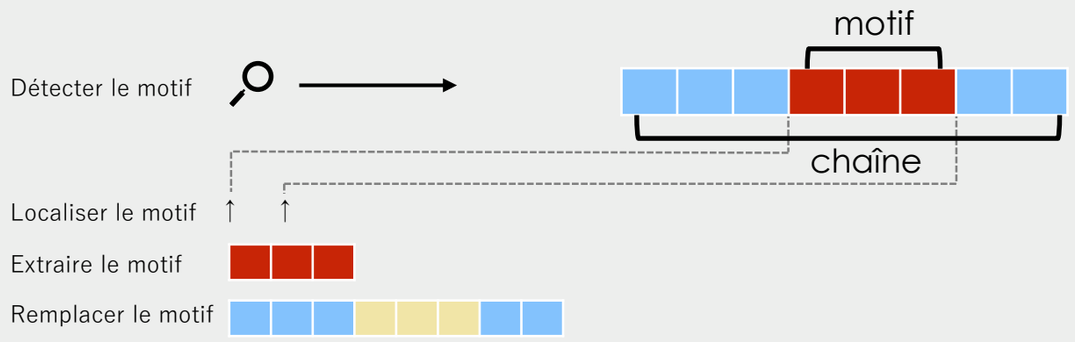
Expressions régulières en R

Aide-mémoire

Classes de Caractères

[[:digit:]] ou \\d	Chiffres décimaux; `[0-9]`
\\D	Tout sauf chiffres décimaux; `[^0-9]`
[[:lower:]]	Lettres miniscules; `[a-z]`
[[:upper:]]	Lettres majuscules; `[A-Z]`
[[:alpha:]]	Caractères alphabétiques; `[A-z]`
[[:alnum:]]	Caractères alphanumériques; `[A-z0-9]`
\\w	Caractères alphanumériques ou tirets; `[A-z0-9_]`
\\W	Tout sauf caractères alphanumériques ou tirets
[[:xdigit:]] ou \\x	Chiffres hexadécimaux; `[0-9A-Fa-f]`
[[:blank:]]	Espace et tabulation
[[:space:]] ou \\s	Caractères d'espacement par ex. espace, tabulation, retour chariot, etc
\\S	Tout sauf caractères d'espacement; `[^[:space:]]`
[[:punct:]]	Signes de ponctuation; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Caractères d'imprimerie; `[[:alnum:][:punct:]]`
[[:print:]]	Caractères imprimables; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] ou \\c	Caractères de contrôle; `\n, \r` etc.

Fonctions pour l'Appariement de Motifs




```
> chaine <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> motif <- "t.m"
```

Détecter des motifs

grep(motif, chaine)
```
[1] 1 3
```

grep(motif, chaine, value = TRUE)
```
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics“
```

grepl(motif, chaine)
```
[1]  TRUE FALSE  TRUE
```

stringr::str_detect(chaine, motif)
```
[1]  TRUE FALSE  TRUE
```

Localiser des motifs

regexpr(motif, chaine)
Début et longueur de la première correspondance

gregexpr(motif, chaine)
Début et longueur de toutes les correspondances

stringr::str_locate(chaine, motif)
Début et longueur de la première correspondance

stringr::str_locate_all(chaine, motif)
Début et longueur de toutes les correspondances

Séparer une chaîne en utilisant un motif

strsplit(chaine, motif) ou stringr::str_split(chaine, motif)

Extraire des motifs

regmatches(chaine, regexpr(motif, chaine))
extrait la première correspondance `[1] "tam" "tim"`

regmatches(chaine, gregexpr(motif, chaine))
extrait toutes les correspondances, retourne une liste
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::str_extract(chaine, motif)
extrait la première correspondance `[1] "tam" NA  "tim"`

stringr::str_extract_all(chaine, motif)
extrait toutes les correspondances, retourne une liste

stringr::str_extract_all(chaine, motif, simplify = TRUE)
extrait toutes les correspondances, retourne une matrice

stringr::str_match(chaine, motif)
extrait le premier groupe correspondant

stringr::str_match_all(chaine, motif)
extrait tous les groupes correspondants

Remplacer des motifs

sub(motif, remplacement, chaine)
remplacer la première correspondance

gsub(motif, remplacement, chaine)
remplacer toutes les correspondances

stringr::str_replace(chaine, motif, remplacement)
remplacer la première correspondance

stringr::str_replace_all(chaine, motif, remplacement)
remplacer toutes les correspondances

Métacaractères Spéciaux

\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation
\v	Tabulation verticale
\f	Saut de page

Classes et Groupes de Caractères

.	N'importe quel caractère sauf `\n`
|	Ou, e.g. `(a|b)`
[…]	Lister les caractères autorisés, ex. `[abc]`
[a-z]	Spécifier la plage des caractères
[^ …]	Lister les caractères exclus
(…)	Grouper, permet d'y faire référence avec `\\N` où `N` est un entier

Ancres

^	Début de la chaîne de caractère
$	Fin de la chaîne de caractère
\\b	Chaîne vide à la limite d'un mot
\\B	Pas à la limite du mot
\\<	Début du mot
\\>	Fin du mot

Quantifieurs

*	Correspond au moins zéro fois
+	Correspond au moins une fois
?	Correspond au plus une fois
{n}	Correspond exactement n fois
{n,}	Correspond au moins n fois
{n,m}	Correspond entre n et m fois

Assertions et Conditionnels*

(?=)	Assertion avant (PERL = TRUE), par ex. `(?=yx)`: position suivie par 'xy'
(?!)	Assertion avant négative (PERL = TRUE); position NON suivie par un motif
(?<=)	Assertion arrière (PERL = TRUE), par ex. `(?<=yx)`: position suit 'xy'
(?<!)	Assertion arrière négative (PERL = TRUE); position NE suit PAS un motif
?(if)then	If-then-condition (PERL = TRUE); utilise assertion avant, caractère optionel, etc dans la clause if
?(if)then|else	If-then-else-condition (PERL = TRUE)

*see, e.g. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

Mode général

Par défaut, R utilise les expressions régulières étendues (POSIX). Vous pouvez utiliser PCRE en R-base en spécifiant `PERL = TRUE` ou en utilisant `perl(motif)` pour stringr

Les recherches exactes peuvent être faites en utilisant `fixed = TRUE` en R-base ou en utilisant `fixed(motif)` pour stringr.
Elles deviennent insensibles à la casse en R-base en spécifiant `ignore.case = TRUE` et `regex(motif, ignore_case = TRUE)` pour stringr

Caractères d'échappement

Les métacaractères (`. * +` etc.) doivent être échappés pour être utilisé littéralement.

Pour échapper un caractere, on place `\\` avant le caractère ou on l'encadre par `\\Q...\\E`.

Conversions de casses

Les expressions régulières peuvent être rendues insensibles à la casse avec `(?i)`. Il est ensuite possible d'y faire réference et convertir la chaîne en miniscule ou majuscule avec respectivement `\\L` ou `\\U` (par ex. `\\L\\1`). Ça demande au préalable `PERL = TRUE`.

Appariement gourmand

Par défaut, l'asterique `*` est gourmand, c.-à-d. il apparie toujours la chaîne la plus longue possible. Il peut être rendu paresseux avec `?`, c.-à-d. `*?`. Pour désactiver Le mode gourmand il faut utiliser `(?U)`. Cela active une syntaxe qui rend `(?U)a*` paresseux et `(?U)a*?` gourmand.

Note

Les expressions régulières peuvent être créées plus convivialement en utilisant des packages R comme rex ou rebus.

Updated: 08/19