Calculabilité
Partiel mai 2015 (corrigé succinct)

Documents de cours et TD autorisés – Durée : 3h.
Le barème est donné à titre indicatif (1 pt ≈ 9 min).
Les différents exercices sont indépendants et peuvent être traités dans n'importe quel ordre.

Exercice 1 : While (4 pts)

On rajoute à WHILE une opération pour effectuer directement l'addition de deux nombres en WHILE, c'est-à-dire qu'on autorise les expressions de la forme `E + F`.

Soit `sum` le programme suivant :

```
read L ;
   RES:=nil;
    while L do
    {
       RES:=(hd L) + RES;
       L:=tl L
    }
write RES
```

1/ Donner la sémantique de cette nouvelle opération, c'est-à-dire expliciter $\mathcal{E}[\![\mathtt{E}+\mathtt{F}]\!]\sigma$.
Attention, les expressions ne représentent pas forcément des entiers.
2/ Évaluer $[\![\mathtt{sum}]\!]([\underline{3},\underline{2},\underline{6}])$
3/ Montrer que $[\![\mathtt{sum}]\!]([\underline{k_1},\ldots,\underline{k_n}]) = \underline{k_1+\ldots+k_n}$.
4/ Donner $\langle \mathtt{sum} \rangle_{WG}$

Corrigé

1. L'idée de base, c'est d'avoir $\mathcal{E}[\![\mathtt{E}+\mathtt{F}]\!]\sigma = \mathcal{E}[\![\mathtt{E}]\!]\sigma + \mathcal{E}[\![\mathtt{F}]\!]\sigma$. Le + à gauche de l'égalité est le nouveau symbole ajouté dans WHILE, celui de droite est l'addition des entiers. Le soucis est que l'addition est définie sur les nombres et pas sur les arbres binaires. Il faut donc s'assurer que $\mathcal{E}[\![\mathtt{E}]\!]\sigma$ est bien (la représentation d') un nombre. On obtient donc

$$\begin{cases} \mathcal{E}[\![\mathtt{E}+\mathtt{F}]\!]\sigma = \underline{m+n} & \text{si } \mathcal{E}[\![\mathtt{E}]\!]\sigma = \underline{m} \text{ et } \mathcal{E}[\![\mathtt{F}]\!]\sigma = \underline{n} \\ \mathcal{E}[\![\mathtt{E}+\mathtt{F}]\!]\sigma = \mathtt{nil} & \text{sinon} \end{cases}$$

   Le choix de `nil` comme valeur "par défaut" dans le cas ou l'un des arbres n'est pas un nombre est arbitraire et n'importe quelle autre valeur fait l'affaire.
2. Avec la sémantique de +, on n'a pas besoin de détailler la représentation des entiers.
3. Comme pour la preuve au premier partiel, on fait d'abord une preuve par récurrence sur la boucle avec une valeur initiale de `RES` qui intervient. Ici, la récurrence est faite sur la longueur de la liste. L'hypothèse de récurrence est donc que si on évalue la boucle dans une certaine mémoire σ où `L` vaut $[\underline{k_1},\ldots,\underline{k_n}]$, on se retrouve dans une mémoire σ' où `RES` vaut $\sigma(\mathtt{RES}) + \underline{k_1+\ldots+k_n}$.
4. Comme toujours, la valeur précise du `goto` de début de boucle ne peut être connue que quand toute la transformation de la boucle est faite. En suivant les règles, on obtient

```
read L;
  1 : RES := nil;
  2 : if L then goto 3 else goto 6;
  3 : RES := (hd L) + RES;
  4 : L := tl L;
  5 : if L then goto 2 else goto 2;
write RES;
```


Correction

Exercice 2 : Théorème de Rice (3 pts)

1/ Rappeler l'énoncé du théorème de Rice et la définition d'ensemble extensionnel.
2/ Donner un exemple d'ensemble extensionnel et un exemple d'ensemble non-extensionnel (autres que ceux des questions suivantes).
3/ Parmi les ensemble suivants, lesquels peuvent être montrés indécidables à l'aide du théorème de Rice (chaque réponse doit être justifiée) ?

a. L'ensemble des programmes qui terminent sur l'entrée `nil`.

b. L'ensemble des programmes qui calculent une bijection.

c. L'ensemble des programmes WHILE dont le code contient un nombre pair de caractères.

Corrigé

1. Question de cours...
2. Extensionnel : l'ensemble des programmes qui sur l'entrée `nil` renvoient 42. Ou beaucoup d'autres exemples vu en cours et TD.
   Non extensionnel : l'ensemble des programmes dont la variable d'entrée s'appelle `X` ; l'ensemble des programmes qui utilisent 3 variables, ...
3. a. Oui. C'est extensionnel et non-trivial. Cf TD08 pour le complémentaire (programmes qui ne terminent pas sur `nil`), le raisonnement est le même.
   b. Oui. C'est non-trivial (il existe des bijections calculables, par exemple l'identité, il existe des non-bijections calculables, par exemple la fonction constante égale à `nil`). C'est extensionnel (si deux programmes calculent la même fonction, soient ils calculent tous les deux une bijection, soit aucun des deux ne calcule une bijection).
   c. Non. C'est non extensionnel. Par exemple en changeant un nom de variable de `X` à `XX`, si la variable apparaît un nombre impair de fois dans le programme, on change la parité du nombre de caractères sans changer ce qu'on calcule.

Exercice 3 : Calculabilité (4 pts)

On considère $Write_{out}$ l'ensemble des programmes WHILE qui, sur l'entrée `nil`, effectuent au moins une affectation dans leur variable de sortie. Il faut ici que l'affectation ait réellement lieu, pas simplement qu'elle soit présente dans le code.
1/ Peut-on utiliser le théorème de Rice pour montrer que $Write_{out}$ est indécidable ? Pourquoi ?
2/ Écrire un programme `P` tel que `P` $\in Write_{out}$.
3/ Écrire un programme `P'` tel que `P'` utilise `Y` comme variable de sortie et contienne au moins une ligne `Y:=E` mais `P'` $\notin Write_{out}$.
Indication : utiliser une boucle infinie pour que l'affectation ne soit jamais évaluée.
4/ Étant donné un programme quelconque `Q`, écrire un programme $\mathtt{P_Q}$ tel que $\mathtt{P_Q} \in Write_{out}$ si et seulement si $[\![\mathtt{Q}]\!](\mathtt{nil})\downarrow$.
5/ Montrer que $Write_{out}$ n'est pas décidable.

Corrigé

1. Non. C'est non extensionnel. Par exemple on peut calculer l'identité avec
   `read X; write X;`
   ou avec
   `read X; Y:=X write Y;`
   seul le deuxième programme appartient à l'ensemble.
2. `read X; Y:=X write Y;`
3. `read X; while true do {X:=X}; Y:=X write Y;`
4. On a `Q` = `read` $\mathtt{X_Q}$; $\mathtt{C_Q}$; `write` $\mathtt{Y_Q}$ et on construit $\mathtt{P_Q}$: `read X; CQ; Y:=X; write Y;`
5. Par réduction à la halte, de manière classique, à l'aide de la question précédente.

Correction

Exercice 4 : VCM (2 pts)

Soit l'instruction d'une CM
l : `X`:=$2 * (\texttt{Y} + \texttt{Z}) + 1$.
1/ Donner une suite d'instructions d'une VCM qui a le même effet (la même sémantique).

Corrigé

```
l   : A := Y+Z;
l+1 : D := 2
l+2 : B := D*A;
l+3 : U := 1;
l+4 : C := B+U;
l+5 : X := C;
```

Exercice 5 : SCM (2 pts)

1/ Écrire une suite d'instructions d'une SCM qui calcule `X + Y`.

Corrigé

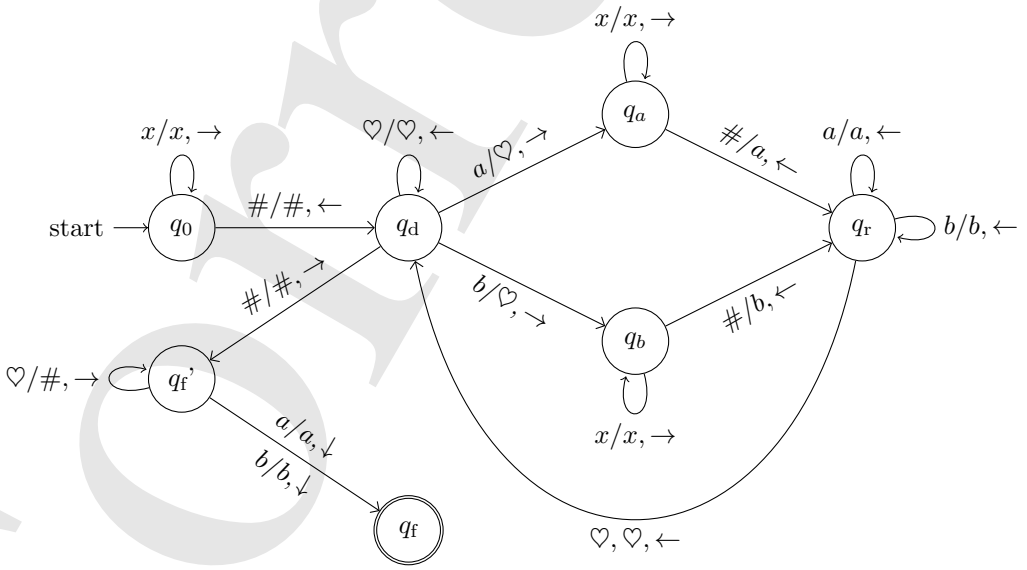
Il faut faire une "boucle" qui fait -1 sur une variable et $+1$ sur l'autre.

```
l   : if X then goto l+1 else goto l+4
l+1 : X := X-1;
l+2 : Y := Y+1;
l+3 : if X then goto l else goto l;
```

Le résultat est au final stocké dans `Y`.

Exercice 6 : Machine de Turing (3 pts)

Soit la machine de Turing sur l'alphabet $\{a, b, \heartsuit\}$ donnée par le diagramme de transitions suivant. Les x représentent soit a, soit b, soit $\heartsuit$, c'est-à-dire que chaque transition étiquetée par x correspond en fait à trois transitions similaires, l'une étiquetée par a la deuxième par b et la dernière par $\heartsuit$.



1/ Simuler l'exécution de cette machine sur l'entrée aab (donner la suite des configurations).
2/ À votre avis, quelle est la fonction calculée par cette machine ?

Correction

Corrigé

n°	état	ruban	n°	état	ruban	n°	état	ruban
0	q_0	$\underline{\text{a}}$ a b	11	q_r	a♡♡$\underline{\text{b}}$a	21	q_r	♡♡♡$\underline{\text{b}}$aa
1	q_0	a $\underline{\text{a}}$ b	12	q_r	a♡$\underline{\heartsuit}$ba	22	q_r	♡♡$\underline{\heartsuit}$baa
2	q_0	a a $\underline{\text{b}}$	13	q_d	a$\underline{\heartsuit}$♡ba	23	q_d	♡$\underline{\heartsuit}$♡baa
3	q_0	a a b$\underline{\#}$	14	q_d	$\underline{\text{a}}$♡♡ba	24	q_d	♡♡♡baa
4	q_d	a a $\underline{\text{b}}$	15	q_a	♡$\underline{\heartsuit}$♡ba	25	q_d	$\underline{\#}$♡♡baa
5	q_b	a a♡$\underline{\#}$	16	q_a	♡♡$\underline{\heartsuit}$ba	26	q_f'	$\underline{\heartsuit}$♡♡baa
6	q_r	a a$\underline{\heartsuit}$b	17	q_a	♡♡♡$\underline{\text{b}}$a	27	q_f'	#$\underline{\heartsuit}$♡baa
7	q_d	a $\underline{\text{a}}$♡b	18	q_a	♡♡♡b$\underline{\text{a}}$	28	q_f'	##$\underline{\heartsuit}$baa
8	q_a	a♡$\underline{\heartsuit}$b	19	q_a	♡♡♡ba$\underline{\#}$	29	q_f'	###$\underline{\text{b}}$aa
9	q_a	a♡♡$\underline{\text{b}}$	20	q_r	♡♡♡b$\underline{\text{a}}$a	30	q_f	$\underline{\text{b}}$aa
10	q_a	a♡♡b$\underline{\#}$						

Sur l'entrée *aab*, la machine renvoie donc *baa*.
De manière plus générale, la machine calcule le "miroir" d'une chaîne de caractères.

Exercice 7 : CM (3 pts)

On définit la suite de Fibonacci par $F_0 = F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$. On définit aussi la suite de Fibonacci modifié, comme une suite de couples définie par $F'_0 = (1, 1)$ et $F'_{n+1} = (a_{n+1}, b_{n+1}) = (b_n, a_n + b_n)$.
1/ Donner les termes F_0 à F_7.
2/ Donner les termes F'_0 à F'_6.
3/ Quelle relation remarque-t-on entre F'_n, F_n et F_{n+1} ?
4/ Écrire une CM qui sur l'entrée n calcule F_n.
Indication : utiliser une "boucle" et deux variables pour calculer F'_n.

Corrigé

1. $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, $F_6 = 13$, $F_7 = 21$.
2. $F'_0 = (1, 1)$, $F'_1 = (1, 2)$, $F'_2 = (2, 3)$, $F'_3 = (3, 5)$, $F'_4 = (5, 8)$, $F'_5 = (8, 13)$, $F'_6 = (13, 21)$.
3. On remarque que $F'_n = (F_n, F_{n+1})$
4. Pour calculer F'_n, on va utiliser deux variables qui correspondront à a_n et b_n et une boucle pour faire le bon nombre de calculs.

```
read N;
  1 : I := 0; (* compteur de boucle *)
  2 : A := 1; (* A0 *)
  3 : B := 1; (* B0 *)
  (* si I=N, on est à la fin de la boucle *)
  4 : if =? I N then goto 10 else goto 5;
  5 : TMP := B; (* sauvegarde *)
  6 : B := A+B; (* B_(I+1) *)
  7 : A := TMP; (* A_(I+1) *)
  8 : I := I+1;
  9 : if I then goto 4 else goto 4; (* fin de la boucle *)
write A;
```

Correction