Professor Jean-Gabriel GANASCIA
Université Pierre et Marie Curie

**Logic & Knowledge Representation Course**

# Assignment #3

## Programming a syntactical parser in PROLOG and with DCG

***Deadline**: the projects have to be submitted to the Doodle website **before Sunday, January 10, 2016***

*Each individual submission is composed of one PROLOG file, the explanations being given as comments, and on text file, for the description of the program and the tests.*

The goal of this project is to program an interpreter for a few UNIX commands. We shall consider here the commands `cal`, `cat`, `cp` and `grep`. The syntax of these commands is briefly described in the annex.

1. Express the syntax of the different commands

2. Write a PROLOG program `read_command(C)` that reads a line on the current input stream and that returns the list of ascii codes it contains.

3. Write a PROLOG program that parses the command line obtained in the previous question and that returns the command under the form of a PROLOG terms defined as follows:

   - Command **`cal`**: return the term **`calendar(Month, Year)`** – if the Year and the Month are not specified give the current Year and Month
   - Command **`cat`**: return the term **`concatenate(option_list, file_list)`**
   - Command **`cp`**: return the term **`copy(option_list, list_source_files, target_file)`**
   - Command **`grep`**: return the term **`search_expr(option_list_1, option_2, expression, list_files)`**
4. Write the same program using the DCG (Definite Clause Grammar) formalism,

5. Let now suppose that the symbol '>' (resp. '>>') followed by a file name create a file redirect the output of the preceding command to this file, erasing and overwriting it, (resp. appending it for '>>'). For instance, **`cat file1 file2 >file`** sends the concatenation of file1 and file 2 to file.

   - Write a PROLOG program able to parse commands with the '>' and '>>' symbols using the two functions **`send(input, file)`** and **`append(input, file)`**.

   - Write the same program using the DCG formalism.

## Annex

### *Unix Commands*

Note that the elements contained in square brackets [ ] are optional and that the options separated by '|' are exclusive.

Command **`cal`**:

- *Syntax*: **`cal [[month] year]`**
  where month ∈ [1..12] is optional and year ∈ [1, 9999] is also optional. Without any argument print the current month

Command **`cat`**:

- *Syntax*: **`cat [-nbsuvet] [file …]`**
  where [-nbsuvt] designates an optional list of options, each option being one character of {n, b, s, u, v, e, t}
  **`[file …]`** designates a non empty list of file

Command **`cp`**:

- *Syntax*: **`cp [-r|-R] [-f] [-i] [-p] file1 [file2 …] target`**
  where [-nbsuvt] designates an optional list of options, each option being one character of {n, b, s, u, v, e, t}

Command **`grep`**:

- *Syntax*: **`grep [-bcihlnvsy] [-e] expr [file …]`**
  where **`[-bcihlnvsy]`** designates an optional list of options, **`[-e]`** an optional option **`e`**, **`expr`** the expression to be searched and **`[file …]`** a non empty list of files in which the expression is searched.

### *BNF formalism*

BNF means *Backus-Naur Form*, which is a widely spread formalism for the notation of context-free grammars.

It is composed of a sequence of rules of the type **`<Symbol> ::= __expression__`** where **`<Symbol>`** is the name of a non terminal symbol, and **`__expression__`** is a sequence of symbols. The symbol **`'|'`** (vertical bar) means a choice. The symbol * in **`<S>*`** means a repetition of **`<S>`**. A symbol that never appears in the left hand side of a rule is considered as terminal.