UNIVERSITÉ DE NANTES

M3301-2: Méthodologie de la production de logiciels
Modélisation et construction des logiciels (C. Attiogbé)
Travaux dirigés/pratiques - 2021/2022

IUT NANTES Université de Nantes

encadrés par Christian Attiogbé

Cahier d'exercices - Des modèles aux programmes

Tous les exos doivent être faits (finissez chez vous ceux qui ne sont pas traités en TD/TP)

Nombre de séances : **7 * 1h20 (sem 46-49)** Compte-rendu demandé : ...selon consignes...

Positionnement du TD dans le déroulement du module

Contenu du cours :
- Cycles de développement des logiciels
- outils, modèles et méthodes de développement
- ► Construction de programmes à partir des modèles à états
- Méthodes de construction formelle : Méthode B

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 transitions 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 : $\langle S, A, \delta, S_0, S_f \rangle$
- $S = \{S_0, \cdots, S_f, \cdots\}$: un ensemble fini d'états
- A : un alphabet d'actions ou d'étiquettes
- δ : une relation de transition définie sur $(S \times A)$ avec $\delta : S \times A \leftrightarrow S$
- S_0 (élément de S) : un état initial
- S_f (éléments de S) : un ou des états finaux

Automate déterministe. Lorsque la relation de transition est une **fonction** (ie. $S \times A \rightarrow S$ au lieu d'une relation de $S \times A \leftrightarrow S$), 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 `alpha`) entre états

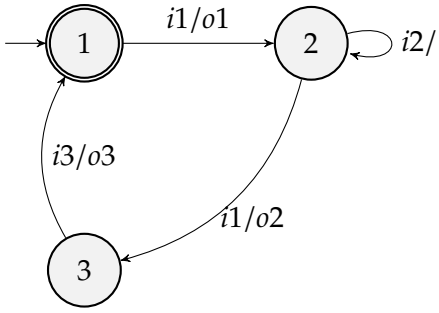
Terminologie/Vocabulaire

- Diagramme Etat/Transition (*State/Transition Diagram*)
- Automate (*Automaton, Automata*)
- Machine à états finis (*Finite State Machine* – utilisée notamment par les américains)
- Système de transitions (étiquetées) : *Labelled Transition System*
- **Alphabet d'actions** : ensembles des actions effectuées par un système
- **Espace d'états** : ensemble des états (possibles) d'un système *state space*
- **Relation de transition, Fonction de transition**.

2 Machine à états de Mealy : définition

Ce sont des automates dont les transitions peuvent comporter des informations d'**entrées/sorties** ; on peut lire des entrées (i_i) et produire des sorties (o_j) ou effectuer des actions.
Un automate de Mealy est défini par un 6-uplet $\langle S, A = In \cup Out, \delta, \delta_o, S_0, S_f \rangle$

- Ensemble d'états : S
- Ensemble d'entrées : In (**événements d'entrées**, conditions)
- Ensemble de sorties : Out (**actions de sortie**, traitements)
- Un état initial : S_0
- Deux relations :
  - transition $\delta : S \times In \to S$
  - sortie $\delta_o : S \times In \to Out$
- Etat final S_f



Machines de Mealy : notation.

- Notation des transitions : $Ss \ \ —i/o \to \ St$
- si l'entrée i est reçue alors que le système est dans l'état Ss,
  la sortie o est produite et le nouvel état du système est St
- i est aussi appelé le déclencheur (*trigger*).

Les transitions entre les états sont étendues par la notation suivante : **evt [garde] / actions***.
evt est l'entrée reçue/lue par le système modélisé.
[garde] : garde est une condition sur l'état du système après l'événement evt.
action* est une suite de 0 ou plusieurs actions.

Caractéristiques d'une machine de Mealy.

- un état dénote l'état complet du système ; une **transition est atomique** (ne peut être décomposée).
- le système est dans **un seul état à la fois**
- **non hiérarchique** ; tout le système est à plat (pas de sous-états).

Généralisation des machines à états. Il y a plusieurs modèles et familles de modèles à états. Plusieurs langages ou formalismes de modélisation utilisent ces modèles à états.

- **System Design Language (SDL)** (normalisé par la *International Telecommunication Union* (ITU), 1988, très utilisé en Télécoms et ailleurs)
- **Statecharts** de David Harel (hiérarchiques) 1987, utilisé par exemple dans UML, papyrus, ...
- **Réseaux de Petri** de Carl Adam Petri (1962, 1969)
- **Algèbres de processus**, telles que *Calculus of Communicating Systems* (CCS, 1980) de R. Milner, *Communicating Sequential Processes* (CSP, 1978) de T. Hoare. C'est la famille la plus expressive des modèles à états.

3 Modélisation avec les automates : par ex. compilateurs de programmes

Dans un premier temps, on construira les modèles (automates à états) pour les exercices successifs, puis on dérivera les analyseurs (en Java) pour chacun des exercices. Vous devez donc préparer soigneusement tous vos automates sur papier avant de programmer.

Exercice 1 : HHMM

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

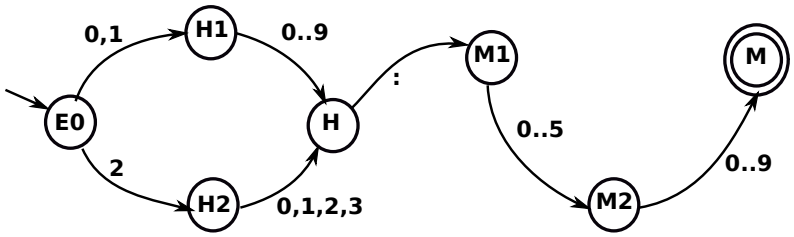


FIGURE 1 – Un automate accepteur

2. En précisant les principaux constituants de l'automate ($\langle S, A, \delta, S_0, S_f \rangle$), décrivez l'automate sous forme d'une table (la fonction de transition). Pour vous guider on donne le début de la table :

```
VARIABLES
    S = ...
    A = ...
    delta =  /* table ----> */
    S_0 = ...
    S_f = ...
```

$S \backslash A$	0	1	2	$\cdots$
$E0$	$H1$	$H1$	$H2$	$\cdots$
$H1$	H	H	H	$\cdots$
$H2$	H	H	H	$\cdots$
H	?	?	?	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

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, à parfaire en TP) :

Algorithm 1 Implantation du comportement d'automate

$Etatc \leftarrow E_0$ {partons de l'état initial}
{On a une suite de caractères en entrée}
{On prendra carCourant comme le caractère suivant dans la suite}
while $Etatc \neq E_f \wedge Etatc \neq EtatNonDef$ **do**
  lire carCourant {on parcourt la suite de caractère en entrée}
  afficher Etatc, carCourant
  $Etatc \leftarrow \delta(Etatc, carCourant)$
end while
if $Etatc = E_f$ **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

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 ?
5. Modifiez l'automate en un automate de Mealy où l'action de chaque transition consiste à afficher le caractère lu.

Exercice 2 : smileys

1. Ecrivez un automate pour reconnaître les *smileys* suivants et uniquement ceux là.
   :-) :-(;-) ;-(:=) :=(
2. Décrivez l'automate (la fonction de transition) sous forme d'une table.
3. Ecrivez un pseudo-algorithme qui implante le comportement de l'automate.
   Quel(s) constat(s) faites-vous à ce stade ?

Automates déterministes analyseurs d'expressions

Exercice 3 : jjmmaaaa

Dans le formulaire d'un logiciel, les utilisateurs doivent saisir à un endroit une date sous la forme jj/mm/aaaa. Pour simplifier ici, on ne traîte pas les années bissextiles (on les prend de 0001 à 9999).

A un autre endroit dans le formulaire, les utilisateurs doivent saisir une heure sous la forme hh :mm :ss

1. Ecrivez un automate qui reconnaît une expression de date bien formée (jj/mm/aaaa).
2. Ecrivez un automate qui reconnaît une expression d'heure bien formée (hh :mm :ss).
3. Pensez à la phase d'analyse du logiciel qui traite le formulaire ; que peut-on réutiliser par rapport à la reconnaissance des expressions ?

Exercice 4 : adresses électroniques

Une adresse électronique normalisée est sous la forme `Prenom.Nom@nomDeDomaine`, ou `chaineQuelconque@nomDeDomaine`. Les caractères alphanumériques et quelques caractères spéciaux sont autorisés. Par exemple `paul-annie.martin44@univ-nantes.fr` est une adresse correcte.

1. Ecrivez un automate qui reconnaît une expression d'adresse mail bien formée.
2. Modifiez l'automate pour reconnaître une liste d'adresses électroniques séparées par des ";"

Exercice 5 : polynômes

Construisez un automate qui reconnaît l'expression d'un polynôme sous la forme : $aX^{**}2 \oplus bX \oplus c$
a, b et c sont des entiers (entre 1 et 25). Les espaces en nombre quelconque sont tolérés.
X est la seule variable autorisée.
$\oplus$ est un des deux opérateurs - ou +
Par exemple voici des polynômes correctement écrits : `8X**2 - 15X + 3` ou bien `4X**2 + 3X - 9`
alors que `42X**2 + 7X` ou bien `4X*2 -3X + 9` sont syntaxiquement incorrects.

Exercice de TP : dérivation des programmes analyseurs, en Java

En vous appuyant sur le pseudo algorithme étudié, et des indications ci-après (et www du module)

— Dérivez le programme Java correspondant à la reconnaissance d'une expression.

— Modifiez simplement l'automate en entrée du programme et testez avec le même programme, pour tous les exercice.

4 Implantation en Java

Dérivez à partir de la modélisation sous forme d'automate et du pseudo-algorithme du comportement un programme/module/procédure (en Java) qui donne la fonctionnalité modélisée dans les exercices précédents.
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.

On traitera les chaînes en entrée caractère par caractère. Un mémo Java est à votre dispi sur le www du module.

La figure 2 donne l'architecture globale de ce qu'on vous demande de faire.

FIGURE 2 – Architecture simplifiée de l'application à écrire

Aussi vous adopterez une démarche combinant une *approche descendante* (allant de l'application vers les classes dont elle a besoin) et une *approche ascendante* (allant des classes et méthodes élémentaires vers leur utilisation dans l'application).

Aide à la conception. 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).

Remarquer 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. Par exemple, de l'état `E0` de la figure 1, on a trois transitions sortantes avec les actions `0, 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 grandement la manipulation de *delta*, l'écriture et la généralisation de votre application.

Aide à la mise au point. Utiliser une méthode java pour créer/charger votre automate. Cette méthode est appelée à partir de votre `main`. Une fois que vous aurez mise en œuvre votre première version de l'application, vous allez structurer le `main` à l'aide d'un simple menu alphanumérique (qui sera progressif) dans votre TP. Vous proposerez à l'utilisateur de choisir un analyseur avant de lui demander la chaîne à analyser ; en fonction de son choix, vous chargerez un de vos automates (chacun créé par une méthode dédiée), puis vous analyserez la chaine donnée (lue en entrée).

```
--------------- Menu de mon TP -------------------------
  1. Smiley (pour reconnaitre un des smileys)
  2. HH:MM  (pour reconnaitre une heure bien formée)
  3. JJ/MM/AAAA
  4. ...
  9. Arrêt
  Votre choix (1-9) ?
  Je vous demanderez ensuite la chaîne à analyser, Merci
  ------------------------------------------------------
```

Généralisation - réutilisation

Dans une application logicielle, on voudrait (ré)utiliser au choix, différents modules de reconnaissance d'expressions. Comment concevoir l'application pour que le module qui reconnaît une expression soit remplacé par un autre module prédéfini qui fait la même chose ?

```
...
public class monApplication
{
  ... // compléter les ...
  ... monAnalyseur = new ... // construire un automate
  ...                    // paramétrer l'automate
  public static void main(String[] argv) {
      ...
      monAnalyseur.reconnaitre(...) // comment faire simplement ici ??
      ...
  }
}
```

1. Ecrivez un programme/module/procédure qui a pour paramètre un automate qui encode une fonction d'analyse (de reconnaissance) d'une expression).
2. Proposez une solution pour mettre en œuvre facilement l'utilisation de différentes versions d'analyseur sans modifier l'application telle que décrite au ci-dessus (utiliser par ex. les interfaces Java).

Exo : Modélisation et programmation du comportement d'un serveur FTP

Le protocole *FTP (File Transfer Protocol)* est un protocole de transfert de fichiers entre machines situées sur un réseau. Une machine (dite **client ftp**) va déposer/charger des données situées sur une autre machine (dite **serveur ftp**).

On veut analyser le comportement d'un **client ftp** et d'un **serveur ftp**. Ces comportements correspondent à des programmes appelés ici processus.
Un processus serveur ftp attend des demandes de connexions (`open`) de processus clients, puis interagit avec eux pour rendre des services.
Un processus client, avant de charger/déposer des données, demande une connexion à un serveur ftp. Lorsque la connexion est effectuée, le serveur demande un mot de passe au client et vérifie la validité du mot de passe ; en cas de succès, le client est autorisé à effectuer des actions (décrites ci-après). En cas d'échec le client n'est pas autorisé à effectuer des actions, il peut redonner le mot de passe si le serveur le réclame.

Les actions effectuées par un processus client sont `ls, get, put, close` ; ce sont des actions attendues par le serveur après la connexion et la validation du mot de passe du client ;

— `ls` : lorsque le serveur reçoit cette demande, il renvoie une donnée au client, le client affiche cette donnée puis peut effectuer une autre action ;
— `get` : lorsque le serveur reçoit cette demande, il renvoie une donnée au client, le client affiche cette donnée puis peut effectuer une autre action ;
— `put` : lorsque le serveur reçoit cette demande, il renvoie un acquittement au client ; le client peut en fonction de l'acquittement recommencer la même action `put` ou effectuer une autre action ;
— `close` : lorsque le serveur reçoit cette demande, il ferme la connexion en cours avec le client ; le client ne peut alors plus interagir avec le serveur. Le serveur lui, se met en attente d'une autre connexion.

En faisant abstraction du traitement des fichiers (on envoie de façon abstraite, une donnée),

1. modélisez avec un automate de Mealy le comportement d'un serveur ftp ;

2. dérivez le programme qui implante le serveur.

Exo : Construction d'un analyseur pour un DSL

Un *Domain Specific Language (DSL)* est un langage adhoc (souvent très réduit) conçu pour une ou des applications spécifiques. Un analyseur adhoc est aussi construit à cet effet.

Dans cet exercice, on vous demande d'écrire l'analyseur (syntaxique) d'un DSL dédié à la commande d'une maison intelligente où il y a plusieurs équipements pilotés par ordinateurs. On envoie donc des commandes par un logiciel, aux différents équipements.

Parmi les commandes nous avons en priorité la commande Marche/Arrêt (MA) qui active/désactive un équipement. Lorsqu'un équipement est activé, il peut recevoir des ordres jusqu'à ce qu'il soit désactivé. Pour commencer, imaginons un seul robot aspirateur qui réagit aux commandes suivantes :

— vlente : change la vitesse actuelle en une vitesse lente (prédéfinie).
— vrapide : change la vitesse actuelle en une vitesse rapide (prédéfinie).
— vnormale : change la vitesse actuelle en une vitesse normale (prédéfinie).
— avance(n) : avance de n unités de déplacement élémentaire.
— quartour : fait un quart de tour.

Par exemple, on peut faire faire un déplacement sous forme d'un carré au robot en donnant l'expression suivante : MA ; lent ; avance(2) ; quartour ; avance(2) ; quartour ; avance(2) ; quartour ; avance(2) ; quartour ; MA

— Modélisez par un automate l'analyser du langage de commande dédié au robot.
— Dérivez le programme qui sera installé sur le robot ; ce programme
— reçoit une expression qui correspond à une suite de commandes,
— analyse l'expression et donne les commandes appropriées au robot
— Simulez le fonctionnement du programme.

Exercices d'entraînement : reconnaissance de termes

1. Proposez un automate déterministe permettant de reconnaître des expressions constituées soit d'une suite de chiffres terminée par une lettre alphabétique, soit d'une suite de lettres alphabétiques terminée par un chiffre ; soit une liste de ces expressions séparées par un ; (mais on n'a pas de ; à la fin). Par exemple
   `632972L  jthsbf8  lsqsbk5 ; 0498588H`
   `4808736G ; sqdsytiq9 ; ozahdlb8  ...`
2. Proposez un automate déterministe permettant de reconnaître une instruction d'affectation avec une expression arithmétique bien parenthésée (appelons là <EXPRAR>), mais sans imbrication de parenthèses, utilisant uniquement les variables x, y et les opérateurs +, - et / .
   L'instruction a la forme : `x := <EXPRAR>`
   Des expressions arithmétiques <EXPRAR> correctes sont par exemple :
   `(x)  (y)  (x + y)  (x - y)  (x + y + x - y / x + y)`
   `(x + y) + (x - y) / (x + y)`
3. Un automate qui reconnaît les nombres qui sont multiples de 3 (Listez-en quelques-uns, observez les (régularite) et décidez).
4. Un automate qui reconnaît les nombres inférieurs ou égal à 138.
5. Un automate qui reconnaît les mots de la forme $a((ab)^*cb^*)^*$ sur l'alphabe $\{a, b\}$.

Références

— Un poly de maths (Info1, Mr Connan, ... les automates) :
http://download.tuxfamily.org/tehessinmath/les%20pdf/Poly_automates_2012.pdf
— Page du cours
http://pagesperso.ls2n.fr/~attiogbe-c/mespages/methodologie-production-applications.html
— API Java : http://sourceforge.net/projects/javafsm/
— https://en.wikipedia.org/wiki/Automata-based_programming
— http://www.brics.dk/automaton/