Examen de Bases formelles du TAL

Pierre-Léo Bégay

14 mai 2018

Les bonus, moins rentables, sont à garder pour la fin. De façon générale, n'hésitez pas à optimiser votre temps, notamment en ne faisant pas les exercices dans l'ordre et en sautant momentanément des questions (en laissant de place pour y revenir)

1 Algorithmique des expressions régulières [5 points]

Soient les expressions suivantes :

$e_1 = a(bb + \epsilon)a$

$e_2 = a^*(ba^*)^*$

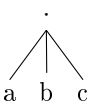
$e_3 = b(aa)^*(bb + \epsilon)$

$e_4 = bab + (abb)^*(ba + \epsilon)$

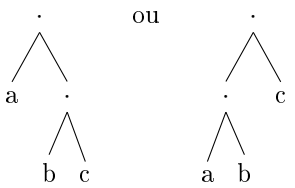
Question 1 [0,5 point]

Dressez l'arbre syntaxique de chacune de ces expressions

Note la concaténation d'expressions rationnelles étant associative, on s'autorisera à utiliser n'importe quel nombre de descendants à un $\cdot$. Concrètement, une expression comme abc pourra être représentée comme



au lieu de



Note Ne perdez pas votre temps à faire des arbres propres, tant que c'est lisible ça ira.

Question 2 [1 point]

Lesquelles contiennent ϵ dans leur langage ? Pas de justification demandée.

Question 3 [3,5 points]

On veut écrire une fonction E qui, étant donnée une expression rationnelle e, renvoie *true* si le langage qu'elle décrit contient ϵ, et $false$ sinon. Pour ça, on veut utiliser un algorithme récursif, qui regarde la racine de l'arbre syntaxique de e et prend une décision en fonction de ce qu'il y trouve.

On a donc 5 cas à définir. Les 2 cas terminaux sont ceux où e est ϵ ou une lettre quelconque. Les 3 cas récursifs sont ceux sont e est composée de 2 sous-expressions unies par un $+$ ou un $\cdot$, ou e est composée d'une seule sous-expression qui est *sous* une étoile de Kleene $*$.

Le squelette d'algorithme ci-dessous sépare donc ces 5 cas[1], à vous de compléter chaque *return* **(sur votre copie ou directement sur le sujet)**. Notez que dans les cas récursifs, vous pouvez utiliser les sous-expressions.

Data: Une expression régulière e
Result: Un booléen : *true* si $[\![e]\!]$ contient ϵ, $false$ sinon
if $e \equiv \epsilon$ **then**
    return ...
else if $e \equiv a$ *(pour n'importe quel* $a \in \Sigma$*)* **then**
    return ...
Si e est de la forme $e_1 + e_2$
e_1 et e_2 sont *utilisables*. On pourrait par exemple faire *return* $E(e_1)$
else if $e \equiv e_1 + e_2$ **then**
    return ...
else if $e \equiv e_1 \cdot e_2$ **then**
    return ...
else if $e \equiv e_1^*$ **then**
    return ...

Algorithm 1: La fonction E(e), qui évalue la présence de ϵ dans $[\![e]\!]$

2 Algo des grammaires algébriques [6 points]

On rappelle qu'un symbole non-terminal A est dit **sans contribution** s'il n'existe pas de dérivation $A \rightarrow^* u$ avec $u \in \Sigma^*$, c'est-à-dire si aucune dérivation partant de A *arrive* sur un mot composé uniquement de symboles terminaux.

Question 1 [2 points]

Soit $G = \langle \{a, b\}, \{S, A, B, C, D\}, S, \{$

[1] La présentation ici est proche du *pattern matching* qu'on trouverait dans un langage comme Ocaml, mais on peut coder quelque chose de similaire en python avec la primitive *isinstance(objet,classe)*

$S \to AA \mid BB$

$A \to CaC \mid aca$

$B \to AB \mid BA \mid CD \mid DC$

$C \to AcA \mid cac$

$D \to dCCB\}\rangle$

Donnez l'ensemble des symboles sans contribution de G (pas de justification demandée)

Question 2 [4 points]

On propose l'algorithme suivant pour calculer l'ensemble des non-terminaux sans contribution d'une grammaire :

Data: Une grammaire algébrique (Type 2) $G = \langle \Sigma, V, S, R \rangle$
Result: L'ensemble des non-terminaux sans contribution de V
\# N_0 contient l'ensemble des non-terminaux qui peuvent immédiatement se
\# réécrire en mot composé uniquement de terminaux
$N_0 := \{A \in V \mid A \to \alpha, \alpha \in \Sigma^*\}$
$i := 0$
repeat
  $i := i + 1$
  $N_i := N_{i-1}$ # on met tout N_{i-1} dans N_i déjà
  # On parcourt l'ensemble des règles de dérivation
  for $(A \to u) \in R$, *avec* $A \in V$ *et* u $(V \cup \Sigma)^*$ **do**
    #On parcourt l'ensemble des non-terminaux apparaissant dans la partie
    #droite de la règle
    for B *(non-terminal) apparaissant dans* u **do**
      if $B \in N_{i-1}$ **then**
        ajouter A à N_i
      end
    end
  end
until $N_i = N_{i-1}$;
return N_i

Algorithm 2: (Mauvais) calcul des symboles sans contribution de la grammaire G

Cet algorithme est incorrect, dans le sens où il ne réalise pas sa spécification. Donnez un contre-exemple concret (c'est-à-dire une grammaire ainsi qu'un symbole sans contribution pas renvoyé par l'algorithme, ou à l'inverse un symbole avec contribution qui fera partie de l'ensemble renvoyé), et expliquez comment le corriger.

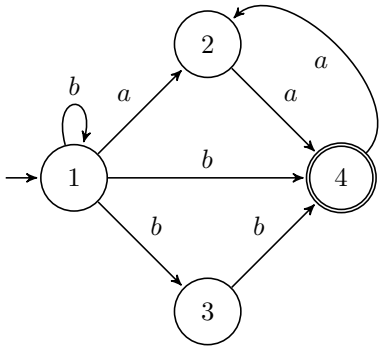
Bonus Expliquez en une phrase comment calculer les états sans contribution d'un automate fini, et comparez au 'bug' de l'algorithme précédent.

3 Automates [6 points]

Question 1 [4 points]

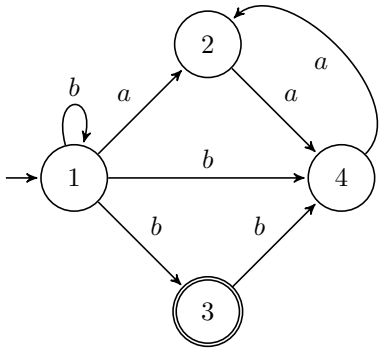
Décrivez, à l'aide d'une expression rationnelle, le langage reconnu par l'automate suivant. Justifiez votre réponse, soit à l'aide de l'application d'un algorithme, soit en

décrivant l'ensemble des chemins possibles (comme vu en cours).



Question 2 [2 points]

Même question avec cet automate :



4 Grammaire mystère [3 points + 1 en bonus]

Soit $G = \langle \{a, b\}, \{A, B, C, D\}, A, \{$

$A \to aB \mid bD$

$B \to bC \mid aA \mid \epsilon$

$C \to aD \mid bB$

$D \to bA \mid aC \} \rangle$

Décrivez, en langue naturelle, le langage engendré par G.

Indice Vous pouvez vous appuyer sur un exemple vu en cours

Question bonus [1 point] Donnez une expression régulière décrivant le langage en question