Modélisation et Vérification
TD Modélisation & FSM Test Generation Methods

Asma BERRIRI

Le TP donnera lieu à la rédaction d'un petit fichier pdf contenant les réponses aux questions. Pour cela utiliser un éditeur quelconque et classer les réponses par numéro. Le but est d'obtenir un texte court permettant de vérifier vos acquis.

Le fichier portera votre nom suivi du nom du TD/TP en l'occurrence ici TDFSM, exemple : martinTDFSM.pdf

Le fichier sera envoyé à asma.berriri@universite-paris-saclay.fr

Les exercices sont indépendants et peuvent être traités dans n'importe quel ordre.
Il vous est demandé de rédiger vos réponses avec soin et rigueur.

1 Notions de bases (rappels et/ou introduction)

Lorsqu'on observe ou imagine le comportement d'un système (quel qu'il soit), on a une suite des différents états du système. Les changements d'états ou transitions entre états peuvent être dus à des événements externes ou internes au système. Une suite d'états décrit un comportement, donc des ransitions entre des états.

Modélisation. En modélisation et en construction de logiciels, on prévoit le comportement des logiciels. On fait pour cela des modèles. On modélise ainsi ce qu'on va construire (aussi bien les données que les traitements). On peut ainsi prédire les comportements corrects et incorrects (bugs) ; il faut pour cela que les modèles soient les plus précis (mathématiquement) possibles.

Un modèle simule mathématiquement le (comportement d'un) système. En ce sens un automate à états sert de modèle d'un système ; il permet d'en simuler le comportement. Un logiciel est un cas particulier de système ; en effet la notion de système est plus général et un système peut contenir du matériel, du logiciel, et l'interaction avec l'humain.

Automates à états. Un automate à états est défini formellement par un 5-uplet : $< S, I, \delta, S_0, S_f >$

- S: un ensemble fini d'états
- I: un alphabet d'actions ou d'étiquettes
- δ: une relation de transition définie sur $S \times I$ avec $\delta : S \times I \to S$
- $S_0 \in S$: un état initial
- $S_f \in S$: un ou des états finaux

Automate déterministe. Lorsque la relation de transition est une fonction (ie. $S \times I \to S$ au lieu d'une relation de $S \times I \overset{[}{\leftrightarrow}]S$, alors l'automate est déterministe; sinon l'automate est non-déterministe.

Automate : notation des états et des transitions Etat initial, état final, transition (étiquetée avec α) entre états

1.1 Question de cours

Rappelez la définition d'une machine à états finis (FSM) de Mealy vue en cours et dites quand considère-t-on qu'elle est a) déterministe, b) partiellement spécifiée, c) Fortement connectée, d) minimale.

2 Modélisation avec les automates

Dans un premier temps, on construira les modèles (automates à états) pour les exercices, puis on dérivera les analyseurs pour chacun des exercices. Il vous est recommandé donc préparer soigneusement tous vos automates sur papier avant de programmer.

2.1 Exercice 1

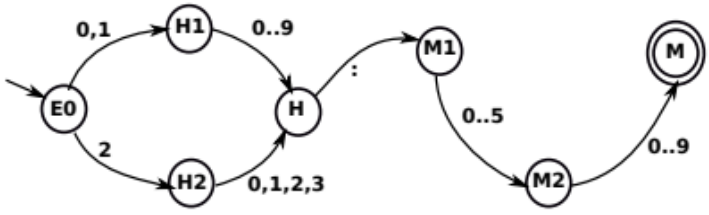


Figure 1: Automate

2.1.1 Question 1

Déroulez l'automate de la figure Fig.1 et trouvez/donnez le comportement qu'il modélise.

2.1.2 Question 2

En précisant les principaux constituants de l'automate $< S, I, \delta, S_0, S_f >$, décrivez l'automate par une représentation tabulaire.

2.1.3 Question 3

Ecrivez un pseudo-algorithme qui donne le comportement de l'automate, pour une suite de caractères en entrée. Pour vous guider, voici un algorithme (incomplet!) :

Algorithm 1 Implantation du comportement d'automate

```
Etatc ← E0 {partons de l'état initial}
{On a une suite de caractères en entrée}
{On prendra carCourant comme le caractère suivant dans la suite}
while Etatc ≠ Ef ∧ Etatc ≠ EtatNonDef do
   lire carCourant {on parcourt la suite de caractère en entrée}
   afficher Etatc, carCourant
   Etatc ← δ(Etatc, carCourant)
end while
if Etatc = Ef then
   Bien terminé : la suite de caractères en entrée est reconnue
else {On s'est arreté sur un état non défini}
   Echec : la suite de caractères n'est pas reconnue
end if
```

Figure 2: pseudo-algorithme de l'Automate

2.1.4 Question 4

Proposez une bonne façon de traiter les cas d'erreur, par exemple dans un état courant, que faire si on lit un caractère inattendu ?

2.1.5 Question 5

Dérivez à partir de la modélisation sous forme d'automate et du pseudo-algorithme du comportement, un programme (en Java) qui donne la fonctionnalité modélisée

dans la Question 3.

Nous vous conseillons de construire au moins : une classe *Automate*, une classe *Etat*, et une classe $MonApplication$. La classe *Automate* doit contenir le 5-uplet qui déinit un automate.

Indications.

- Pour implanter les automates en Java et notamment les transitions entre états, nous vous conseillons d'éviter les structures de données rigides ou inefficaces (par exemple, il faut concevoir la table de transitions autrement).
- Remarquez que pour tout état de l'automate (sauf peut-être pour certains finaux), on a des transitions sortantes (allant vers d'autres états). Si on décrit toutes les transitions sortantes de chaque état, (et si on le fait pour tous les états) alors on a l'ensemble des transitions de l'automate.

  Exemple: en se référant à la Figure 1, de l'état $E0$, on a trois transitions sortantes avec les actions $O, 1, 2$. Il s'agit de $(E0, 0, H1), (E0, 1, H1), (E0, 2, H2)$. Etant dans $E0$ l'action 0 amène à l'état $H1$, 2 amène à l'etat $H2$, etc. On va donc définir dans la classe Etat,les transitions sortantes (d'un état courant), avec une $HashMap < key, value >$. Ceci facilite la manipulation de δ, l'écriture et la généralisation de votre application.

3 Méthodes de génération de tests à partir de machines à états finis

3.1 Exercice 1

Soit la spécification FSM $\mathcal{S}$ de la Figure 3.

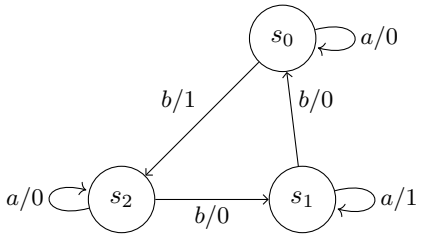


Figure 3: spécification FSM $\mathcal{S}$

1. Donner la suite de test TS ainsi que la séquence de sortie selon la méthode Tour de Transition pour le FSM $\mathcal{S}$ ci-dessus

Input (TS)	.....
Output	.....

2. Dériver une DS de longueur jusqu'à 2 pour $\mathcal{S}$

State	S_0	S_1	S_2	S_0	S_1	S_2	S_0	S_1	S_2
Input (TS)	a	a	a	b	b	b			
Output									

Attention: "a" comme entrée à chaque état bouclera sur l'état, la séquence de "$a.a.$" ne peut pas être une DS, la sortie sera $0.0\ldots$ ou $1.1\ldots$

3. Appliquer le tour de transition dérivé à l'implémentation I de la figure 1 et commenter
4. Appliquer la DS dérivée à l'implémentation I de la figure 4 et commenter

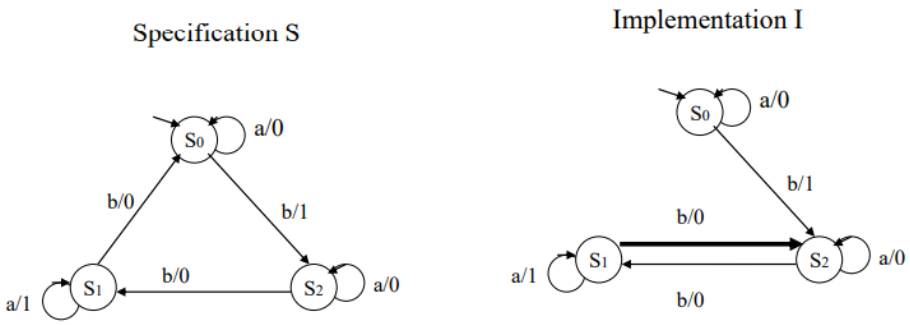


Figure 4: Implementation I

3.2 Exercice 2

Soit la spécification $\mathcal{S}$ de la Figure 5

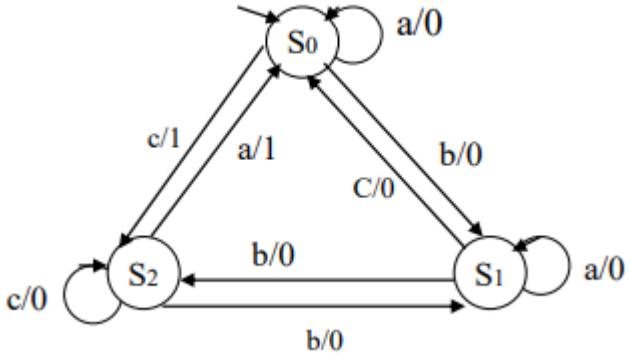


Figure 5: Specification $\mathcal{S}$

1. Dériver un tour de transition pour $\mathcal{S}$
2. Dériver une séquence UIO pour $\mathcal{S}$

3.3 Exercice 3

Considérez le FSM ci-dessous de la Figure 6.

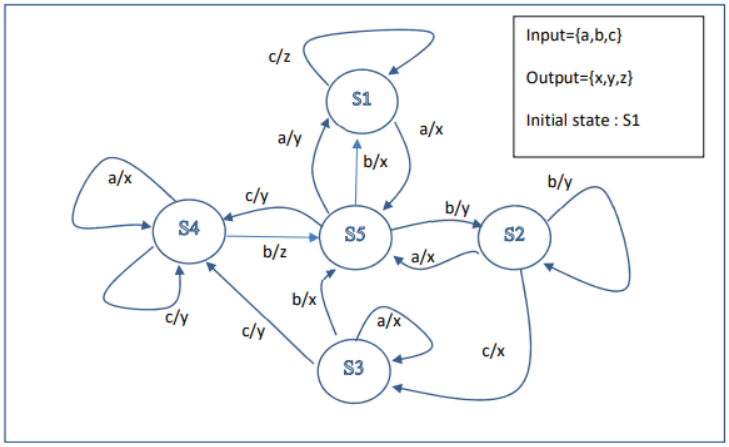


Figure 6: Specification $\mathcal{S}$

1. Trouvez l'ensemble W et la séquence $(P)UIO$ pour tous les états.
2. À partir de la question précédente et de la stratégie de votre choix, déterminez un TS pour la transition $(S2, b/y, S2)$.

3.4 Exercice 4

Considérez le FSM illustré de la Figure 7

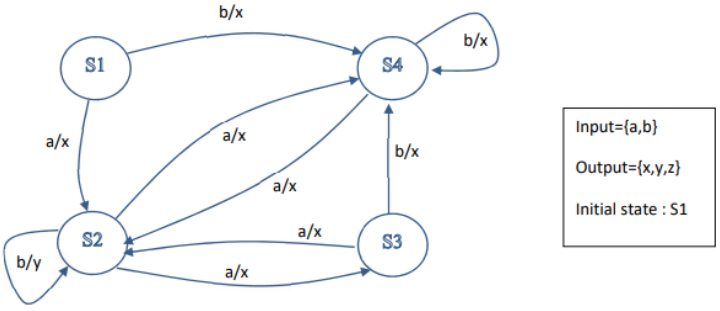


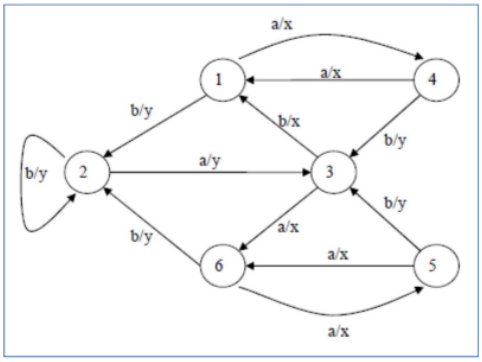
Figure 7: Specification $\mathcal{S}$

1. Déterminez une DS (s'il n'existe pas, utilisez une approche W) pour cet FSM
2. En utilisant la stratégie utilisée ci-dessus, écrivez un TS pour tester la transition $(S2, b/y, S2)$

3. Après l'exécution d'un tel TS, une implémentation I répond : $NULL, y, y, x, x$. Quel serait le verdict du test ?

3.5 Exercice 5

Considérez le FSM illustré de la Figure 8



Input={a,b}

Output={x,y,z}

Initial state : 1

Figure 8: Specification $\mathcal{S}$

1. Déterminer, s'il existe, la DS, l'ensemble W et les $(P)UIO$s.
2. À partir de la question précédente, écrivez 2 séquences de test différentes pour vérifier en particulier la transition $(4, a/x, 1)$. Vous expliquerez brièvement vos choix.
3. En utilisant une de ces séquences, prévoir 2 séquences de sorties, 1 menant à une faute.
4. En utilisant une de ces séquences de test, pourrait-on bloquer l'architecture de test utilisée ? Justifier.

4 Quelques ressources utiles

- API Java *http://sourceforge.net/projects/javafsm/*
- Wiki Automata *https* : $//en.wikipedia.org/wiki/Automata-based_programming$
- Brics Automata *http://www.brics.dk/automaton/*
- Fichier text Memo-Java: des petits rappels

Good Luck!