utilisation du bouton « Mode maintenance » (input 'm' sur le schéma).

- Testez que votre méthode fonctionne correctement en exécutant la fonction principale fournie ; le bouton « Mode maintenance » permet de basculer le feu d'un mode à l'autre.

Bon travail !