Option informatique, le 8 Novembre 2005 **DS 1 - 3 heures**

Le premier exercice (plutôt sur le programme de Sup) m'a été inspiré récemment et est assez anecdotique (technique de preuve de terminaison), tandis que le deuxième étudie deux algorithmes permettant de calculer une expression rationnelle décrivant le langage reconnu par un automate. Le troisième en annexe est le problème sur les automates du concours des Mines 2004.

Les trois parties sont complètement indépendantes et seront rédigées sur des copies différentes.

"Il n'est pas tout à fait exclu que vous ne terminiez pas l'épreuve"...

1 Une preuve de terminaison

1. On munit $\mathbb{N}^2$ de la relation suivante :

$$(\alpha, \beta) \leq_L (\alpha', \beta') \iff (\alpha < \alpha' \text{ ou } (\alpha = \alpha' \text{ et } \beta \leq \beta'))$$

   (a) Vérifier que $\leq_L$ est une relation d'ordre (réflexive, antisymétrique[1] et transitive).

   (b) Montrer qu'il n'existe pas de suite infinie strictement décroissante pour $\leq_L$, i.e.

$$\cdots <_L (\alpha_n, \beta_n) <_L \cdots <_L (\alpha_1, \beta_1) <_L (\alpha_0, \beta_0)$$

   (c) Montrer que si $\alpha \geq 1$, alors on ne peut pas borner a priori la longueur d'une suite strictement décroissante de premier terme (α, β).

2. On s'intéresse au programme suivant :

```
let rec foo n=
  if n<=1
  then 0
  else 1+(if n mod 2=0 then foo (n/2) else foo (n+1));;
```

   (a) Si le calcul `foo n` termine, quelle est la valeur retournée ?

   (b) Montrer que si n est un entier, alors le calcul `foo n` termine effectivement.

   (c) Déterminer la valeur maximale du résultat retourné par `foo n` lorsque n décrit $[\![1, 91171]\!]$.

   (d) Redites-le[2].

2 Calcul du langage d'un automate

2.1 Généralités

Les types de données avec lesquels on va travailler sont :

```
type 'a expr_rat=
  | Vide
  | Eps
  | Lettre of 'a
  | Concat of ('a expr_rat)*('a expr_rat)
  | Etoile of ('a expr_rat)
  | Plus of ('a expr_rat)*('a expr_rat);;
type 'a automate={
  A :'a list;
  Q : int;
  initiaux : int list;
  finaux : int list;
  transitions: (int*'a*int)list};;
```

[1] $x \leq_L y \leq_L z \Rightarrow x \leq_L z$

[2] Si vous avez traité correctement la question précédente, vous comprendrez le sens de cette question débile !

Les automates ont leurs états indexés de 1 à $n \in \mathbb{N}$, et c'est cette seule information n qui est stockée dans le champ `Q`. Les transitions sont stockées sous forme de triplets (origine, étiquette, but), ce qui autorise le non-déterminisme (plusieurs états initiaux sont également possibles, puisqu'on dispose d'une liste d'états initiaux dans le champ `initiaux`).

1. Les deux méthodes étudiées feront intervenir des constructions automatiques d'expressions rationnelles à l'aide d'autres expressions. Lorsqu'elles sont exécutées à la main, on réalise à la volée des simplifications, du type : `Plus(Vide,e)<-e`, ou encore `Etoile(Etoile(e))<-Etoile(e)`.
   Écrire une fonction `additionner:'a expr_rat*'a expr_rat->'a expr_rat=<fun>` prenant deux expressions rationnelles e_1 et e_2 et retournant une expression rationnelle décrivant $e_1 + e_2$ (modulo des simplifications raisonnables).
2. Même chose pour la concaténation et la mise en étoile :

   ```
   concatener : 'a expr_rat * 'a expr_rat -> 'a expr_rat = <fun>
   etoiler : 'a expr_rat -> 'a expr_rat = <fun>
   ```

   Dans la suite, ces fonctions seront systématiquement utilisées dans les constructions d'expressions rationnelles.
3. Les expressions rationnelles produites étant d'assez grosse taille, on souhaite obtenir un affichage raisonnable. Écrire une fonction réalisant ce travail. Cette fonction prendra en argument une `'a expr_rat` ainsi qu'une action à réaliser (instuction d'affichage) sur les lettres de type `'a`.

   ```
   ecrire_exp_rat_general : ('a -> unit) -> 'a expr_rat -> unit = <fun>
   ```

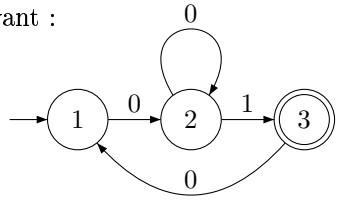
   Typiquement, si `e` est une expression rationnelle sur l'alphabet $\{0, 1\}$, on pourra obtenir :

   ```
   #ecrire_exp_rat_general print_int e;;
   0(00*10)*- : unit = ()
   ```

   Votre fonction ne parenthèsera pas forcément de façon minimale (comme dans l'exemple précédent), mais il ne faudra pas qu'il y ait d'ambigüité dans l'écriture produite.

2.2 Méthode d'Arden

1. Montrer que si K et M sont deux langages sur l'alphabet A, avec $\varepsilon \notin K$, alors il existe un seul langage L tel que $L = KL + M$: il s'agit de K^*M.
2. Soit $\mathcal{A} = (A, Q, I, F, \Delta)$ un automate (sans ε-transition), avec $Q = \{1, 2, \ldots, n\}$. I est l'ensemble des états initiaux, F est l'ensemble des états finaux, et $\Delta \subset Q \times A \times Q$ est l'ensemble des transitions. Pour $i \in Q$, on définit L_i comme l'ensemble des mots étiquetant au moins un chemin allant de i à un état final de $\mathcal{A}$.
   (a) Exprimer $L(\mathcal{A})$ à l'aide des L_i.
   (b) Montrer :
   $$\forall i \in [\![1, n]\!], \qquad L_i = M_i + \sum_{j=1}^{n} K_{i,j} L_i \tag{E_i}$$
   avec $M_i = \{\varepsilon\}$ si $i \in F$ et $M_i = \emptyset$ sinon, et avec $K_{i,j} = \{\alpha \in A \mid (i, \alpha, j) \in \Delta\}$.
   La méthode d'Arden consiste à appliquer le lemme de la question 1 à la n-ième équation, puis réinjecter dans la $(n-1)$-ième et re-Ardeniser... jusqu'à obtenir un sytème triangulaire inférieur (strict). On obtient alors une expression rationnelle décrivant L_1 (ne faisant pas intervenir les autres L_i). On réinjecte dans L_2, pour obtenir L_2, et ainsi de suite jusqu'à L_n.
3. On considère l'automate $\mathcal{A}_1$ suivant :



   Écrire les trois équations aux langages vérifiées par les L_i, puis résoudre ce système, et enfin donner une expression simple de $\mathcal{L}(\mathcal{A}_1)$.

4. (a) Écrire en pseudo-code les opérations réalisées sur les coefficients du système lors de la mise sous forme triangulaire.
   (b) Justifier l'utilisation du lemme d'Arden à chaque étape.
   (c) Écrire enfin en pseudo-code les opérations réalisées lors des substitutions de la seconde phase de la résolution (injections successives de L_1 jusqu'à L_n).
5. Le squelette de la méthode d'Arden est le suivant :

```
let arden a=
  let n=a.Q in
  let K=make_matrix (n+1) (n+1) Vide
  and M=make_vect (n+1) Vide
  in
    begin
      initialise_M M ...;
      initialise_K K ...;
      for l=n downto 1
      do
         (* Mise sous forme triangulaire inferieure *)
      done;
      for l=2 to n
      do
         (* Injections successives *)
      done;
      reconstituer_L .......
    end;;
```

   `K` est le vecteur de vecteurs d'expressions rationnelles contenant les différents coefficients des équations (initialement, les $K_{i,j}$), et M est le vecteur contenant les expressions rationnelles "libres" des différentes équations (initialement, les M_i). Les longueurs de $n+1$ permettent une indexation de 1 à n...
   (a) Écrire une fonction `initialise_M : 'a expr_rat vect -> biiiip -> unit = <fun>` initialisant (par effet de bord) le vecteur `M` à l'aide de biiiip.
   (b) Écrire une fonction `initialise_K : 'a expr_rat vect vect -> biiiip -> unit = <fun>` initialisant le vecteur `K` à l'aide de biiiip.
   (c) Écrire une fonction `reconstituer_L : (.......) -> 'a expr_rat = <fun>` prenant en entrée... ce qu'il faut... et retournant une expression rationnelle décrivant le langage associé à l'automate avec lequel on travaille.
   (d) Compléter la fonction `arden : 'a automate -> 'a expr_rat = <fun>` calculant une expression rationnelle décrivant le langage reconnu par un automate.

2.3 Une autre méthode

En supposant toujours les états d'un automate étiquetés de 1 à n, on note $L_{i,j,k}$ l'ensemble constitué des mots étiquetant des chemins de i à j en ne passant que par des états (strictement) intermédiaires inférieurs ou égaux à k, ceci pour $(i,j,k) \in [\![1,n]\!] \times [\![1,n]\!] \times [\![0,n]\!]$.

1. Dire ce que vaut $L_{i,j,0}$.
2. Pour $k \in [\![0,n-1]\!]$, exprimer $L_{i,j,k+1}$ à l'aide de langages $L_{\alpha,\beta,k}$. Prouver ladite relation (preuve non lue en l'absence de dessin...).
3. Écrire une fonction `initialise : 'a expr_rat vect vect -> biiiip -> unit = <fun>` initialisant par effet de bord les $L_{i,j,0}$.
4. Écrire une fonction `reconstituer_langage : (.....) -> 'a expr_rat = <fun>` calculant $\mathcal{L}(\mathcal{A})$ à partir de...
5. Enfin, écrire une fonction `pas_arden : 'a automate -> 'a expr_rat = <fun>` calculant une expression rationnelle décrivant le langage reconnu par un automate à l'aide de la méthode décrite dans cette partie.

2.4 Calcul des coûts

1. Donner (en justifiant) une définition raisonnable de la *taille* d'une expression rationnelle.
2. Évaluer, dans la seconde méthode, la taille des expressions manipulées, ainsi que le temps global d'exécution.
3. On s'intéresse maintenant à la première méthode (à la Arden) :
   (a) Que dire de la taille des $K_{i,j}$ et M_i avant de résoudre le système ?
   (b) Évaluer la taille des $K_{n,i}$ et M_n après la première Ardenisation, puis celle des $K_{n-1,i}$ et M_{n-1} après la deuxième... et ce jusqu'à la taille de L_1 après la n-ième ardenisation.
   (c) Évaluer la taille de L_2 après la première substitution dans la remontée du système, puis L_3... et ce jusqu'à L_n.
   (d) Évaluer la taille de l'expression rationnelle obtenue à l'aide de la méthode d'Arden, et le temps total d'exécution.
4. Et donc ?

3 Longueur discriminante de deux automates

Cf document annexe.