Compilation et interprétation (HMIN104)

Master AIGLE
Département Informatique
Faculté des Sciences de Montpellier

Examen du 15 janvier 2020

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

Exercice 1 (6 pts)

On considère la suite de Syracuse d'un entier $N > 0$ définie comme suit :

$$u_0 = N$$
$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } n \text{ est pair} \\ 3u_n + 1 & \text{si } n \text{ est impair} \end{cases}$$

La suite de nombres ainsi calculée est appelée le vol de N et les nombres de la suite sont appelés les étapes du vol. Le nombre d'étapes avant d'obtenir 1 est appelé la durée du vol et le plus grand nombre obtenu dans la suite est appelé l'altitude maximale du vol.

1. La conjecture de Syracuse dit que quel que soit le nombre de départ dans la suite obtenue avec l'algorithme de Syracuse, on finit toujours par obtenir 1. Autrement dit, la durée d'un vol est toujours finie.
   Quel est la durée et l'altitude maximale du vol de 5 ?
2. Écrire la suite de Syracuse en PP. On supposera que l'on a, à sa disposition, une fonction, appelée *pair*, qui teste la parité d'un entier.
3. Traduire le code obtenu à la question 2 en UPP.
4. Traduire le code obtenu à la question 3 en RTL.
5. Traduire le code obtenu à la question 4 en ERTL.

Exercice 2 (6 pts)

On considère la suite de Syracuse donnée dans l'exercice précédent.

1. Dessiner le graphe de flot de contrôle du corps de la fonction PP de la suite de Syracuse, que vous avez écrite dans l'exercice précédent.
2. Faire une analyse de durée de vie des variables de ce code sachant qu'à la fin de ce code, aucune variable n'est vivante.
3. Dessiner le graphe d'interférences correspondant.
4. Déterminer le nombre minimal de couleurs nécessaire pour colorier le graphe d'interférences obtenu sans « spiller ».

Exercice 3 (12 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 :

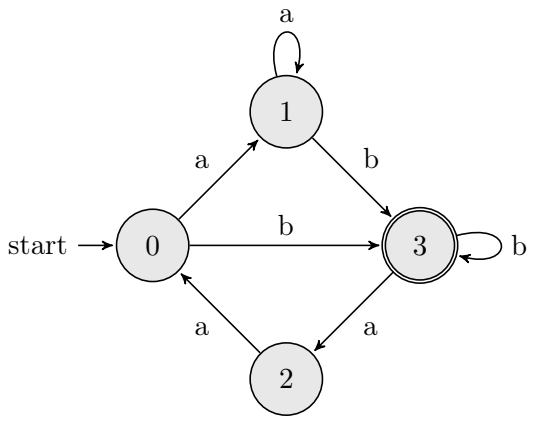
— `(car R1 R2)` prend la cellule dont l'adresse est dans le registre `R1` et charge son champ `car` dans le registre `R2` ; (`cdr` a le rôle symétrique pour le champ `cdr`) ;

— 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 `(a b b a)`, 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'état final est 1 :



1. Commencer par indiquer comment il est possible de traduire les états de l'automate dans la VM.
2. Écrire le code VM correspondant à l'automate donné en exemple ci-dessus, en le commentant.

On suppose que l'on dispose, en LISP, d'un type de données abstrait 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 voisin de celui que vous avez écrit dans la question précédente pour l'automate de la figure).

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.