# Trabajo Práctico 2: Programación Lógica

Paradigmas de Lenguajes de Programación — 1er cuat. 2018

Fecha de entrega: 5 de Junio de 2018

## 1. Introducción

En este trabajo práctico trabajaremos con expresiones regulares en *prolog*.

Las expresiones regulares son una secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Su sintaxis esta definida según la siguiente gramática:

```
(1) E ::=
(2) E ::= symbol
(3) E ::= E E
(4) E ::= E | E
(5) E ::= E *
```

La regla `(1)` permite aceptar la cadena vacía (“”). La regla `(2)` permite aceptar un símbolo, el cual puede ser cualquier caracter (letras, dígitos, etc.). La regla `(3)` es la concatenación de dos expresiones regulares. La regla `(4)` permite aceptar alguna de las dos expresiones regulares pero no necesariamente ambas. Y por último, la regla `(5)` permite aceptar zero o más veces una expresión regular.

Ejemplos:

| Expresión regular | Cadenas aceptadas |
|---|---|
| `a` | a |
| `ab` | ab |
| `a\|b` | a, b |
| `(a\|)c` | ac, c |
| `a*b` | b, ab, aab, ... |

## 2. Implementación

Para representar las expresiones regulares en *prolog* utilizaremos functores que representen los operadores de las expresiones de la siguiente manera. La expresión regular que acepta la cadena vacía será representada con el término `empty`. Para simplificar la implementación preveemos el predicado `symbol(?C)`, que es verdadero cuando el caracter `C` es un caracter válido.

| Expresión regular | Representación en *prolog* |
|---|---|
| | `empty` |
| `a` | `a` |
| `ab` | `concat(a, b)` |
| `a\|b` | `or(a, b)` |
| `a*` | `star(a)` |

## 3. Ejercicios

### Ejercicio 1

Definir el predicado `tieneEstrella(+RegEx)` que es verdadero cuando la expresión regular `RegEx` contiene al menos una estrella.

Por ejemplo:

```
?- tieneEstrella(or(a, b)).
false.
?- tieneEstrella(or(concat(star(a), b), c)).
true.
?- tieneEstrella(star(concat(a, b))).
true.
```

### Ejercicio 2

Definir el predicado `longitudMaxima(+RegEx, -N)` que es verdadero cuando las cadenas aceptadas por la expresión regular `RegEx` tienen longitud máxima `N`.

Observemos que si `RegEx` acepta cadenas infinitas, el predicado debe fallar.

Por ejemplo:

```
?- longitudMaxima(a, N).
N = 1.
?- longitudMaxima(concat(a, b), N).
N = 2.
?- longitudMaxima(concat(or(a, b), b), N).
N = 2.
?- longitudMaxima(concat(or(a, star(b)), b), N).
false.
```

### Ejercicio 3

Definir el predicado `cadena(?X)` que es verdadero cuando `X` es una lista de los símbolos del alfabeto. En caso de que `X` no esté instanciada, este predicado debe generar todas las cadenas posibles (sin repeticiones).

Por ejemplo:

```
?- cadena(X).
X = [] ;
X = [a] ;
X = [b] ;
X = [c] ;
X = [a, a] ;
X = [a, b] ;
X = [a, c] ;
X = [b, a] ;
X = [b, b] ;
X = [b, c] ;
X = [c, a] ;
X = [c, b] ;
X = [c, c] ;
X = [a, a, a]
```

### Ejercicio 4

Definir el predicado `match_inst(+Cadena,+RegEx)` que, dada una cadena y una expresión regular, es verdadero cuando `Cadena` es aceptada por `RegEx`.

Por ejemplo:

```
?- match_inst([], empty).
true.
?- match_inst([a], empty).
false.
?- match_inst([a], b).
false.
?- match_inst([b], b).
true.
?- match_inst([c], or(a, b)).
false.
?- match_inst([b], or(a, b)).
true .
?- match_inst([a, a, b], concat(star(a), b)).
true.
?- match_inst([b, a], concat(star(a), b)).
false.
```

### Ejercicio 5

Definir el predicado `match(?Cadena,+RegEx)` que extienda `match_inst` para que además pueda generar todas las cadenas aceptadas por la expresión regular `RegEx`.

Por ejemplo:

```
?- match(X, a).
X = [a] ;
false.
?- match(X, empty).
X = [] ;
false.
?- match(X, concat(b, or(c,a))).
X = [b, c] ;
X = [b, a] ;
false.
?- match(X, concat(star(a), b)).
X = [b] ;
X = [a, b] ;
X = [a, a, b] ;
X = [a, a, a, b] ;
...
?- match(X, star(concat(star(a), star(b)))).
X = [] ;
X = [a] ;
X = [b] ;
X = [a, a] ;
X = [a, b] ;
X = [b, a] ;
X = [b, b] ;
X = [a, a, a] ;
...
```

### Ejercicio 6

Definir el predicado `diferencia(?Cadena, +R1, +R2)` que es verdadero cuando `Cadena` es aceptada por la expresión regular `R1`, y no es aceptada por la expresión regular `R2`.

Por ejemplo:

```
?- diferencia(X, star(a), empty).
X = [a];
X = [a, a];
...
?- diferencia(X, or(a, b), b).
X = [a].
?- diferencia(X, star(or(a, b)), star(b)).
X = [a];
X = [a, a];
X = [a, b];
X = [b, a];
X = [a, a, a];
X = [a, a, b];
...
```

**Ejercicio 7**

Definir el predicado `prefijoMaximo(?Prefijo, +Cadena, +RegEx)` que es verdadero cuando `Prefijo` es el prefijo más largo de `Cadena` aceptado por la expresión regular `RegEx`.

Por ejemplo:

```
?- prefijoMaximo(P, [a,a,a,b], star(a)).
P = [a, a, a].
?- prefijoMaximo(P, [a,a,a,b,b,a], concat(star(a), star(b))).
P = [a, a, a, b, b].
```

**Ejercicio 8**

Definir el predicado `reemplazar(+Cadena, +RegEx, +T, -R)` que es verdadero cuando `R` es la lista resultante de reemplazar por la cadena `T` las subcadenas de longitud máxima aceptadas por la expresión regular `RegEx`. No deben tenerse en cuenta reemplazos de cadenas vacías.

Por ejemplo:

```
?- reemplazar([a, b], [a], 1, X).
X = [1, b].
?- reemplazar([a, a, a, b, b], concat(star(a), b), [1], X).
X = [1, 1].
?- reemplazar([a, a, a, b, b, a, b,c], concat(star(a), star(b)), [1], X).
X = [1, 1, c].
?- reemplazar([c, a, a, a, c], star(a), [1], X).
X = [c, 1, c].
?- reemplazar([c, a, a, a, c, a, a, c], star(a), [1], X).
X = [c, 1, c, 1, c].
```

# 4. Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado.

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp2.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

⚠ Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

## 5. Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía → Programación Lógica*).

Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Cosas útiles* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.