Lógica y Programación
Trabajo práctico Prolog

21 de octubre de 2025

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 Predicados

3.1 tieneEstrella(+RegEx)

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.
```

3.2 longitudMaxima(+RegEx, -N)

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.
```

3.3 cadena(?X)

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]
```

3.4 match_inst(+Cadena,+RegEx)

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.
```

3.5 match(?Cadena,+RegEx)

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] ;
...
```

3.6 diferencia(?Cadena, +R1, +R2)

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];
...
```

4 Entrega

La entrega es por mail a la lista lds-doc-lyp@listas.unq.edu.ar. Se debe incluir el código fuente junto con un pequeño informe en formato PDF en donde además de los predicados definidos se comente la idea de implementación en cada uno de los predicados. **Fecha de entrega 04-11-2025.**

Si el email rebota, deben subir los archivos a Google Drive o a donde quieran y enviar los links. Es responsabilidad de ustedes asegurarse de que nos llegue el trabajo.