Examen du cours «Analyse d'algorithmes» (2ème session)

10 mars 2006

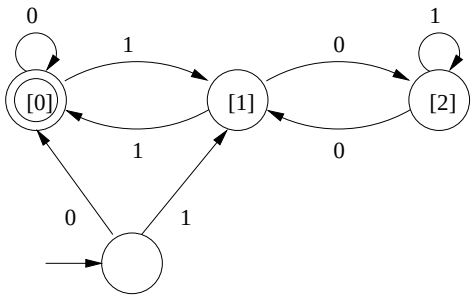
Rappel : seule une feuille A4 de notes manuscrites est autorisée pendant l'examen. Tout autre document est interdit. Le barême donné est indicatif. Soyez précis et concis. On rappelle que le soin apporté aux justifications demandées est déterminant. Veillez à ne pas confondre votre copie et vos feuilles de brouillon, merci !

1 Automates finis

Exercice 1 (3 points)
Pour chaque langage ci-dessous, donner l'automate fini **déterministe et minimal** qui le décide :

1. Le langage des mots sur l'alphabet $\{a, b\}$ qui contiennent au moins un a
2. Le langage des mots sur l'alphabet $\{a, b\}$ qui contiennent un nombre impair de a
3. Le langage des mots sur l'alphabet $\{a, b\}$ dont le deuxième caractère à partir de la fin est un a
4. Le langage des mots sur l'alphabet $\{0, 1\}$ qui représente l'ensemble des entiers pairs

Exercice 2 (4 points)
L'automate fini déterministe minimal et complet ci-dessous reconnaît l'ensemble des entiers, encodés en binaire, qui sont des multiples de 3.



Donner l'automate fini **déterministe, minimal et complet** qui reconnaît l'ensemble des entiers, encodés en binaire, qui sont à la fois des multiples de 3 et pairs. Vous expliciterez la méthode adoptée et justifierez votre résultat.

2 Preuve d'algorithmes

Nous considérons le while-program suivant qui calcule le nombre d'occurrences d'un élément `e` dans une liste `l`.

```
n := 0; l' := l;
while (l' != NULL) do
  if (elm(l') = e) then
    n := n+1
  endif;
  l' := next(l')
endwhile
```

Le TAD (Type Abstrait de Données) `list` fournit les opérations suivantes :

```
NULL : list
elm : list -> element
next : list -> list
```

où `NULL` désigne la liste vide, `elm(l)` renvoie l'élément stocké dans la cellule en tête de la liste `l`, et `next(l)` renvoie la liste `l` privée de sa première cellule. Nous disposons de plus de la fonction `nbocc : element * list -> int` qui indique le nombre d'occurrences d'un élément dans une liste.

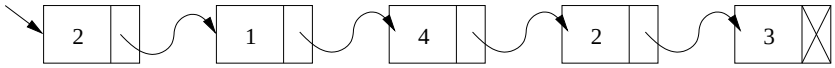
Exercice 3 (7 points)

Notre but est de prouver que ce while-program est **correct**, c'est à dire qu'il satisfait les spécifications :

- Pré-cond : `vrai`
- Post-cond : `n=nbocc(e,l)`

Répondre aux questions suivantes :

1. On considère l'exemple de la liste suivante :



   Compléter le tableau ci-dessous où chaque ligne correspond à une itération de la boucle `while`.

`n`	`nbocc(2,l')`	`nbocc(2,l)`
0	2	2
⋮	⋮	⋮

2. Donner un invariant pour la boucle `while` qui permet de prouver la correction. Vous expliquerez votre choix.
3. Calculer la preuve de ce while-program et donner les obligations de preuve que vous générez.
4. Indiquer, en le justifiant, si vos obligations de preuve sont satisfaites. Conclure sur la correction du while-program.

Exercice 4 (6 points)

Nous nous intéressons maintenant à la **terminaison** de ce while-program.

1. Quelle méthode allez-vous utiliser ? Que faut-il prouver lors de l'application de celle-ci ?
2. Proposer une mesure en justifiant votre choix. Vous pouvez, si cela vous semble nécessaire, ajouter une opération au TAD `list`
3. Donner les obligations de preuve générées par la preuve de terminaison (on ne demande pas le calcul de la preuve). Vous indiquerez la validité de chacune d'elles

Règles de la logique de Hoare

Règle de conséquence :

$$\frac{P' \Rightarrow P, \{P\}\, S\, \{Q\}\ ,\ Q \Rightarrow Q'}{\{P'\}\, S\, \{Q'\}}$$

Règle d'affectation :

$$\frac{}{\{P[e/x]\}\, x{:=}e\, \{P\}}$$

Règle de séquence :

$$\frac{\{P\}\, S_1\, \{Q\}\ ,\ \{Q\}\, S_2\, \{R\}}{\{P\}\, S_1;S_2\, \{R\}}$$

Règle du «`si`» :

$$\frac{\{P \wedge B\}\, S_1\, \{R\}\ ,\ \{P \wedge \neg B\}\, S_2\, \{R\}}{\{P\}\, \texttt{si}\ B\ \texttt{alors}\ S_1\ \texttt{sinon}\ S_2\ \texttt{finsi}\, \{R\}}$$

Règle du «`tantque`» :

$$\frac{\{P \wedge B\}\, S\, \{P\}}{\{P\}\, \texttt{tantque}\ (\texttt{B})\ \texttt{faire}\ \texttt{S}\ \texttt{fintantque}\, \{P \wedge \neg B\}}$$