3R-IN3 – Systèmes d'exploitation
TD 3

Xavier Hilaire
x.hilaire@esiee.fr

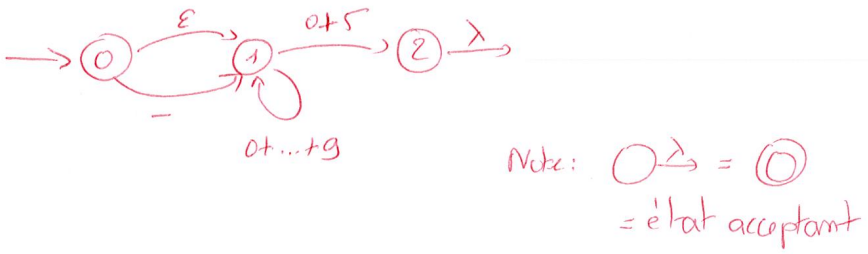
28 mars 2025

1 AEF et expressions régulières

Donner les automates à états finis, et les expressions régulières correspondantes (étendues, et de base lorsque c'est possible) qui reconnaissent les chaînes de caractères suivantes :

1. Un entier multiple de 5

   Sol.: Un tel entier peut être signé, et doit obligatoirement se terminer par 0 ou 5. Il peut contenir en plus un nombre arbitraire de chiffres devant le 0 ou 5 terminal.



- Expression régulière de base : `^-\{0,1\}[0-9]*[05]$`. Validation :

```
$ expr -5 : '^-\{0,1\}[0-9]*[05]$'
2
$ echo $?
0
$ expr 195 : '^-\{0,1\}[0-9]*[05]$'
3
$ echo $?
0
$ expr -22 : '^-\{0,1\}[0-9]*[05]$'
0
$ echo $?
1
$
```

- Expression régulière étendue : `^-?[0-9]*[05]$`. Validation :

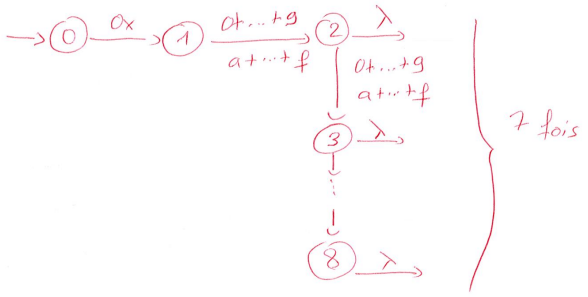
```
$ [[ -745 =~ ^-?[0-9]*[05]$ ]]
$ echo $BASH_REMATCH
-745
$ [[ -745 =~ ^-?[0-9]*[05]$ ]]
$ echo $?
0
$ [[ 92 =~ ^-?[0-9]*[05]$ ]]
```

```
$ echo $BASH_REMATCH

$ [[ 92 =~ ^-?[0-9]*[05]$ ]]
$ echo $?
1
$
```

2. Un entier positif de 32 bits, codé en base hexadécimale, comme il serait écrit en langage C

   Sol.: Un tel entier doit débuter par `0x` (notation du C), et ne doit normalement pas comporter plus de 8 chiffres ou lettres de a à f.

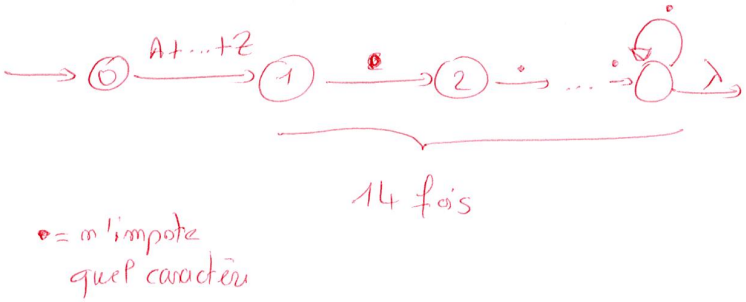


   Expression régulière de base : `^0x[0-9a-f]\{1,8\}$`

   Expression régulière étendue : `^0x[0-9a-f]{1,8}$`

   Si l'on autorise à déborder les 8 chiffres, alors les chiffres supplémentaires doivent être des 0 en préfixe. Il suffit alors :
   — de rajouter à l'automate une boucle de l'état 1 vers lui-même, la transition étant conditionnée par le caractère 0
   — de remplacer `0x` par `0x0*` dans les deux expressions régulières précédentes

3. Les chaînes débutant par une majuscule, et longues d'au moins 15 caractères.



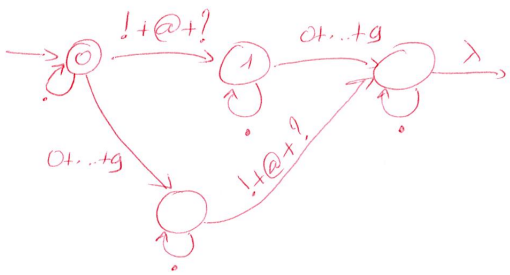
   Sol.:

   Expression régulière de base : `^[A-Z].\{14\}.*$`

   Expression régulière étendue : `^[A-Z].{14}.*$`

4. Un mot de passe : il doit contenir au moins un caractère spécial parmi '!', '+', '@', '?' et '-', et au moins un chiffre. Peut-on assurer en plus qu'il soit long d'au moins 15 caractères ?

   Sol.: La complication ici est que l'on sait qu'on doit croiser un chiffre et un caractère spécial au moins une fois, mais on ne sait pas dans quel ordre. On considère donc les deux cas de figure dans l'automate ci-dessous :



Expression régulière étendue : `^.*(([!+@?-].*[0-9])|([0-9].*[!+@?-])).*$`

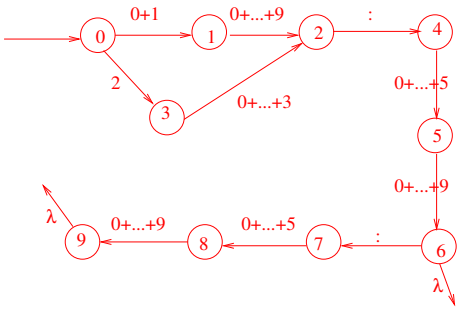
Expression régulière de base : inexistante

Il est théoriquement possible d'assurer qu'il contienne au moins 15 caractères, mais pour cela il faudrait énumérer TOUTES les positions possibles du caractère spécial et de la majuscule, donc TOUTES les expressions régulières qui en résultent, car de la différence de positions entre ces deux caractères dépend le nombre de caractères minimal qui formera la fin de chaîne.

Or, il y a 2*(14*13/2) = 182 expressions possibles. Donc en toute rigueur, oui, c'est possible ; mais très peu raisonable. Mieux vaut tester la longueur indépendamment de la validité de la chaîne, ce qui peut d'ailleurs se faire en un seul temps avec `expr`, puisqu'elle écrit la longueur de chaîne qui a été extraite sur sa sortie standard en plus de statuer sur la validité de la chaîne candidate dans son code de sortie.

5. Une heure, dans le format `hh:mm[:ss]` où les secondes sont falcultatives.

   Sol.: Les heures hh sont comprises entre 00 et 23 inclus, ce qui donne deux branches : 0 ou 1 suivi de n'impote quel chiffre dans la plage 0-9 ; et 2, suivi de n'importe quel chiffre dans la plage 0-3 seulement. Les minutes et les secondes ont le même format, et sont limtées à 59, donc produisent le même automate. Sortie possible à l'état 6 si les secondes ne sont pas précisées, sinon à l'état 9.



   Expression régulière de base : inexistante

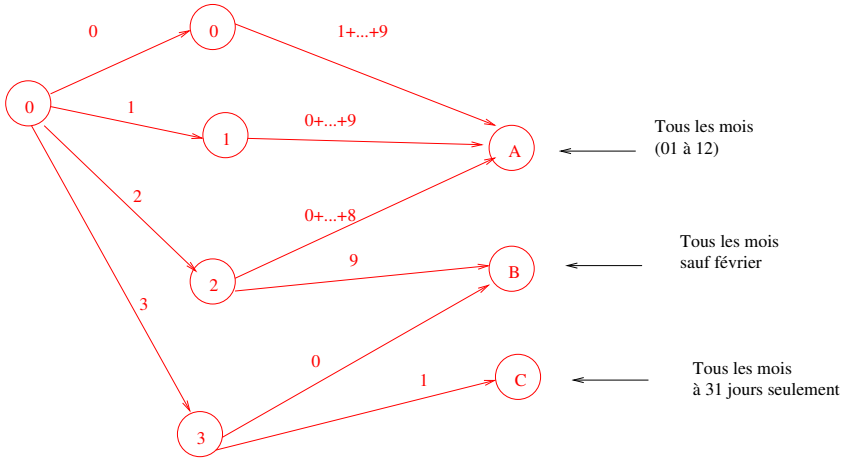
   Expression régulière étendue : `^([01][0-9]|2[0-3])(:[0-5][0-9]){1,2}$`

6. Une date, dans le format `jj/mm/aaaa`, en admettant qu'il n'y a pas d'années bisextiles.

   Sol.: La partie la plus difficile est la spécification des jours en accord avec les mois, car les deux fonctionnent ensemble. L'absence d'années bisextiles implique que les mois de février ont tous 28 jours. Il existe donc 3 branches pour les jours, conditionnées par le mois qui suit :
   - Si le jour est au plus de 28, alors il peut être suivi par n'importe quel mois entre 01 et 12 : c'est le cas A
   - Si le jour est 29 ou 30, alors le mois qui suit peut être n'importe lequel sauf février (02) : c'est le cas B

— Et si le jour est 31, alors seul un mois à 31 jours peut suivre (01,03,05,07,08,10, et 12) : c'est le cas C

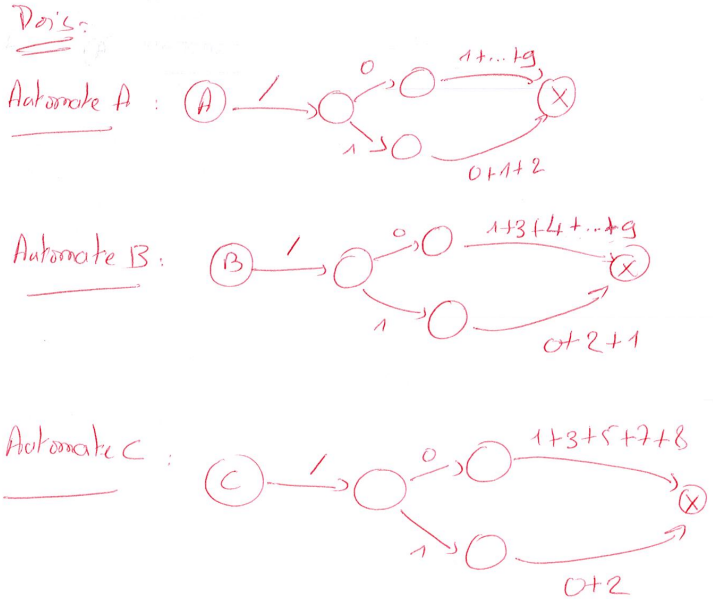
Ces trois cas sont résumés dans la partie ci-dessous. Il reste à écrire les automates qui leur correspondent.



Les expressions régulières étendues correspondantes sont

```
0[1-9]|1[0-9]|2[0-8] + regex A
29|30 + regex B
31 + regex C
```

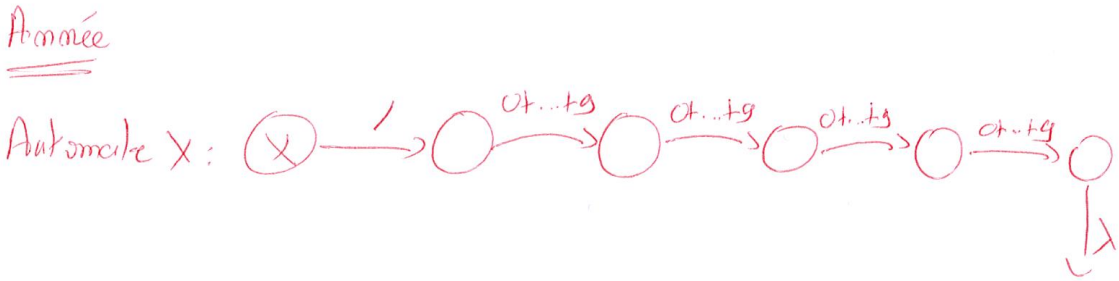
Pour les mois, on obtient les automates suivants :



Les expressions régulières étendues respectives sont :

```
/(0[1-9]|1[0-2])
/(0[13-9]|1[0-2])
/(0[13578]|1[02])
```

Quant à l'année (automate X), elle est indépendante des états précédents, et ne présente aucune difficulté, il s'agit simplement de 4 chiffres quelconques enchaînés :



L'expression régulières étendue correspondante est simplement

```
/[0-9]{4}
```

En combinant les expressions précédentes en une seule par des choix multiples, et en factorisant l'année (dernière ligne ci-dessous), on obtient finalement l'expression étendue suivante :

```
^(
((0[1-9]|1[0-9]|2[0-8]|2[0-8])/(0[1-9]|1[0-2]))
|
((29|30)/(0[13-9]|1[0-2]))
|
(31/(0[13578]|1[02]))
)
/[0-9]{4}$
```

normalement écrite sur une seule ligne. Il n'y a pas d'expression de base correspondante.

Vérifier la validité de votre expression régulière de base avec l'utilitaire `expr` pour les regex de base, et avec la commande `[[ =~ ]]` du Bash pour les regex étendues.

2 Utilitaires grep et sed

Au TD1, vous avez produit un script qui utilisait la sortie de `ifconfig` pour calculer et afficher la somme de tous les paquets réseau (entrants comme sortants) échangés au travers de toutes les interfaces de votre machine.

Nous allons reprendre le code écrit et le modifier pour qu'il fasse la même chose, mais en n'utilisant plus que grep, sed, et echo.

1. En utilisant `ifconfig` et `grep`, faites en sorte de n'afficher que les lignes relatives aux RX packets ou aux TX packets

   Sol.:

```
$ ifconfig | grep '[RT]X packets'
        RX packets 27694215  bytes 11088683441 (10.3 GiB)
        TX packets 19183430  bytes 23994772100 (22.3 GiB)
        RX packets 1439945  bytes 22086687930 (20.5 GiB)
        TX packets 1439945  bytes 22086687930 (20.5 GiB)
        RX packets 266554  bytes 15504344 (14.7 MiB)
        TX packets 430060  bytes 721006094 (687.6 MiB)
$
```

2. Faites en sorte d'afficher toutes les autres

   Sol.:

```
$ ifconfig | grep -v '[RT]X packets'
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

```
        inet 147.215.50.28  netmask 255.255.255.0  broadcast 147.215.50.255
        inet6 fe80::dd76:6d12:4141:64d5  prefixlen 64  scopeid 0x20<link>
        ether c0:18:03:c0:8c:db  txqueuelen 1000  (Ethernet)
        RX errors 0  dropped 28256  overruns 0  frame 0
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xe1200000-e1220000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:aa:56:07  txqueuelen 1000  (Ethernet)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

$
```

3. En utilisant `sed` en plus des commandes précédentes, supprimer ce qui n'est pas intéressant dans chaque ligne

   Sol.:

```
$ ifconfig | grep '[RT]X packets' | sed -E "s/^.*[RT]X packets ([0-9]+).*$/\1/"
27694882
19183506
1439970
1439970
266554
430060
$
```

   Remarque : il est également possible d'obtenir le même résultat en appelant deux fois sed pour éliminer ce qu'on ne veut pas :

```
ifconfig | grep '[RT]X packets' | sed 's/^ *[RT]X packets //' | sed 's/ bytes.*$//'
```

   Il revient aussi au même de ne faire qu'un appel à sed, mais utiliser l'option -e deux fois :

```
ifconfig | grep '[RT]X packets' | sed -e 's/^ *[RT]X packets //' -e 's/ bytes.*$//'
```

   Mais la première forme, qui utilise `\1` reste la plus appropriée au problème – on veut éliminer certaines parties de chaque ligne, et en conserver une autre.

4. Modifier encore l'appel à `sed` pour que le shell puisse calculer la somme finale, et l'afficher avec un `echo` (vous ne devez plus utiliser `tr`)

   Sol.:

```
$ a=$(ifconfig | grep '[RT]X packets' | sed -E "s/^.*[RT]X packets ([0-9]+).*$/\1+/";
echo 0)
$ echo $a
27691917+ 19183002+ 1439879+ 1439879+ 266554+ 430060+ 0
$ echo $((a))
50452235
$
```

3 Validation d'adresse IPv4

Ecrivez une fonction Bash qui termine avec le code d'erreur 0 si et seulement si son unique argument est une adresse IPv4 valide. Votre fonction ne devra utiliser aucune commande externe.