TP 1
Expressions rationnelles

Version du 26 septembre 2016

Objectif

L'objectif de ce TP est d'écrire des expressions rationnelles et de les appliquer sur des fichiers afin d'en extraire du texte et de le manipuler. Nous utiliserons l'outil Perl [1], installé sur tout bon système d'exploitation.

Rappels commandes Unix

Voilà un petit rappel des commandes et outils incontournables :

`mkdir` *`dir`*	crée un répertoire nommé *`dir`* (MaKe DIRectory)
`ls`	affiche le contenu du répertoire courant (LiSt)
`cd` *`dir`*	va dans le répertoire pointé par *`dir`* (Change Directory)
`cd ..`	retourne dans le répertoire parent
`cd`	retourne dans votre *HOME*
`cp` *`src dest`*	copie un fichier (CoPy)
`cp -R` *`src dest`*	copie un répertoire (CoPy)
`mv` *`src dest`*	déplace/renomme un fichier ou un répertoire (MoVe)
`cat` *`file`*	sort le contenu de *`file`* sur la sortie standard

À tout moment, vous pouvez invoquer le manuel en utilisant la commande `man` pour MANual. Par exemple, pour savoir comment utiliser la commande `mkdir`, faites : '`man mkdir`'.

Essayez d'organiser proprement votre espace de stockage pour vous y retrouver au moment de réviser. Créez par exemple un répertoire `tps` ('`mkdir tps`'), allez y ('`cd tps`'), et ajoutez un répertoire `thlr` ('`mkdir thlr`'). Vous pourrez organiser ce répertoire par TP et par exercice. **Pensez à y copier à chaque fois la solution des questions pour pouvoir reprendre le TP plus tard !** Pour cela vous pouvez copier-coller vos expressions vers un fichier texte. (Astuce : la commande `history` affiche la liste des dernières commandes tapées : vous pouvez facilement y trouver les commandes que vous souhaitez recopier.)

Vous pourrez ajouter plus tard d'autres répertoires dans le répertoire `tps` pour les TPs d'autres matières.

Utiliser les expressions rationnelles en Perl

La syntaxe de Perl pour les expressions rationnelles diffère de celle que nous utilisons en TD. Dans le tableau qui suit $c_1, c_2, \ldots, c_n$ désignent des lettres $l \in \Sigma$, et $L(e)$ est le langage dénoté par l'expression rationnelle e.

1. http://www.perl.org. `man perlretut` affiche un tutoriel sur l'usage des expressions rationnelles dans Perl. Pour sortir du manuel, taper '`q`'.

langage	expression rationnelle Perl
Σ	`.`
$\{c\}$	`c`
$\{c_1, c_2\}$	`[`c_1c_2`]`
$\{c_1, c_2, \ldots, c_n\}$	`[`c_1`-`c_n`]`
$\Sigma \setminus \{c_1, c_2\}$	`[^`c_1c_2`]`
$L(e)L(f)$	ef
$L(e) \cup L(f)$	$(e \mid f)$
$L(e) \cup \{\varepsilon\}$	e`?`
$L(e)^\star$	e`*`
$L(e)^+$	e`+`

Notez qu'il est possible d'assembler des classes de caractères entre elles. Par exemple, `[a-zA-Z]` représente une lettre minuscule ou majuscule. Enfin, pensez à échapper les caractères spéciaux par un '\'. Par exemple, si vous avez besoin de reconnaître le caractère '`.`' (un point), il faut écrire '`\.`'. Ces caractères spéciaux sont `\ . ? * [ ] | ( )`.

Pour tester vos expressions rationnelles avec Perl, entrez une commande de la forme suivante dans un terminal :

`perl -0777 -pe 's/`*expression-rationnelle*`/`*texte-à-substituer*`/gsm' `*fichier*

texte-à-substituer permet de spécifier par quoi remplacer le texte reconnu. Par exemple, si l'on applique '`perl -0777 -pe 's/[a-z]/@/gsm'`' sur un fichier contenant '`azertyAZERTYuiop`', on obtient comme affichage '`@@@@@@AZERTY@@@@`' : chaque lettre minuscule reconnue a été remplacé par '`@`'. Le fichier original n'a pas été modifié par cette opération : le résultat de la substitution est uniquement envoyé à l'écran.

Fichiers utiles pour réaliser le TP

Pour chaque question nous vous fournissons des fichiers sur lesquels vous appliquerez votre expression rationnelle. Ces fichiers sont contenus dans les répertoires '`TP1_files/exo1`' à '`TP1_files/exo4`' et ont comme nom le numéro de la question associée, précédé par '`q`'. Ainsi vous devrez tester votre réponse à la question 2 de l'exercice 1 sur le fichier '`TP1_files/exo2/q2`'. Cependant rien ne vous empêche d'effectuer des tests sur vos propres textes ! Au sein d'un même exercice, pensez à vérifier que votre expression rationnelle fonctionne avec les questions précédentes.

Pour recopier tous ces fichiers dans votre répertoire personnel, entrez dans un terminal :

```
wget http://www.lrde.epita.fr/~akim/thlr/TP1_files.tar.bz2
tar -xjvf TP1_files.tar.bz2
```

Exercice 1 – Commentaires du langage Pascal

On cherche à reconnaître les commentaires en Pascal qui sont de la forme {*texte*}, où *texte* peut être composé de n'importe quelle suite de caractères.

1. Dans un premier temps, on suppose que l'on a un seul commentaire par fichier. Utilisez Perl pour remplacer les commentaires reconnus par le texte `COMMENTAIRE`.
2. Si maintenant, on a plusieurs commentaires dans un fichier, que se passe-t-il ? Écrivez une nouvelle expression permettant de pallier ce problème, toujours en remplaçant les commentaires reconnus par `COMMENTAIRE`.

Exercice 2 – Chaînes de caractères du langage C

On cherche cette fois à trouver les chaînes de caractères du langage C. Celles-ci sont comprises entre guillemets et contiennent un nombre indéfini de caractères : "*texte*". *texte* est encore une suite de n'importe quel caractère.

1. Écrivez une expression rationnelle permettant de reconnaître de telles chaînes (il peut y en avoir plusieurs dans le même fichier) : elle devra remplacer les motifs reconnus par le texte `CHAINE`.
2. Il est possible dans ces chaînes « d'échapper » (ou « déspécialiser ») les caractères spéciaux en les faisant précéder du caractère \. Par exemple, si on veut pouvoir afficher le caractère ", on doit écrire `"Texte \" suite du texte"` (sinon il aurait été reconnu comme le guillemet de fin de chaîne). En fait, tous les caractères peuvent être échappés de cette manière. Ajoutez cette possibilité à votre précédente expression rationnelle.

Exercice 3 – Modifications d'identifiants en série

Il est possible de faire référence aux motifs reconnus en utilisant `$1`, `$2`, etc. Ces variables correspondent aux groupes de parenthèses placées dans une expression rationnelle. `$n` référence la n^{e} paire de parenthèses. Ainsi, `perl -0777 -pe 's/([a-z])/$1@/gsm'` appliqué sur le texte `'azertyAZERTY'` produit `'a@z@e@r@t@y@AZERTY'`. On a placé `'@'` derrière chaque lettre minuscule reconnue.

Ce rappel du motif attrapé au sein du texte de substitution est très utile lorsqu'on a besoin d'appliquer des modifications en série sur du texte. Par exemple, on peut vouloir passer en minuscule la première lettre d'un identifiant.

1. On veut ici modifier les identifiants de fonctions qui ont été écrites sous la forme `get_`*`Identifiant`*`()` ou `set_`*`Identifiant`*`(arguments)`, où *`Identifiant`* est une suite quelconque de caractères alphanumériques. Le but ici est de supprimer le caractère '_' au sein de ces identifiants. On repère qu'il s'agit d'une fonction puisque le nom est suivi d'un texte entre parenthèses et qu'elle est précédée par un espace. Le nom de la fonction est uniquement composé de caractères alphanumériques, ainsi que les arguments.
2. On veut maintenant modifier tous les appels de fonctions commençant par une majuscule, en remplaçant cette majuscule par la lettre minuscule correspondante. Par exemple, `Apply()` se trouve modifié en `apply()` et `Process(int i)` devient `process(int i)`.

   Les caractères sont toujours des caractères alphanumériques.

   Pour modifier la casse d'un caractère, il faut mettre `\l` devant ce caractère [2], dans le motif de substitution.

Exercice 4 – Commentaires imbriqués

1. On souhaite reconnaître les chaînes de l'exercice 1, mais cette fois-ci, elles doivent pouvoir être imbriquées : `{Texte {Texte imbriqué}}`. Dans cas, on veut donc voir affiché `@COMMENTAIRE@COMMENTAIRE@@`

2. 'man perlre'.