Systèmes Synchrones
MPRI – cours 2.23-1 2017-2018

Examen
28 novembre 2017

L'énoncé est composé de 6 pages. Cette épreuve est prévue pour une durée de 3h. Les notes de cours sont autorisés.

Le sujet comporte volontairement peu de questions. Elles sont la pour vous aider à atteindre l'objectif annoncé. Vous avez toute liberté de suivre une autre voie. Vous veillerez à décrire vos solutions avec précision, c'est-à-dire, en justifiant la complexité en temps et espace des algorithmes, les conditions d'applications et les limites éventuelles. Si cela vous est utile, les algorithmes pourront être écrits en OCaml.

La machine `grep` de Raymond

L'objectif de ce problème est de découvrir l'algorithme, dû à Raymond [1] de reconnaissance d'expressions régulières à partir d'un circuit synchrone. Etant donnée une expression régulière E de taille n, cet algorithme construit en $O(n)$ un réseau data-flow booléen de taille $O(n)$ qui reconnaît les mots du langage $L(E)$ engendré par E.

> **La machine "grep":** Soit vocabulaire Σ fini et L, un langage régulier sur Σ. La machine "grep" est une machine qui reçoit une séquence $s_0.s_1....s_n...$ de lettres $s_i \in \Sigma$ et qui calcule une séquence $b_0.b_1....b_n...$ de booléens, tels que b_n est vrai si et seulement si le mot $s_0.s_1...s_n$ appartient à L.

Rappels et notations: Une machine de Mealy est un automate à entrée/sorties caractérisé par $M = (i, Q, I, O, \delta, \lambda)$ où (1) Q est un ensemble fini d'états; (2) $i \in Q$ est l'état initial; (3) I est un alphabet fini d'entrées; (4) O est un alphabet fini de sorties; (5) $\delta : Q \times I \to Q$ est la fonction de transition; (6) $\lambda : Q \times I \to O$ est la fonction de sortie. On note $p \xrightarrow{a/b} q$ lorsque $\delta(p, a) = q$ et $\lambda(p, a) = b$.

Dans la suite, on s'intéressera aux langages réguliers engendrés par les expressions régulières suivantes, où $a \in \Sigma$ désigne une lettre.

$$E ::= a \mid E.E \mid E + E \mid E^* \mid E^\epsilon \mid (E)$$

Le langage engendré $L(E)$, c'est-à-dire l'ensemble des mots de Σ^*, est défini par:

$$\begin{array}{lll} L(a) = \{a\} & L(E + F) = L(E) \cup L(F) & L(E.F) = L(E).L(F) \\ L(E^*) = \bigcup_{i=0}^{\infty} L(E^i) & L(E^\epsilon) = \{\epsilon\} \cup L(E) & L((E)) = L(E) \end{array}$$

où $E^0 = \{\epsilon\}$ et $E^n = E^{n-1}.E$. ϵ désigne le mot vide (mot de longueur nulle). On utilisera des parenthèses pour éviter les ambiguités d'écriture.

Exemples

Question 1 Écrire la machine de Mealy qui reconnaît le langage engendré par l'expression $(abc)^*a$. Vous prendrez $I = \{a, b, c\}$ et $O = \{0, 1\}$.

Question 2 On souhaite reconnaître le sens de rotation d'une roue en observant l'alternance de trois valeurs a, b et c. On dit qu'elle tourne dans le sens direct lorsque l'entrée est de la forme $...a\,b\,c\,a\,b\,c\,a\,b\,c...$, indirect lorsqu'elle est de la forme $...c\,b\,a\,c\,b\,a\,c\,b\,a\,c\,b\,a...$, et non défini sinon. Spécifier sous forme d'expression régulière (1) le fait que les entrées arrivent dans le sens direct; (2) le fait que les entrées arrivent dans le sens indirect.

Question 3 Écrire l'automate de Mealy d'entrées $I = \{a, b, c\}$ et de sortie $O = \{0, 1\}$ le reconnaisseur des mots de (1) et le reconnaisseur des mots de (2). Ansi, l'automate de Mealy de (1) répondra $0\,0\,0\,1\,1\,1\,0$ sur l'entrée $a\,a\,b\,c\,a\,b\,a$. Écrire l'automate de Mealy d'entrées $I = \{a, b, c\}$ et de sortie $O = \{d, i, n\}$ (d = "direct"; i = "indirect"; n = "non défini") qui produit la sortie d lorsque l'entrée arrive dans le sens direct; i lorsque l'entrée arrive dans le sens indirect; n, sinon.

Question 4 Écrire sous forme de programme Lustre booléen d'interface:

```
node sens_de_rotation(a, b, c: bool) returns (d, i, n: bool)
```

le système qui détermine le sens de rotation. Vous pourrez le définir sous la forme de deux sous-systèmes (deux noeuds Lustre), l'un qui détermine si l'entrée apparaît dans le sens direct, l'autre qui détermine si l'entrée apparaît dans le sens indirect.

Des expressions régulières aux systèmes d'équations linéaires

Question 5 Étant donnée une expression régulière, définir la fonction $vide(.) : E \mapsto \{false, true\}$ telle que $vide(E)$ renvoie la valeur $true$ si et seulement si $\epsilon \in L(E)$.

```
Vide(a) = false
Vide(E ^ epsilon) = true
Vide(E ^ *) = true
Vide(E+F) = Vide(E) or Vide(F)
Vide(E.F) = Vide(E) and Vide(F)
```

On étudie maintenant la traduction des expressions régulières vers des règles de grammaire récursives à gauche. Pour cela, on introduit la syntaxe concrète suivante:

$$
\begin{array}{lll}
\mathit{system} & ::= & X = \mathit{terms} \\
\mathit{terms} & ::= & \epsilon \mid X \mid X.a \mid \mathit{terms} + \mathit{terms} \\
 & & \mid \mathsf{let}\ X = \mathit{terms}\ \mathsf{in}\ \mathit{terms} \mid \mathsf{rec}\ X = \mathit{terms}
\end{array}
$$

rec $X = t$ est un raccourci pour let $X = t$ in X. Par exemple, rec $X = Xabc + a$ correspond à l'écriture d'une grammaire en notation BNF:

$$X ::= Xabc \mid a$$

qui définit le langage régulier $a(abc)^*$. Si t est un terme ($t \in terms$), on note $L(t)$ le langage engendré par la grammaire $X = t$ (où X n'apparaît pas dans t). Par définition $L(X) = L(t)$.

L'objectif est maintenant de construire une fonction $Trad(.) : E \mapsto terms$. Cette fonction pourra produire, à partir de $(a^* + b)^*.a$, par exemple, le système d'équations:

$$X = Y.a \quad Y = T \quad T = Z + W + T.b \quad W = T + W.a \quad Z = \epsilon$$

où X est le symbole de départ.

Écrit dans la syntaxe du langage défini ci-dessus, $Trad((a^* + b)^*.a)$ produira:

$$\text{let } Z = \epsilon \text{ in let } Y = \text{rec } T = Z + (\text{rec } W = T + W.a) + T.b \text{ in } Y.a$$

Question 6 Étant donnée une expression E, on veut définir une fonction $TradAux(.)(.)$ de traduction vers un terme. Cette fonction utilise un non terminal auxiliaire. Intuitivement, si X est un non terminal, alors $TradAux(X)(E)$ reconnaît le même langage que X suivi du langage reconnu par E. Précisément, $TradAux(X)(E)$ renvoie un élément $t \in terms$ tel que $L(t) = L(X).L(E)$.

Définir la fonction $TradAux(X)(E)$ par cas suivant la structure de E. En déduire la fonction $Trad(E)$.

```
Trad(X,a) = X.a
Trad(X,E^epsilon) = X + Trad(X, E)
Trad(X, E+F) = Trad(X, E) + Trad(X, F)
Trad(X, E.F) = let Y = Trad(X,E) in Trad(Y, F) ou Y est neuf
Trad(X, E^*) = rec Y = X + Trad(Y, E) ou Y est neuf
Trad(E) = (let Z = epsilon in Trad(Z, E))
```

Question 7 Donner et justifier la complexité en temps et en espace de $Trad(E)$ par rapport à la taille de E. La taille est celle de l'expression renvoyée par $Trad(E)$.

```
Soit o, le nombre d'occurrences de symboles, d le nombre de ".", e, le nombre
de puissances (epsilon) et k, le nombre d'etoiles de Kleene. Alors, la taille
de l'automates non deterministe avec transitions epsilon est caracterisee par:

|Q| = d + k + 2
|T_Sigma| (nombre de transitions) = o
|T_espilon| (nombre de transitions epsilon) = k + e
```

Réseaux booléens

On définit la syntaxe des réseaux booléens ainsi:

$$\begin{array}{rcl} net & ::= & X \mid \mathit{true} \mid \mathit{false} \mid \mathsf{not}\ net \mid net\ \mathsf{and}\ net \mid net\ \mathsf{or}\ net \mid net\ \mathsf{fby}\ net \\ & & \mid \mathsf{let}\ X = net\ \mathsf{in}\ net \mid \mathsf{rec}\ X = net \end{array}$$

N'importe quel système d'équations booléennes peut s'y traduire. E.g.,

$$S = X\ \mathsf{and}\ \mathsf{not}\ Y \quad X = A\ \mathsf{and}\ B \quad Y = \mathit{false}\ \mathsf{fby}\ X\ \mathsf{or}\ Y$$

correspond à:

$$\mathsf{let}\ X = A\ \mathsf{and}\ B\ \mathsf{in}\ X\ \mathsf{and}\ \mathsf{not}\ \mathsf{rec}\ Y = \mathit{false}\ \mathsf{fby}\ X\ \mathsf{or}\ Y$$

On ne considère que les réseaux booléens sans boucle combinatoire. Toute définition récursive doit apparaître à droite de l'opérateur fby et on dira alors que le réseau est correct. La sémantique d'un réseau booléen est celle de Lustre. *false* fby X est un raccourci pour `false -> pre(X)`.

Etant donné un réseau booléen dont les variables libres sont $I = \{X_1, ..., X_l\}$ (c'est-à-dire non liées par un let ou rec), une valuation $\rho : I \mapsto \{\mathit{false}, \mathit{true}\}$ associe une valeur booléenne à chacune de ces variables. Une trace est une séquence finie de valuations $\rho_1...\rho_k$. Soit une trace d'entrée $\rho_1....\rho_k$ (des variables de I), l'exécution du réseau $n \in net$ produit une séquence $b_1...b_k$ où $b \in \{\mathit{false}, \mathit{true}\}$. On dira qu'un réseau $n \in net$ reconnait la trace $\rho_1...\rho_k$ si $b_k = \mathit{true}$.

Question 8 Quel serait le réseau booléen correspondant au système linéaire:

$$X = (\mathsf{rec}\ Y = \epsilon + Y.b)a$$

(et qui définit le langage régulier $(b)^*a$)

```
(true fby (y)) and a and y = true fby y and b
```

Question 9 Définir la fonction *TradBoolean*(.) telle que *TradBoolean*(t) renvoie un réseau booléen n tel que $L(t) = L(n)$.

```
Soit system, le system d'equations initial. On definit Initial(system), l'ensemble
des non terminaux X tels que epsilon in L(X). Autrement dit, X reconnait le mot vide.

Trad(epsilon) = false
Trad(X) = X
Trad(tau + tau') = Trad(tau) or Trad(tau')
Trad(X.a) = (true fby X) and a if X in Initial(system)
Trad(X.a) = (false fby X) and a otherwise
Trad(let X = tau in tau') = (let X = Trad(tau) in Trad(tau'))
```

Question 10 Quel est le réseau booléen correspondant à l'expression régulière $((a^*+b)^*.a)$. Le réseaux booléen est-il correct?

```
X = (false fby Y) and a
Y = T
T = Z or W or (true fby T) and b
W = T or (true fby W) and a
Z = false

Non. Il y a une boucle combinatoire W < T < W.
```

On peut montrer qu'une boucle combinatoire peut apparaître dès que E contient une sous-expression de la forme $(F)^*$ avec $vide(F)$.

Question 11 Définir la fonction $Norm(.)$ telle que $Norm(E)$ renvoie une nouvelle expression E', équivalente à E mais qui ne contient plus de sous-expression de la forme $(F)^*$ avec $vide(F)$.

Par exemple $Norm((a^* + b)^*) = (a + b)^*$; $Norm(((a^*)^*)^*) = a^*$.

```
Norm(a) = a
Norm(E+F) = Norm(E)+Norm(F)
Norm(E.F) = Norm(E).Norm(F)
Norm(E^epsilon) = (Norm(E)) ^ epsilon
Norm(E^*) = (NormBis(E))^*

NormBis(E) = Norm(E) si not (vide(E)
NormBis(E+F) = NormBis(E)+NormBis(F) si vide(E+F)
NormBis(E.F) = NormBis(E) + NormBis(F) si vide(E.F)
NormBis(E^epsilon) = NormBis(E)
NormBis(E^*) = NormBis(E)
```

Question 12 Écrire une fonction $TradExpBoolean(.)$ telle que $TradExpBoolean(E)$ renvoie le réseau booléen équivalent à E. Quelle est sa complexité en temps et en espace (taille du réseau produit vis-à-vis de la taille de l'expression booléenne)?

```
Phi(E) = let Z = false in Psi(Z, true, E)

Psi(X, b, a) = (b fby X) and a
Psi(X, b, E+F) = Psi(X, b, E) or Psi(X, b, F)
Psi(X, b, E ^ \epsilon) = X or Psi(X, b, E)

Soit Y, un identificateur neuf.

Psi(X, b, E.F) = let Y = Psi(X, b, E) in Psi(y, b and Vide(E), F)
```

```
Psi(X, b, E ^ star) = rec Y = X or Psi*(Y, b, E)

Psi*(X, b, E) = Psi(X, b, E) si not Lambda(E)
Psi*(X, b, E+F) = Psi*(X, b, E) or Psi*(X, b, F) si Vide(E+F)
Psi*(X, b, E.F) = Psi*(X, b, E) or Psi*(X, b, F) si Vide(E.F)
Psi*(X, b, E ^ epsilon) = Psi*(X, b, E)
Psi*(X, b, E ^ star) = Psi*(X, b, E)
```

Nous vous recommandons la lecture de l'article de Raymond [1]. A partir de celui-ci, Raymond a proposé un langage de description de propriétés temporelles [2]. Il est à l'origine du langage Stimulus développé depuis peu par la jeune pouce Argosim [1]

References

[1] P. Raymond. Recognizing regular expressions by means of dataflows networks. In *23rd International Colloquium on Automata, Languages, and Programming, (ICALP'96)*, Paderborn, Germany, July 1996. LNCS 1099, Springer Verlag.

[2] Pascal Raymond, Yvan Roux, and Erwan Jahier. Lutin: a language for specifying and executing reactive scenarios. *EURASIP Journal on Embedded Systems*, 2008, 2008. Article ID 753821.

[1] `https://hal.archives-ouvertes.fr/hal-01292286/document`