Compilation et interprétation (HMIN104)

Master AIGLE
Département Informatique
Faculté des Sciences de Montpellier

Examen du 29 mars 2018

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)

Soit la fonction f qui teste la parité d'un entier :

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

1. Écrire la fonction f en PP.
2. Traduire la fonction f en UPP.
3. Traduire la fonction f en RTL.
4. Traduire la fonction f en ERTL.

Exercice 2 (6 pts)

Soit le programme PP suivant :

$x := 1;$
$y := t;$
$y := z;$
$u := v + x$
$u := v + y;$
$x := u + y$

---

1. Dessiner le graphe de flot de contrôle de ce programme.
2. Faire une analyse de durée des variables sachant qu'à la fin du programme, x et y sont vivantes.
3. Dessiner le graphe d'interférences correspondant.
4. Colorier le graphe d'interférences avec 3 couleurs. Doit-on « spiller » ? Mêmes questions avec 2 couleurs.

Exercice 3 - Compilation d'automates vers une VM (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 ;

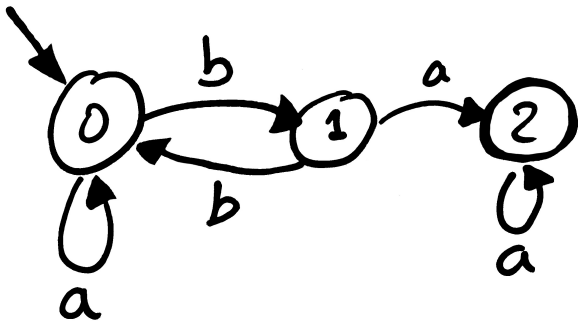
— `(bconsp #label)` est une instruction de branchement conditionnel qui effectue le branchement si le drapeau préalablement positionné par `cmp` indique qu'il s'agit bien d'une cellule ; avec `batom` et `bnull`, le branchement est conditionné au fait que la valeur testée est un atome ou `nil`.

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 b a)`, contenue dans le registre `R0`. A 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 déterministe suivant, dont l'état initial est 0 et l'état final est 2 :

1. Commencez 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 la figure, (0 1 2)
- `(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 2, 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.

* * * * * * * * * * * * *