Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
IFT-3101

Danny Dubé
Version: Hiver 2013

Exercices reliés au chapitre 3

Exercices

Voici les exercices que je recommande de faire:

- **Exercice 3.1.1**. (L'exercice 3.3 (b) dans la 1ère édition est similaire.)
- **Exercice 3.3.2**. (Exercice 3.6 dans la 1ère édition où les `0` et les `1` ont été remplacés par des `a` et des `b`.)
- **Exercice 3.3.3**. Pour que les sous-questions (d) et (e) aient un sens, on fait l'hypothèse que la chaîne est composée de caractères qui sont tous distincts; par exemple, `abcdef` et non `abcabc`. (Exercice 3.5 dans la 1ère édition mais avec les items (c) et (d) inversés.)
- **Exercices 3.3.5** (a), (b) et (c). Pour plus de difficulté, faire (d), (e), (f), (h) et (i). (Exercice 3.7 dans la 1ère édition.)

  ***Note**: au numéro (a), l'énoncé peut être précisé ainsi:*

  - *Il s'agit des 5 voyelles anglaises (a, e, i, o, u);*
  - *les mots de ce langage contiennent exactement une occurrence de chacune des 5 voyelles;*
  - *les 5 voyelles apparaissent en ordre mais ne sont pas forcément consécutives.*

  ***Note**: aux numéros (d) et (e), ne pas tenter de créer une définition pour tous les chiffres, se limiter à $\{0,1,2\}$ est amplement suffisant.*

  ***Note**: au numéro (d), il s'agit de répétitions **successives** d'un chiffre; autrement dit* 0122 *n'est pas accepté, mais* 012012 *l'est, i.e. chaque chiffre peut apparaitre plus d'une fois dans la chaine mais on ne retrouvera jamais deux chiffres identiques côte à côte).*

  ***Note**: au numéro (e), nous reprenons la même définition de "répétition" qu'en (d), à l'exception qu'un mot du langage peut contenir **une** répétition (tous chiffres confondus). Par exemple,* 012, 0221, 2110 *sont acceptés, mais* 1122 *et* 01001200 *ne le sont pas.*

- **Exercice 3.3.9**. (Exercice 3.10 (c) dans la 1ère édition.)

- **Exercice supplémentaire 1**. Soit la définition régulière suivante:

$$
\begin{array}{rcl}
\mathit{letter} & \rightarrow & A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \\
\mathit{digit} & \rightarrow & 0 \mid \ldots \mid 9 \\
\mathit{whitespace} & \rightarrow & (\;\sqcup \mid \backslash\mathrm{t} \mid \backslash\mathrm{n}\;)^{+} \\
\mathit{id} & \rightarrow & \mathit{letter}\ (\mathit{letter} \mid \mathit{digit} \mid \_)^{*} \\
\mathit{num} & \rightarrow & (-)^{?}\ \mathit{digit}^{+}\ (\ .\ \mathit{digit}^{+}\ )^{?}
\end{array}
$$

Note: \t représente une tabulation, et \n, un changement de ligne.

Informellement, *comment* est défini comme une expression commençant par `/*` et finissant par `*/` (pouvant contenir des `*` ou des `/` mais pas consécutivement).

Construisez une série d'automates finis déterministes permettant de procéder à l'analyse lexicale du langage suivant:

Expression régulière	**Jeton**	**Attribut**
whitespace	–	–
comment	–	–
id	**id**	(pointeur vers la table des symboles)
num	**num**	(valeur de la constante)

Réponses

3.1.1

Notons qu'il existe plusieurs choix valides; par exemple, dans la solution ci-dessous, tous les opérateurs sont regroupés sous le jeton **op** et le type précis de l'opérateur est en attribut. Créer un type de jeton pour chaque type d'opérateur aurait aussi été correct.

Jeton	Attribut associé
float	
id	limitedSquare
(	
id	x
)	
float	
id	x
{	
return	
(	
id	x
comp	$\leq$
num	-10.0
op	$\|$
id	x
comp	$\geq$
num	10.0
)	
op	?
num	100
op	:
id	x
op	*
id	x
;	
}	

3.3.2

- (a) Le langage qui contient des `a` et des `b` et où les mots commencent et finissent par `a`, de longueur au moins 2.
- (b) le langage $\{\mathtt{a}, \mathtt{b}\}^*$, i.e. le langage où les mots contiennent des `a` et des `b`.
- (c) le langage $\Sigma^* \cdot \{\mathtt{a}\} \cdot \Sigma^2$ où $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, i.e. le langage contenant des mots d'au moins 3 lettres, qui contiennent des `a` et des `b`, et où la troisième lettre en partant de la fin du mot est un `a`.

- (d) Le langage qui contient des `a` et des `b` et où les mots contiennent exactement 3 `b`.
- (e) le langage

$$\{w \in \{\texttt{a}, \texttt{b}\}^* \mid \text{il y a un nombre pair de } \texttt{a} \text{ et un nombre pair de } \texttt{b} \text{ dans } w\}.$$

3.3.3

(a) $n + 1$

(b) $n + 1$

(c) $n - 1$

(d) $1 + n + (n - 1) + \ldots + 2 + 1 = 1 + n(n + 1)/2$. En effet, il y a:
1 sous-chaine de longueur 0,
n sous-chaines de longueur 1,
$(n - 1)$ sous-chaines de longueur 2,
$\ldots$,
2 sous-chaines de longueur $(n - 1)$ et
1 sous-chaine de longueur n.

(e) 2^n. En effet, pour former une sous-séquence, on doit décider, pour chaque caractère de la séquence originale, si on l'inclut ou pas dans la sous-séquence. Il y a 2^n façons de décider et, donc, 2^n sous-séquences différentes.

3.3.5

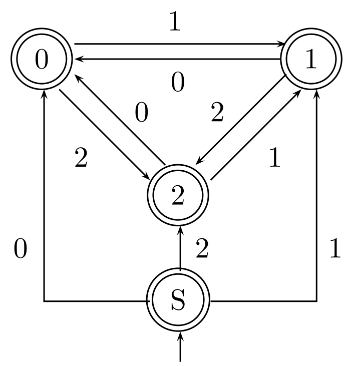
- (a) C représente une ou plusieurs consonnes (optionnelles)

$$\begin{array}{rcl} C & \rightarrow & [\text{bcdfghjklmnpqrstvwxyz}] * \\ S & \rightarrow & C\,\texttt{a}\,C\,\texttt{e}\,C\,\texttt{i}\,C\,\texttt{o}\,C\,\texttt{u}\,C \end{array}$$

- (b) $S \rightarrow a^*b^*c^*d^*e^*f^*g^*h^*i^*j^*k^*l^*m^*n^*o^*p^*q^*r^*s^*t^*u^*v^*w^*x^*y^*z^*$
- (c) Q représente le contenu de guillemets ("quotes"); C représente le contenu du commentaire. Le contenu du commentaire est tout caractère sauf `*` suivi de `/`, ou des guillemets qui eux peuvent contenir tout caractère sauf des guillemets.

$$\begin{array}{rcl} Q & \rightarrow & \texttt{"}[\hat{}\,\texttt{"}]^*\texttt{"} \\ C & \rightarrow & \texttt{*}^*([\hat{}\,\texttt{*/"}] \mid Q) \mid \texttt{/} \\ S & \rightarrow & \texttt{/*}\,C^*\,\texttt{*}^+\texttt{/} \end{array}$$

- (d) Puisqu'il est plus facile d'imaginer un automate fini qu'une définition régulière pour ce problème, construisons l'automate fini puis utilisons l'algorithme de conversion d'un automate en définition régulière.



En appliquant l'algorithme, premièrement, en ajoutant un état acceptant unique et, deuxièmement, en éliminant les états 2, 1 et 0, dans l'ordre, nous obtenons la définition régulière suivante:

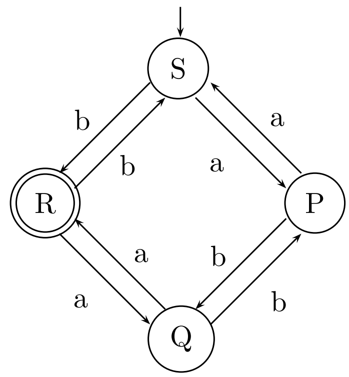
$$
\begin{array}{rcl}
d_0 & \rightarrow & \texttt{20} \\
d_1 & \rightarrow & \texttt{21} \\
e_0 & \rightarrow & \texttt{0} \mid d_0 \\
e_1 & \rightarrow & \texttt{1} \mid d_1 \\
e_\epsilon & \rightarrow & \texttt{2} \mid \epsilon \\
f_0 & \rightarrow & e_1 \;\; {d_1}^* \;\; e_0 \\
f_\epsilon & \rightarrow & e_1 \;\; {d_1}^* \;\; e_\epsilon \\
g & \rightarrow & (e_\epsilon \mid f_\epsilon) \mid (e_0 \mid f_0)(d_0 \mid f_0)^*(e_\epsilon \mid f_\epsilon)
\end{array}
$$

- (e) Nous pourrions construire un automate fini, comme au numéro (d), puis le convertir en expression régulière. Toutefois, notons que nous pouvons simplement réutiliser la définition régulière "g" définie en (d) ainsi:

$$S \rightarrow gg$$

En effet, en concaténant deux fois "g", on permet au plus une répétition, à la jonction des deux "g".

- (f) Puisqu'il n'est pas évident de construire une telle définition régulière, commençons par construire un automate fini.



En appliquant l'algorithme, premièrement, en ajoutant un nouvel état de départ et un nouvel état acceptant et, deuxièmement, en éliminant les états P, R, Q et S, dans l'ordre, nous obtenons la définition régulière suivante:

$$\begin{array}{lcl} d & \rightarrow & \texttt{aa} \mid \texttt{bb} \\ e & \rightarrow & \texttt{ab} \mid \texttt{ba} \\ f & \rightarrow & (d \mid e\, d^*\, e)^*(\texttt{b} \mid e\, d^*\, \texttt{a}) \end{array}$$

- (h) Cette définition régulière s'assure que toute sous-chaine `ab` est soit immédiatement suivie d'un `a`, soit à la fin du mot. Ainsi on évite toute sous-chaine `abb`.

$$S \quad \rightarrow \quad \texttt{b}^*\, (\texttt{a}\, \texttt{b}^?)^*$$

- (i) On permet un nombre arbitraire de `b` au début car ceux-ci ne peuvent servir à créer la sous-séquence `abb`. Après le début, dès qu'on trouve un `b` suivant un `a`, on ne permet plus que d'ajouter des `a`.

$$S \quad \rightarrow \quad b^*\ a^*\ (ab\ a^* \ \mid\ \epsilon)$$

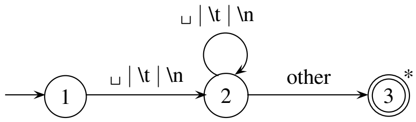
3.3.9

Soit $r\{m, n\}$ où r est un patron et $0 \leq m \leq n$. Alors on a:

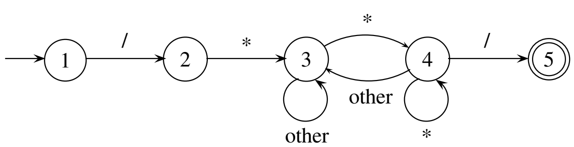
$$\underbrace{r\ r\ \ldots\ r}_{m\ fois}\ \underbrace{r^?\ r^?\ \ldots\ r^?}_{n-m\ fois}$$

Cette expression régulière est équivalente, donc la notation $r\{m, n\}$ n'apporte aucune puissance supplémentaire.

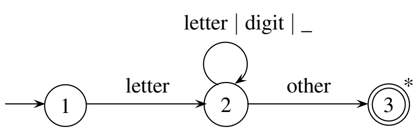
Exercice supplémentaire 1



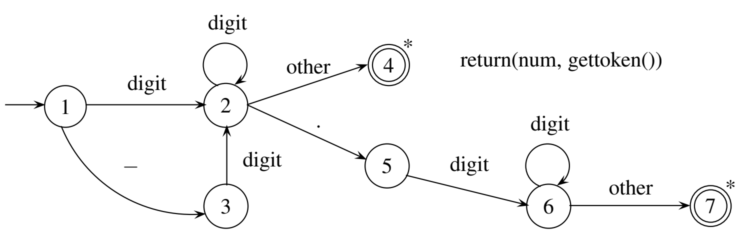
drop & restart



drop & restart



return(id, install_id(gettoken()))



return(num, gettoken())

return(num, gettoken())