ESIEE Engineering
IN3R21 Systèmes d'exploitation

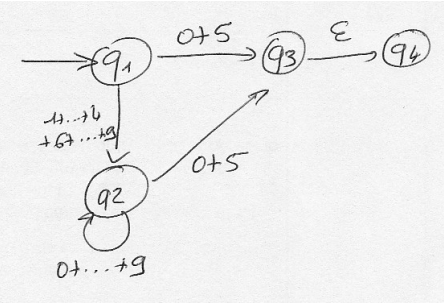
Correction du TD n°3
Expressions régulières

Exercice 1

a. `[A-Z][a-z]\{1,\} [A-Z][a-z]\{1,\}`
b. `[A-Z][a-z]\{1,\}\(- [A-Z][a-z]\{1,\}\)\{0,1\}`
c. `[A-Z][a-z]\{1,\}\(- [A-Z][a-z]\{1,\}\)*`
d. `^[^A]`
e. Impossible dans les deux cas : choix multiple indispensable. L'automate comporte deux sous-automates différents de plus d'un état chacun -> impossible avec une ERB
f. `[1-9][0-9]\{,3\} *[A-Za-z]\{2,3\} *[0-9]\{2,2\}`

Exercice 2

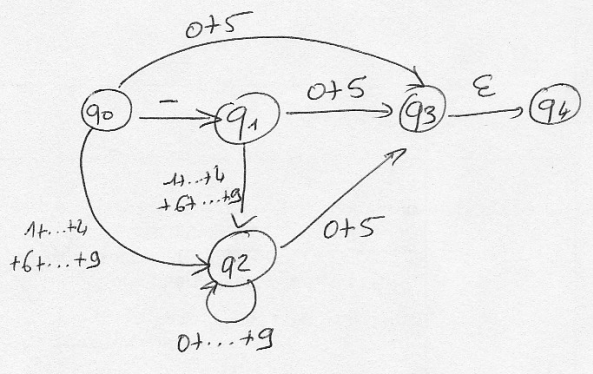
a.



Automate non déterministe : ambiguïté de transition q2->q2 ou q2->q3 pour les symboles 0 et 5.
Rappels : x+y signifie « x ou y », x+…+y « tous les symboles entre x et y », epsilon représente le mot vide. Reconnaître un mot C exactement revient à reconnaître C epsilon (on reconnaît aussi la fin de mot). q1 est l'état initial, q4 l'état final.

Regex étendue : `[05]|([1-9][0-9]*[0-5])`
Regex de base : impossible, car on doit distinguer le cas 0 séparément -> choix multiple

b. Automate non déterministe pour les mêmes raisons qu'au a). Le '-' optionnel entraîne un nouvel état q0 qui devient l'état initial, et la répétition de toutes les transitions dont q1 est à l'origine. Le reste est inchangé.

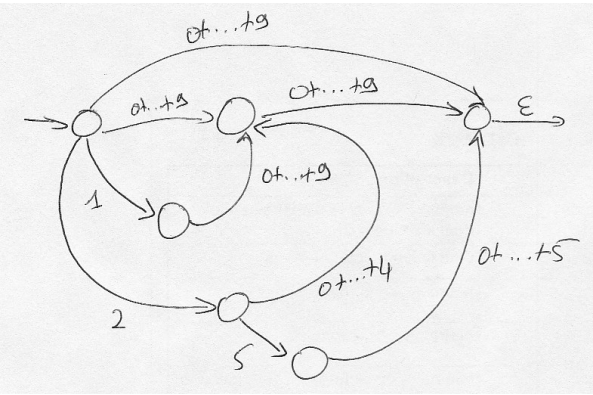


Regex étendue : `-?[05]|([1-9][0-9]*[0-5])`
Regex de base : impossible, pour les mêmes raisons qu'à la question précédente.

c. On peut donner les deux (déterministe et non déterministe). Version non déterministe : on raisonne en considérant 4 grandes branches :

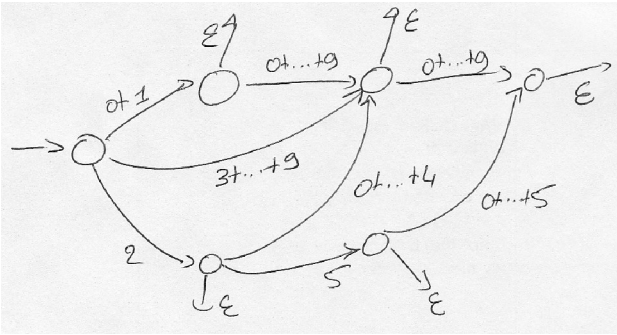
- les nombres à 1 seul chiffre (entre 0 et 9)
- les nombres à 2 chiffres entre (0 et 9)
- les nombres à 3 chiffres dont le 1er est 0 ou 1
- les nombres à 3 chiffres dont le 1er est 2

Tout le reste est interdit.



Version déterministe : on doit considérer des choix exclusifs à chaque état, ce qui amène à distinguer :

- les nombres qui commencent par 0 ou 1 : ils ne posent aucun problème, et peuvent être suivis de aucun, 1 ou 2 chiffres entre 0 et 9 (on tolère 0x et 00x).
- les nombres qui commencent par 2 :
  - si le chiffre suivant est dans la plage 0..4 alors il peut être suivi d'un chiffre entre 0..9 ou non
  - s'il vaut 5, alors le chiffre suivant s'il existe doit être dans la plage 0..5
- les nombres qui commente par 3..9 : ils ont nécessairement 2 chiffres au plus.



Regex étendue à partir de l'automate déterministe : `^(([01][0-9]?)|[3-9]|(2[0-4]?))[0-9]?|25[0-5])$`
Regex de base : impossible

d. Il s'agit simplement de reconnaître M.M.M.M où M est un sous-automate qui répond à la question précédente.

Regex étendue : `^(M\.){3,3}M$`
Regex de base : impossible car M lui-même n'en admet pas.

Exercice 3

Oui : il faut rendre déterministe l'extraction du premier et du dernier champ dans sed en utilisant ^ et $, et [^ :] pour « tout sauf : » : cela définit les chaînes extraites 1 et 3. Les champs intermédiaires étant alors vus implicitement comme une seule chaîne, remplacée sans modification.

```
sed "s/\^([^:]*\):\(.*\):\([^:]*\)$/\3:\2:\1/"
```

Remarque : la solution ne vaut que s'il y a au moins 3 champs.

Exercice 4

1. On veut que la sortie soit formatée de telle sorte que :
- les informations d'une interface soient toutes concentrées sur 1 ligne
- les lignes sont séparées entre elles par des retours chariots.

Donc les lignes vides produites par ifconfig doivent être remplacées par des retours chariot, et les retours chariots par des espaces. L'utilitaire sed permet de réaliser le premier remplacement. En revanche, seule la backquote peut réaliser le deuxième, mais :

```
echo `/sbin/ifconfig –a`
```

supprime aussi les lignes vides et ramène toute la sortie sur une seule ligne. On doit donc :

a. commencer par marquer les lignes vides avant d'appliquer la backquote, par un mot qui n'a pas de raison d'apparaître dans la sortie de ifconfig (exemple : toto)
   `/sbin/ifconfig | sed "s/^$/toto/"`

b. « backquoter » ce premier traitement pour tout ramener sur une ligne
   `echo `/sbin/ifconfig | sed "s/^$/toto/"``
c. remplacer les occurrences de « toto » par des retours chariots avec sed :
   `echo `/sbin/ifconfig | sed "s/^$/toto/" ` | sed "s/toto /\n/g"`

On pourra vérifier que cette dernière ligne répond bien au problème posé.

2. Le nom d'interface est toujours le premier mot qui apparaît. L'adresse internet est toujours précédée de « inet adr » et les paquets reçus et émis par « RX packets » et « TX packets » respectivement. De plus, tous les mots sont forcément délimités par des caractères d'espacement à cause de la backquote. Il suffit d'utiliser sed en s' « appuyant » sur ces chaînes et sur '^' pour l'interface, les expressions à extraire étant « tout sauf espace ».

```
sed "s/^\([^ ]*\).*inet adr:\([^ ]*\).*RX packets:\([^ ]*\).*TX packets:\([^
]*\).*$/\1|\2|\3|\4/"
```

3. Les commandes précédentes donnent le formatage attendu. Il faut simplement relire ligne à ligne pour calculer les sommes, ce qui peut se faire ainsi :

```
while [ 1 ]; do
clear
echo "Interface | IP  | Entrant | Sortant  | Total"
echo "-------------------------------------------"
echo `/sbin/ifconfig -a | sed "s/^$/toto/"` | sed "s/toto /\n/" | sed "s/\([^ ]*\
) .*adr:\([^ ]*\).*RX packets:\([^ ]*\).*TX packets:\([^ ]*\).*$/\1 | \2 | \3 | \4/"
| {
  ttx=0
  trx=0
  while read line; do
        tx=`echo "$line" | cut -f3 -d\|`
        ty=`echo "$line" | cut -f4 -d\|`
        ttx=$((ttx+tx))
        tty=$((tty+ty))
        echo "$line"
  done
  echo "total | - | $ttx | $tty"
}
sleep 1
done
```