Langages, interprétation, compilation – Partiel

Durée 2 heures. Tous documents autorisés.

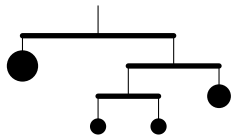
1. Récurrence

Un *mobile* est formé par un ensemble d'objets, suspendus à des barres elles-mêmes suspendues en équilibre à d'autres barres, et ainsi de suite jusqu'à un unique point de suspension auquel pend l'ensemble de la structure. Les mobiles ont une structure récursive. On peut les décrire à l'aide de la signature comportant les éléments suivants :

— pour tout nombre x, un symbole O_x d'arité 0 désignant un objet de masse x,

— pour tout nombre x, un symbole B_x d'arité 2 tel que $\mathsf{B}_x(m_1, m_2)$ désigne la combinaison de deux mobiles m_1 et m_2 à l'aide d'une barre de masse x.

Ainsi, le mobile dessiné ci-dessous à gauche peut être représenté par le terme à droite, où les masses reflètent approximativement les tailles des éléments dessinés.



$$\mathsf{B}_3(\mathsf{O}_8, \mathsf{B}_2(\mathsf{B}_1(\mathsf{O}_1, \mathsf{O}_1), \mathsf{O}_3))$$

La masse d'un mobile est la somme des masses de tous ses éléments. On dit qu'un mobile est *équilibré* si pour chacune de ses barres, la masse du sous-mobile de gauche est égale à la masse du sous-mobile de droite. Le mobile donné en exemple ci-dessus est équilibré.

Questions

1. Dessiner le mobile représenté par le terme $\mathsf{B}_{x_1}(\mathsf{B}_4(\mathsf{B}_{x_2}(\mathsf{O}_1, \mathsf{O}_{x_3}), \mathsf{O}_4), \mathsf{B}_{x_4}(\mathsf{O}_5, \mathsf{B}_1(\mathsf{O}_{x_5}, \mathsf{O}_{x_6})))$, et préciser des valeurs des x_i telles que ce mobile soit équilibré.
2. On note $b(m)$ le nombre de barres d'un mobile m, et $o(m)$ le nombre d'objets suspendus à un mobile m. Démontrer par récurrence que pour tout mobile m on a $o(m) = b(m) + 1$.
3. Donner des équations récursives définissant $\mathsf{masse}(m)$, la masse totale du mobile m.
4. Les équations suivantes définissent une fonction des mobiles vers les mobiles.

$$\begin{array}{rcl} \mathsf{alourdir}(\mathsf{O}_x, y) & = & \mathsf{O}_{x+y} \\ \mathsf{alourdir}(\mathsf{B}_x(m_1, m_2), y) & = & \mathsf{B}_{x+y/3}(\mathsf{alourdir}(m_1, y/3), \mathsf{alourdir}(m_2, y/3)) \end{array}$$

   Montrer que $\mathsf{masse}(\mathsf{alourdir}(m, x)) = \mathsf{masse}(m) + x$.

5. Proposer un type caml `mobile` pour représenter les mobiles. Donner une définition caml de la fonction `masse`. Définir également une fonction caml `stable` telle que `stable`(m) renvoie `true` si le mobile m est équilibré et `false` sinon.

2. Automates

On considère l'alphabet $A = \{a, b, r\}$ et le langage L des mots sur A contenant au moins une occurrence de la séquence « *babar* ».

Questions

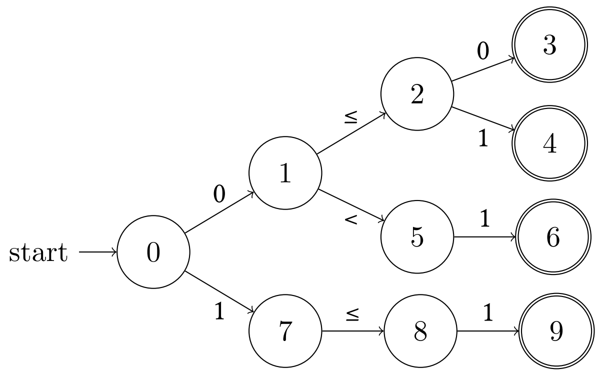
— Donner une expression régulière pour L.

— Donner un automate déterministe et complet reconnaissant L.

3. Langages reconnaissables

On considère l'alphabet $A = \{0, 1, <, \leq\}$. Les mots sur A permettent notamment d'écrire des chaînes de comparaisons de nombres binaires, comme $101 \leq 1000 < 1001 \leq 1001$.

L'automate ci-dessous reconnaît le langage L_1 des mots sur A formant une comparaison valide de deux nombres à 1 chiffre.



Questions

1. Donner une expression régulière reconnaissant L_1.
2. Dans l'automate ci-dessus, fusionner les états qui peuvent l'être pour obtenir un automate déteministe reconnaissant L_1 avec aussi peu d'états que possible (on ne veut pas forcément un automate complet).
3. Soit L_2 le langage des chaînes de comparaisons valides de nombres à 1 chiffre. L_2 contient donc par exemple $0 \leq 0 < 1 \leq 1$ ou $0 \leq 1 \leq 1$. Le langage L_2 est-il reconnaissable ? Donner une démonstration adéquate.
4. Soit L_3 le langage des comparaisons valides de deux nombres de tailles quelconques. L_3 contient donc par exemple $101 < 1001$ mais pas $110 \leq 101$. Le langage L_3 est-il reconnaissable ? Donner une démonstration adéquate.

4. Grammaires

Voici 5 propositions différentes d'une grammaire pour décrire des séquences d'instructions séparées par des points-virgules. Dans tous les cas, on considère un symbole terminal ; (point-virgule), et deux symboles non terminaux *instr* et *seq*, respectivement pour les instructions et pour les séquences. Le symbole ε désigne une séquence vide.

$$G_a \quad seq \longrightarrow seq\,;\,seq \;\mid\; instr$$

$$G_b \quad seq \longrightarrow seq\ seq \;\mid\; instr\,;$$

$$G_c \quad seq \longrightarrow instr\,;\,seq \;\mid\; \varepsilon$$

$$G_d \quad seq \longrightarrow seq\,;\,instr \;\mid\; \varepsilon$$

$$G_e \quad seq \longrightarrow instr\,;\,seq \;\mid\; instr\,;$$

Questions

1. Parmi les grammaires proposées, lesquelles permettent de dériver la séquence suivante ?

   instr ; *instr* ; *instr* ; *instr* ;

   Donner un arbre de dérivation pour chacune.
2. Parmi les grammaires proposées, lesquelles sont ambigues ?
3. Identifier parmi les propositions au moins deux grammaires décrivant exactement le même langage.
4. Pour chaque grammaire qui n'a pas été citée à la question précédente, donner un exemple d'une séquence qui est dérivable avec cette grammaire mais n'est dérivable avec aucune des autres propositions.

5. Interprétation

On se donne une syntaxe abstraite minimale pour des expressions arithmétiques avec variables locales, ainsi qu'une fonction d'évaluation écrite en caml.

```
type expr =
  | Cst of int
  | Add of expr * expr
  | Var of string
  | Let of string * expr * expr

let eval e =
  let env = Hashtbl.create 32 in
  let rec eval = function
    | Cst n -> n
    | Add(e1, e2) ->
       let v1 = eval e1 in
       let v2 = eval e2 in
       v1 + v2
    | Var x ->
       let e = Hashtbl.find env x in
       let v = eval e in
       Hashtbl.replace env x (Cst v);
       v
    | Let(x, e1, e2) ->
       Hashtbl.add env x e1;
       let v = eval e2 in
       Hashtbl.remove env x;
       v
  in
  eval e
```

Rappel des effets des fonctions sur les tables de hachage (`Hashtbl`) utilisées :

— `Hashtbl.create` n crée une nouvelle table de taille n (la taille étant susceptible d'être ajustée automatiquement plus tard).

— `Hashtbl.add` t x v ajoute une nouvelle association entre la clé x et la valeur v dans la table t, sans détruire les éventuelles autres associations déjà présentes pour x.

— `Hashtbl.find` t x renvoie la dernière valeur associée à x dans la table t.

— `Hashtbl.remove` t x retire de la table t la dernière association pour x.

— `Hashtbl.replace` t x v est équivalent à à l'enchaînement
`Hashtbl.remove` t x`;` `Hashtbl.add` t x v.

Ainsi, si on réalise l'enchaînement

```
let t = Hashtbl.create 32 in
Hashtbl.add t "x" 1;
Hashtbl.add t "x" 2;
Hashtbl.replace t "x" 3;
let a = Hashtbl.find t "x" in
Hashtbl.remove t "x";
let b = Hashtbl.find t "x" in
...
```

on obtiendra pour `a` la valeur 3 et pour `b` la valeur 1.

Questions

1. On considère l'expression e suivante :

   ```
   let x = 1 + 2 in
   let y = 1 + 4 in
   x + x
   ```

   Détailler l'évaluation de l'expression e, en précisant à chaque étape le contenu de la table de hachage. Combien de fois est réalisée l'addition `1+2` ? Combien de fois est réalisée l'addition `1+4` ? Reconnaissez-vous le comportement d'une stratégie d'évaluation particulière ?

2. À quoi sert la ligne

   ```
   Hashtbl.replace env x (Cst v);
   ```

   du cas `Var` ? Que se passerait-il si on l'enlevait ? L'évaluateur serait-il encore correct ? (justifier ou donner un contre-exemple)

3. À quoi sert la ligne

   ```
   Hashtbl.remove env x;
   ```

   du cas `Let` ? Que se passerait-il si on l'enlevait ? L'évaluateur serait-il encore correct ? (justifier ou donner un contre-exemple)

4. On veut reformuler cet évaluateur pour qu'il ne repose plus la modification d'une structure de données mutable. Définir des équations pour une fonction eval, telle que $\mathsf{eval}(e, \rho)$ évalue e dans l'environnement ρ et renvoie la valeur calculée *et* le nouvel environnement ρ' reflétant les modifications éventuelles de ρ. Le format de réponse est au choix : une définition mathématique ou du code caml.