

# Compilation (HAI705I)

Master Informatique  
Département Informatique  
Faculté des Sciences de Montpellier  
Université de Montpellier



## Examen du 21 mars 2025

Tous les documents de cours sont autorisés. L'examen dure 2h. Le barème est donné à titre indicatif. Le sujet comporte 3 pages et il y a 4 exercices.

### Exercice 1 (5 pts)

Soit la fonction  $f$  sur les entiers naturels définie de la façon suivante :

$$f(n) = \begin{cases} 0, & \text{si } n = 0 \\ f(n-1) + 2, & \text{sinon} \end{cases}$$

1. Que fait la fonction  $f$ ?  
Justifier en déroulant son exécution sur au moins deux exemples.
2. Écrire la fonction  $f$  en PP.
3. Traduire la fonction  $f$  en UPP.
4. Traduire la fonction  $f$  en RTL.
5. Traduire la fonction  $f$  en ERTL.  
A-t-on besoin de registres *callee-save* dans cette traduction ? Justifier pourquoi.

### Exercice 2 (5 pts)

Soit le programme PP suivant :

```
y := y + 1;  
if y = t then  
  x := u + v  
else  
  x := x + 1;  
z := x + v
```

1. Dessiner le graphe de flot de contrôle du programme.  
Réaliser l'analyse de durée de vie des variables sachant qu'à la fin du programme, il n'y a aucune variable vivante.  
Dessiner le graphe d'interférences correspondant.

2. Combien de couleurs faut-il au minimum pour colorier le graphe d'interférences sans devoir « spiller » ? Vous devrez justifier votre réponse en coloriant le graphe avec  $k$  et  $(k - 1)$  couleurs, où  $k$  est le nombre de couleurs minimum pour colorier le graphe sans « spiller ». Ces coloriations devront se faire en utilisant l'algorithme de Chaitin.

### Exercice 3 (6 pts)

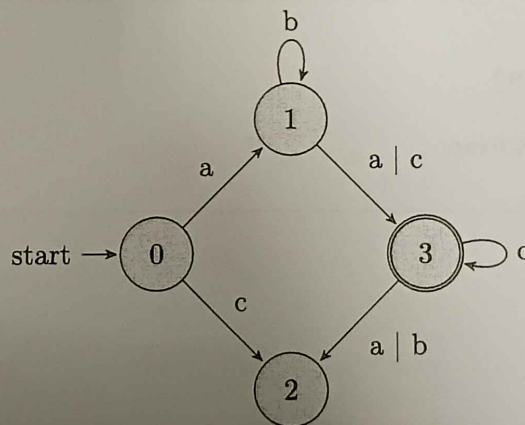
Nous disposons d'une machine virtuelle (VM) à registres, proche de celle du cours mais très simplifiée. L'objectif est de générer un code de cette VM permettant l'interprétation d'automates déterministes, c'est-à-dire la reconnaissance d'un mot par un automate. On suppose que la VM peut gérer des listes LISP, avec des opérations spécialisées :

- (`car R1 R2`) prend la cellule dont l'adresse est dans le registre R1 et charge son champ `car` dans le registre R2 ; (`cdr` a le rôle symétrique pour le champ `cdr`) ;
- l'opération de comparaison (`cmp R1`), utilisée avec un seul opérande, permet d'effectuer des tests de cellules (`consp`, `atom`, `null` en LISP) sur le contenu du registre R1 en positionnant des drapeaux de manière usuelle ;
- (`bconsp #label`) est une instruction de branchement conditionnel qui effectue le branchement si le drapeau préalablement positionné par `cmp` indique qu'il s'agit bien d'une cellule. Avec `batom` et `bnull`, le branchement est conditionné au fait que la valeur testée est un atome ou `nil`.

Les conventions pour le code d'interprétation des automates sont les suivantes. La donnée (mot à reconnaître) est une liste de caractères (symboles), par exemple (`c b a c`), contenue dans le registre R0. À l'issue de l'exécution, la VM s'arrête et R0 contient l'état final atteint lors de l'exécution de l'automate, ou `nil`, suivant que le mot a été reconnu ou pas.

#### Question 1

Soit l'automate ci-dessous, dont l'état initial est 0 et l'ensemble des états finaux est {3} :



1. Indiquer comment traduire les états de l'automate dans le jeu d'instructions de la VM.
2. Écrire le code VM correspondant à l'automate donné ci-dessus, en le commentant.

On suppose que l'on dispose, en LISP, d'un type de données automate, muni de l'interface fonctionnelle suivante :



- `(auto-etat-liste auto)` retourne la liste des états (entiers) de l'automate : pour celui de l'exemple, `(0 1 2 3)` ;
- `(auto-init auto)` retourne l'état (entier) initial de l'automate (0 dans l'exemple) ;
- `(auto-final-p auto etat)` retourne vrai si l'état argument est final (dans l'exemple, vrai pour 3, faux pour les autres) ;
- `(auto-trans-list auto etat)` retourne la liste des transitions issues de l'état argument, sous la forme d'une liste.

### Question 2

Écrire la fonction LISP `auto2vm` qui prend en argument un automate déterministe (au sens de la structure de données précédente) et retourne le code VM correspondant (c'est-à-dire un code similaire à celui que vous avez écrit dans la question précédente pour l'automate donné en exemple).

1. Spécifier le principe de la génération : comment traduire les états, les transitions, les états finaux, l'état initial, etc.
2. Décomposer le problème en définissant des fonctions annexes pour traiter séparément, chaque transition, chaque état, etc.

## Exercice 4 (4 pts)

Les questions qui suivent concernent les éléments du projet compilation : le compilateur, la machine virtuelle, et le chargeur.

### Question 1

Expliquez comment vous implémentez la machine virtuelle (VM) en termes de structures de données. Pour les informations de la VM, pour lesquelles choisissez-vous de les modéliser comme attributs et lesquelles vous placez dans la mémoire de la VM ? Justifiez vos choix.

### Question 2

Expliquez comment s'organise la transformation source à source et la génération de code avec le jeu d'instruction de la VM. Donnez un exemple parlant.

### Question 3

Comment chargez-vous la version compilée du chargeur ? À quel moment le faites-vous ? Comment chargez-vous du code avec la version compilée du chargeur ?