Mines Informatique MP 2001 — Corrigé

Ce corrigé est proposé par Olivier Bertrand (ENS Lyon) ; il a été relu par Xavier Goaoc (ENS Cachan) et Codrin Nichitiu (Maître de conférence).

L'épreuve se compose de trois problèmes indépendants.

Dans le premier problème, consacré aux automates finis, on écrit un algorithme permettant d'émonder un automate, c'est-à-dire de déterminer ses états inutiles. On utilise dans ce but l'algorithme de Roy-Warshall qui construit la matrice d'accessibilité d'un automate.

Dans le deuxième problème, consacré à l'algorithmique, on étudie différents algorithmes essayant de minimiser le nombre de multiplications dans le calcul de x^n.

Dans le troisième problème, consacré à la logique propositionnelle, on construit différents circuits logiques aboutissant à la conception d'un *comparateur 8-bits*.

Indications

Partie I

I.2 Penser aux types `array` et `boolean`.

I.5 Parcourir entièrement le tableau `G`.

I.9 Supposer $C_{k+1}[i,j]$ vrai et $C_k[i,j]$ faux...

I.11 Modifier le tableau C_k grâce à la propriété de la question I.9.

I.13 Utiliser la question I.11 pour k de 0 à $N_E - 1$, ou de 1 à N_E.

I.16 Utiliser la matrice d'accessibilité.

I.17 Ne pas oublier le temps pris par l'algorithme de Roy-Warshall.

Partie II

II.3.a Faire une récurrence forte sur n.

II.4.a Séparer les cas n pair et n impair.

II.4.f Trouver une formule de récurrence pour π à partir des deux récurrences précédentes, et remarquer que les fonctions `cd` et `pi` calculent les mêmes valeurs.

Partie III

III.1 Penser à $\vee$ et $\wedge$.

III.2 Utiliser le circuit de la question III.1.

III.3 Faire un tableau de Karnaugh pour chacune des sorties.

III.4 Comparer deux nombres se fait de la façon suivante : on compare leurs premiers bits, s'ils diffèrent on sait quel est le plus grand des deux nombres. Sinon on recommence sur leurs deuxièmes bits, et ainsi de suite.

I. Problème sur les automates finis

I.1 Dans ce corrigé nous vous proposons systématiquement une version Caml et une version Pascal de chaque programme. Le jour du concours, vous devez bien sûr choisir un seul langage (et vous y tenir).

I.2 L'énoncé propose une structure de vecteurs emboîtés pour décrire le graphe orienté et étiqueté d'un automate. Une telle structure s'implémente facilement sous forme de tableau, que ce soit en Pascal ou en Caml. Dans ces deux langages, on utilise une structure de tableau de N_E tableaux de N_L tableaux de N_E booléens.

Version Caml

```
val G : bool array array array
```

Version Pascal

```
var G : array of array of array of boolean
```

> Il ne serait pas judicieux d'utiliser des tableaux d'entiers avec 1 pour *vrai* et 0 pour *faux*. En effet, ceci supprimerait quelques vérifications de types du compilateur : un tel tableau pourrait contenir d'autres valeurs que 0 et 1, ce qui n'a pas de sens. De plus, cela conduirait à un code plus lourd.

I.3 Dans les deux langages, on commence par initialiser un tableau à trois dimensions où toutes les cases possèdent la valeur *faux*. Il suffit alors d'instancier à *vrai* chacune des cases $G[i, j, k]$ telles que (e_i, e_k) soit dans E et étiquetée par l_j.

Version Caml

```
let g = init_vect 5
  (fun _ -> init_vect 2 (fun _ -> make_vect 5 false));;
g.(0).(0).(1) <- true;
g.(1).(1).(2) <- true;
g.(2).(0).(0) <- true;
g.(2).(1).(4) <- true;
g.(2).(0).(3) <- true;
g.(4).(0).(3) <- true;;
```

> `init_vect` a pour type `int -> (int -> 'a) -> 'a vect`.
> `init_vect n f` renvoie le vecteur (de longueur `n`)
> `[|f 0 ; f 1 ; f 2 ; ... ; f (n-1)|]`

Version Pascal

```
const ne = 5;
const nl = 2;

var g = array[1..ne, 1..nl, 1..ne] of boolean;

begin
   for i := 1 to ne do
      for j := 1 to nl do
         for k = 1 to ne do
            g[i, j, k] := false;
   g[1, 1, 2] := true;
   g[2, 2, 3] := true;
   g[3, 1, 1] := true;
   g[3, 2, 5] := true;
   g[3, 1, 4] := true;
   g[5, 1, 4] := true;
end;
```

> Il ne faut pas oublier qu'en Caml la première case d'un tableau porte le numéro 0. Croire que ce numéro est 1 (confusion fréquente) conduit à des erreurs qui sont difficiles à déceler, donc à corriger.

I.4 La matrice d'accessibilité C d'un automate peut être implémentée, en Caml comme en Pascal, par un tableau de N_E tableaux de N_E booléens. $C[i, j]$ a la valeur *vrai* si, et seulement si, il existe un calcul d'origine e_i et d'extrémité e_j.

I.5 L'idée est de construire une matrice carrée C de taille N_E où tous les éléments ont la valeur *faux*, et d'examiner ensuite chaque couple $(i, j) \in [\![1 ; N_E]\!]^2$ en fixant $C[i, j]$ à *vrai* si

$$i = j \quad \text{ou} \quad (\exists k \in [\![1 ; N_L]\!] \quad G[i, k, j] = vrai)$$

Version Caml

```
let init g =
 let ne = vect_length g in
 let nl = vect_length g.(0) in
 let result =
   init_vect ne (fun _ ->
     init_vect ne (fun _ -> false)) in
 for i=0 to ne-1 do
   for j=0 to ne-1 do
     if i=j
     then result.(i).(j) <- true
     else
       for k=0 to nl-1 do
         if g.(i).(k).(j) then result.(i).(j) <- true
       done
   done
 done;
 result;;
```