# TP Spécification de langages

L'objectif de cette séance est de découvrir différents types d'outils qui exploitent la spécification de langages sous la forme d'expressions régulières et de grammaires. Il s'agit d'outils de recherche (`egrep`), de transformation (`vi`, `sed`) et d'analyse d'informations structurées décrites sous la forme de langages.

En premier lieu, nous allons exploiter le standard « *POSIX 1003.2 regular expressions* »qui décrit le format des expressions régulières utilisées par les outils du système d'exploitation Unix tels `egrep` (recherche d'informations dans un texte), `vi` (version `vim`, éditeur de texte) ou `sed` (transformation de texte). Le format des expressions régulières est disponible dans la documentation accessible par la commande `man re_format`.

En second lieu, nous utiliserons les outils `ocamllex` (générateur d'analyseur lexical) et `menhir` (générateur d'analyseur syntaxique) pour construire un outil de reconnaissance de la structure d'un document qui affichera les règles de production utilisée pour analyser le document.

Lorsque vous écrivez une expression régulière, vous pouvez utiliser l'application `https://www.debuggex.com/` pour visualer le langage décrit par une expression régulière sous une forme graphique similaire à la notation de Conway pour les grammaires. Ceci peut vous aider à comprendre pourquoi votre expression régulière n'accepte pas le langage auquel vous pensez. Attention, cet outil utilise des expressions régulières avec plusieurs formats qui peuvent être légèrement différents les uns des autres, et qui ne correspondent pas exactement au format accepté pour les outils que nous exploitons dans cette séance, même s'ils en sont très proches.

## 1 Préliminaires

1. Récupérer sur `moodle` l'archive `tp5-langage.tgz` et déposer la dans le répertoire créé pour cette séance de travaux pratiques.
2. Désarchiver son contenu dans le répertoire avec la commande :

```
tar xzvf tp5-langage.tgz
```

## 2 Exploitation de texte avec les outils Unix

### 2.1 Recherche d'informations avec l'outil `egrep`

Placez vous dans le répertoire `tp5-langages/outils-unix`.

Ce répertoire contient le fichier `exemple.txt` qui est composé de lignes de texte comportant des informations de différentes natures sur chaque ligne avec un commentaire précisant la nature de chaque information.

L'outil `egrep` reconnait dans un document texte les lignes contenant une séquence de caractères acceptée par une des expressions régulières transmises comme paramètres de l'outil. Il affiche sur la sortie standard (`stdout`, la console usuellement) les lignes reconnues.

La documentation de cet outil est accessible par la commande habituelle :

```
man egrep
```

La commande suivante affiche les lignes contenant au moins une suite de chiffres :

```
egrep -e "[0-9]+" exemple.txt
```

Vous pouvez directement exécuter les commandes dans un terminal pour visualiser directement le résultat.

Vous pouvez utiliser le fichier `commandes-TP5.sh` fourni dans le répertoire dans lequel vous saisirez les différentes commandes ainsi que des messages affichant la commande exécutée. Vous pouvez visualiser le contenu de ce fichier avec la commande `more commandes-TP5.sh` et l'éditer avec la commande `vi commandes-TP5.sh` (voir enseignement *Environnement Informatique*).

```
#!/bin/sh
echo "Affichage des lignes contenant des nombres entiers naturels :"
egrep -e "[0-9]+" exemple.txt
echo "Affichage des lignes contenant des nombres entiers relatifs :"
echo "Affichage des lignes contenant des nombres décimaux :"
echo "Affichage des lignes contenant des nombres rationnels :"
echo "Affichage des lignes contenant des nombres complexes rationnels :"
```

Vous pouvez ensuite rendre ce fichier exécutable avec la commande :

```
chmod u+x commandes_TP5.sh
```

Et l'exécuter avec la commande :

```
./commandes_TP5.sh
```

1. Afficher les lignes contenant un nombre entier naturel avec la commande précédente
2. Quelles sont les lignes affichées ? Pourquoi ?
3. Modifier l'expression régulière en conséquence (vous pouvez utiliser le caractère `$` qui représente la fin de la ligne donc le suffixe du nombre ainsi que le caractère espace qui représente le préfixe du nombre).
4. Afficher les lignes contenant un nombre entier relatif
5. Afficher les lignes contenant un nombre décimal
6. Afficher les lignes contenant un nombre rationnel
7. Afficher les lignes contenant un nombre rationnel complexe

## 2.2 Recherche et remplacement de texte avec l'outil `vim`

L'éditeur de texte `vim` est une version modernisée de l'outil historique `vi` disponible sous Unix pour éditer des documents texte. Cet outil intégre l'éditeur de texte ligne à ligne `ed` qui permet de rechercher des lignes de texte contenant des séquences de caractères acceptées par une expression régulière de manière similaire à `egrep`. Pour exécuter une commande `ed`, il faut quitter le mode édition de `vim` grâce au caractère `esc`, puis passer en mode commande `ed` grâce à la touche `:`. Le caractère `:` apparaît seul sur la ligne en bas de la console. Il faut ensuite saisir le commande `ed` souhaitée.

La commande pour la recherche d'une ligne dans le texte suivant le curseur contenant une séquence de caractères acceptée par une expression régulière est le caractère `/` suivie de l'expression régulière et du caractère `/`. Cette commande positionne le curseur au début de la ligne contenant la séquence de caractères. La commande `n` de `vim` permet de passer à la séquence suivante acceptée par l'expression. L'utilisation des caractères `?` au lieu de `/` effectue une recherche dans le texte précédent le curseur.

Par exemple, la commande `/[1-9][0-9]*/` recherche une suite de chiffres non vide commençant par un chiffre non nul.

Attention, si vous voulez utiliser la répétition au moins une fois avec l'outil `vim`, il faut banaliser l'opérateur `\+` .

La commande pour rechercher et remplacer une séquence de caractères dans le texte suivant le curseur est le caractère `s/` suivie de l'expression régulière, du caractère `/`, de la séquence de caractères remplaçant la séquence acceptée par l'expression régulière et de `/`.

Par exemple, la commande `s/[1-9][0-9]*/0/` recherche une suite de chiffres non vide commençant par un chiffre non nul et la remplace par le chiffre nul. Un seul remplacement est effectué.

Il est possible de préciser les lignes du texte sur lesquelles la commande s'applique en précédant la commande par :

- le numéro de la ligne concernée ( `$` désignant la dernière ligne)
- le numéro de la première ligne concernée suivi de `,` et du numéro de la dernière ligne concernée ( `1,$` désignant tout le texte). Attention, la commande est alors appliquée sur chaque ligne concernée et pas une seule fois sur l'ensemble des lignes.

Par exemple, la commande `5,15s/[1-9][0-9]*/0/` recherche une suite de chiffres non vide commençant par un chiffre non nul dans sur chaque ligne du texte allant de la ligne 5 à la ligne 15 et la remplace par le chiffre nul.

Si la fin de la commande `/` est suivie de `g`, le remplacement est effectué globalement sur l'ensemble du texte autant de fois que cela est possible sur la même ligne.

L'objectif de cet exercice est d'extraire les informations textuelles contenues dans un document XML, c'est-à-dire éliminer les étiquettes d'éléments. Le langage des étiquettes d'éléments en XML est défini par :

- Une étiquette de début commence par `<`, suivi d'un identificateur XML, d'une séquence d'attributs valués XML et se termine par `>`

— Une étiquette de fin commence par `<`, suivi de `/`, d'un identificateur XML et se termine par `>` ;
— Une étiquette d'élément complet commence par `<`, suivi d'un identificateur XML, d'une séquence d'attributs valués XML et se termine par `/` et `>`
— Un attribut valué XML est un identificateur XML suivi de `=` suivie d'une chaîne de caractères telle que décrite dans la séance de TD précédente

1. Dupliquer les fichiers `enseignant-element.xml-vim` et `enseignant-attribut.xml-vim`
2. Editer le texte des copies des fichiers `enseignant-element.xml-vim` et `enseignant-attribut.xml-vim` avec `vim`
3. Supprimer les balises de fin d'éléments XML
4. Supprimer les balises de début d'éléments XML
5. Que constatez vous ?

### 2.3 Transformation de texte avec l'outil `sed`

L'outil `sed` (qui signifie "*stream editor*") permet d'appliquer une commande au format `ed` sur chaque ligne d'un texte passé en paramètre de l'outil.

Par exemple, la commande `sed -E -e "s/[1-9][0-9]*/0/" < exemple.txt > resultat.txt` recherche une suite de chiffres non vide commençant par un chiffre non nul et la remplace par le chiffre nul. La transformation est appliquée une fois sur chaque ligne du fichier source. Pour l'appliquer autant de fois que possible, il faut ajouter le suffixe `g`.

1. Appliquer l'outil `sed` sur les fichiers `enseignant-element.xml-sed` et `enseignant-attribut.xml-sed` pour éliminer les balises de début et fin d'éléments XML.

## 3 Construction d'un outil d'analyse

Un outil d'analyse de la structure d'un document exprimé dans un langage est composé de plusieurs étapes :
— l'analyse lexicale [1] illustrée dans cette séance ;
— l'analyse syntaxique [2] illustrée dans cette séance ;
— l'analyse sémantique [3] qui sera l'objet des UE (Sémantique et) Traduction des Langages dans plusieurs majeures de deuxième année.

### 3.1 Génération d'analyseur lexical : L'outil camllex

Un analyseur lexical est spécifié sous la forme d'une séquence d'associations entre une expression régulière et une action. L'action est effectuée lorsque l'analyseur lexical reconnait un mot qui fait partie du langage décrit par l'expression régulière associée. Si plusieurs expressions régulières reconnaissent le même mot, seule l'action associée à la première expression régulière est exécutée. Si plusieurs mots reconnus par des expressions régulières contiennent un même préfixe, alors l'action exécutée est celle associée à l'expression régulière qui reconnait le mot le plus long.

Pour réaliser cette séance, nous allons utiliser l'outil `camllex` [4] qui traduit la spécification d'un analyseur lexical en un programme `caml` [5] qui implante un analyseur lexical sous la forme d'un automate fini.

### 3.2 Génération d'analyseur syntaxique : L'outil menhir

Un analyseur syntaxique est spécifié sous la forme d'une grammaire dont les règles de production peuvent contenir des actions sémantiques. Ces actions sont effectuées lorsque l'analyseur syntaxique reconnait un mot qui fait parti du langage décrit par la grammaire associée. La grammaire peut également contenir la spécification des unités lexicales pour assurer la communication avec l'analyseur lexical.

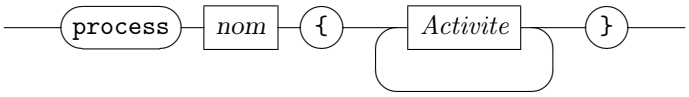
1. `http://en.wikipedia.org/wiki/Lexical_analysis`
2. `http://en.wikipedia.org/wiki/Parsing`
3. `http://en.wikipedia.org/wiki/Semantics`
4. `http://caml.inria.fr/pub/docs/manual-ocaml-4.05/lexyacc.html`
5. `http://caml.inria.fr/pub/docs/manual-ocaml-4.05/`

Pour réaliser cette séance, nous allons utiliser l'outil `camlyacc`[6] qui traduit la spécification d'un analyseur syntaxique en un programme `caml`[7] qui implante un analyseur syntaxique sous la forme d'un automate fini à pile. Cet outil est conçu pour collaborer avec un analyseur lexical implanté avec l'outil `camllex`.

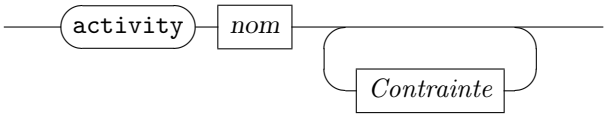
## 3.3 Etude d'un exemple

Soit le langage de modélisation de processus définie par le diagramme de Conway et la grammaire EBNF équivalente suivant :

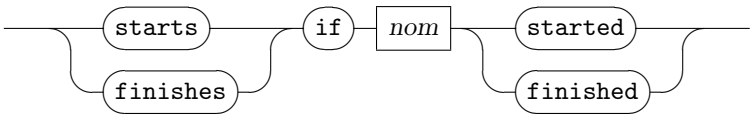
*Processus*




*Activite*




*Contrainte*




```
<Processus> ::= process nom \{ <Activite> { <Activite> } \}
<Activite> ::= activity nom { <Contrainte> }
<Contrainte> ::= ( starts | finishes ) if nom ( started | finished )
```

Les unités lexicales reconnues par l'analyseur lexical et transmise à l'analyseur syntaxique au fur et mesure de la lecture d'un fichier sont définies en entête du fichier `parserProcessus.mly`.

L'objectif est d'étudier l'exemple suivant en s'appuyant sur la documentation des outils `camllex` et `camlyacc` : `http://caml.inria.fr/pub/docs/manual-ocaml-4.05/lexyacc.html`. Plus précisément, nous utiliserons l'outil `menhir` compatible avec le format de fichier de `camlyacc` mais qui fournit un diagnostic plus facile à interpréter si la grammaire est ambiguë (voir `http://gallium.inria.fr/~fpottier/menhir/`).

L'analyseur lexical est défini dans le fichier `lexerProcessus.mll`.

L'analyseur syntaxique est défini dans le fichier `parserProcessus.mly`. Il utilise l'analyseur lexical généré.

Le programme principal qui utilise l'analyseur syntaxique généré est défini dans le fichier `mainProcessus.ml`. Il prend en paramètre un nom de fichier, ouvre ce fichier, effectue l'analyse lexicale en affichant les unités lexicales reconnues puis l'analyse syntaxique en affichant les règles de production appliquée dans la construction de la dérivation. Il s'agit d'une analyse ascendante qui construit l'arbre des feuilles vers la racine.

Pour compiler le programme, utilisez la commande `make` qui produit l'exécutable `mainProcessus`. Cet exécutable prend en paramètre le nom du fichier que l'on veut analyser. Des exemples se trouvent dans le répertoire `tests`.

## 3.4 Extension des analyseurs lexical et syntaxique

L'objectif de cet exercice est d'étendre les fichiers `lexerProcessus.mll` et `parserProcessus.mly` pour traiter les extensions suivantes du langage de modélisation des Processus.

1. Nous voulons enrichir le langage de modélisation de processus avec une notion de ressources disponibles pour un processus qui sont utilisées par les activités.
   — Une ressource est déclarée au même niveau que les activités en utilisant le mot clé `resource` suivi du nom de la resource, puis du mot clé `amount` et d'un entier naturel

6. `http://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html`
7. `http://caml.inria.fr/pub/docs/manual-ocaml/`

— Une demande de ressource par une activité est déclarée après le nom de l'activité et avant les contraintes sur l'activité par : le mot clé `requires` suivi d'un entier naturel, le nombre d'éléments utilisés, et du nom de la ressource utilisée

Voici un exemple respectant la syntaxe étendue :

```
process ABC {
   resource R amount 5
   activity A
      requires 2 R
   activity B
   activity C
      requires 3 R
      starts if A started
      finishes if B finished
}
```

Le fichier fourni contient déjà la déclaration des ressources, vous devez le modifier pour ajouter l'utilisation des ressources.

2. Nous voulons enrichir le langage de modélisation de processus avec une notion d'activité composite (activité composée d'autres activités).
   — Le contenu des activités composites est déclaré après le nom de l'activité composite et avant les demandes de ressources et les contraintes par : une `{` (accolade ouvrante) suivie de déclarations d'activités et de `}` (accolade fermante)
   — Une activité composite ne contient pas de déclarations de resources

   Voici un exemple respectant la syntaxe étendue :

```
process ABC {
   resource R amount 5
   activity A {
      activity A1 {
         activity A11
      }
         requires 1 R
      activity A2
         requires 2 R
         starts if A1 finished
   }
      requires 2 R
   activity B
   activity C requires 3 R
      starts if A started
      finishes if B finished
}
```

   Modifier les fichiers fournis pour prendre en compte les activités composites.