Trabajo Práctico 2: Programación Lógica

Paradigmas de Lenguajes de Programación — 1er cuat. 2016

Fecha de entrega: 7 de junio

1. Introducción

El objetivo del trabajo es descubrir mensajes codificados con símbolos. Similar a los que se pueden encontrar en revistas de ocio de verano.

Cada letra del mensaje está representada por un único símbolo. Por ejemplo: cuadrado, triángulo, círculo, etc. A su vez a cada símbolo le corresponde una letra distinta.
Un mensaje cifrado contiene los espacios correspondientes al mensaje original.
Por ejemplo, el mensaje cifrado

bien podría corresponder a *la casa*.

Para evitar programar en emojis, el mensaje cifrado anteriormente lo podemos representar en prolog como: `[rombo, cuadrado, espacio, perro, cuadrado, sol, cuadrado]`.

Se cuenta a su vez con un diccionario de las palabras válidas que pueden ser utilizadas en los mensajes. El enunciado del trabajo viene acompañado con archivos `.txt` con las listas de palabras y además tendrán código de predicados que permiten cargar estos archivos.

1.1. Aclaraciones

- Cada símbolo va a corresponder a una letra.
- Cada letra puede corresponderse con un solo símbolo.
- El espacio se corresponde siempre con el espacio.
- No hay signos de puntuación.
- No hay un listado fijo de símbolos.
- No hay un listado fijo de caracteres. Depende del diccionario.
- Los caracteres con tilde son distintos los que no tienen tilde: `a` $\neq$ `á`.

2. Descifrado de mensajes

La base de conocimientos contiene un predicado `diccionario/1` que permite instanciar o verificar la pertenencia de una palabra representada como *string* en el diccionario disponible para descifrar el mensaje.
Es decir, un predicado análogo al siguiente:

```
diccionario("el").
diccionario("la").
diccionario("casa").
% etc
```

Ahora bien, como se desea poder cargar distintos diccionarios, el programa a realizar arrancará con un diccionario vacío. Se cuenta con el predicado `cargar/1` que cargará el contenido del nombre de archivo indicado e imitará el predicado `diccionario/1` descripto antes.
Ejemplo:

```
$ swipl tp.pl
?- diccionario(S).
false

?- cargar("1000_formas").
true.

?- diccionario(S).
S = "de" ;
S = "la" ;
S = "que" ;
S = "el" ;
S = "en" ;
...
```

El predicado `cargar/1` vacía el diccionario actual antes de cargar el diccionario especificado.

2.1. Ejercicios

Para **todos** los ejercicios se requiere, además de resolverlos, estudiar su **reversibilidad**, es decir:

- Especificar (mediante un comentario en el código) la instanciación de cada uno de los parámetros de los predicados que se implementen (tanto los correspondientes a los ejercicios como los auxiliares que agreguen). Esto debe respetar **su implementación**, incluso si es más flexible que el objetivo inicial del predicado.
- Explicar el por qué de esas instanciaciones para los predicados principales de cada ejercicio. Esto **no** debe convertirse en una explicación de qué hace el predicado, sino el por qué funciona o no bajo distintas instanciaciones.
- En caso de existir una combinación extraña de las instanciaciones, aclararlo. Por ejemplo, si `p(?X, ?Y)` funciona bien si alguna de las dos está instanciada pero no si las dos vienen sin instanciar, debería estar aclarado y explicado (ejemplo de este tipo de problemas: `append/3`).

Ejercicio 1

En Prolog un *string* no es una lista. Por ende, para operar con caracteres individuales se utiliza su representación numérica de código ASCII. Por ejemplo: el espacio ' ' es 32, la 'a' es 97.
Implementar el predicado `diccionario_lista/1` que sea equivalente a `diccionario/1`, pero que trabaje con listas de códigos de caracteres en lugar de strings.
Ayuda: Utilizar `string_codes/2`.

```
?- cargar("dicc0.txt"), diccionario_lista(L).
L = [101, 108] ; %% "el"
L = [108, 97] ;  %% "la"
...
```

Ejercicio 2

Escribir el predicado `juntar_con(L,J,R)` que, siendo `L` una lista de listas y `J` un elemento, instacie en `R` la lista resultante de unir los elementos de `L` intercalando `J` entre cada elemento y el siguiente.

```
?- juntar_con([[x],[x,y],[z]],a, R).
R = [x, a, x, y, a, z] ;
false.
```

Ejercicio 3

Implementar el predicado `palabras(S, P)`, que dado un mensaje cifrado `S`, instancie en `P` una lista de listas de símbolos correspondientes a las palabras. Es decir, que separe por el átomo `espacio`.

```
?- ej(1, S), palabras(S, P).
S = [rombo, cuadrado, espacio, perro, cuadrado, sol, cuadrado],
P = [[rombo, cuadrado], [perro, cuadrado, sol, cuadrado]] ;
```

Ejercicio 4

Una lista de tuplas clave/valor puede ser usada para representar mapeos o estructuras de diccionario.

Implementar el predicado `asignar_var(A, MI, MF)` que, dados un átomo `A` y un mapeo inicial `MI` con elementos de la forma `(atomo, variable)`, genere un mapeo `MF` que contenga los mismos elementos que `MI` más una correspondencia para el símbolo `A` (si no estaba presente en `MI`). A medida que se introduzcan átomos nuevos en el mapeo, las variables asociadas deberán ser frescas.

```
?- asignar_var(rombo, [], M).
M = [(rombo, _G4012)],
false.

?- asignar_var(cuadrado, [(rombo, _G4012)], M).
M = [(cuadrado, _G4013),(rombo, _G4012)],
false.

?- asignar_var(rombo, [(cuadrado, _G4013),(rombo, _G4012)], M).
M = [(cuadrado, _G4013),(rombo, _G4012)],
false.
```

Responder en un comentario en el código: ¿por qué funciona `asignar_var/3`?

Ejercicio 5

Implementar el predicado `palabras_con_variables(P, V)` que si `P` es una lista de listas de átomos, instancie en `V` una lista de listas de variables. Tener en cuenta que a cada átomo le corresponde una única variable.

```
?- ej(1, S), palabras(S, P), palabras_con_variables(P, V).
S = [rombo, cuadrado, espacio, perro, cuadrado, sol, cuadrado],
P = [[rombo, cuadrado], [perro, cuadrado, sol, cuadrado]],
V = [[_G3061, _G3079], [_G3100, _G3079, _G3124, _G3079]],
false.
```

Ejercicio 6

Implementar el predicado `quitar(E, L, R)`, que siendo `E` un átomo y `L` una lista de átomos, instancie en `R` el resultado de quitar todas las apariciones de `E` en `L`. `L` puede contener elementos instanciados y no instanciados. `E` puede no estar instanciado.

```
?- quitar(z,[A,B,A,z],L).
L = [A, B, A] ;
false.

?- quitar(A,[A,B,A,z],L).
L = [B, z] ;
false.
```

Ejercicio 7

Escribir el predicado `cant_distintos(L, S)` que, dada `L` una lista de átomos y variables, instancie en `S` la cantidad de elementos distintos que contiene `L`.

```
?- cant_distintos([A,B,A], N).
N = 2 ;
false.
```

Ejercicio 8

Escribir el predicado `descifrar(S, M)` que, dada una lista de símbolos con un mensaje secreto (`S`), instancie en `M` los posibles mensajes descifrados (uno por vez) utilizando las palabras del `diccionario/1`.

```
?- cargar("dicc0.txt").
true.
% diccionario("el").
% diccionario("la").
% diccionario("casa").
% diccionario("cosa").

?- ej(1, S), descifrar(S, M).
```

```
S = [rombo, cuadrado, espacio, perro, cuadrado, sol, cuadrado],
M = "la casa";
false.
```

Nota: recordar las aclaraciones hechas en la introducción.

Ejercicio 9

Escribir el predicado `descifrar_sin_espacios(S, M)` que, dada una lista de símbolos sin espacios `S`, instancie en `M` los posibles mensajes descifrados utilizando las palabras del `diccionario/1`, intercalando los espacios necesarios para que el mensaje use todos los símbolos.

```
?- cargar("dicc1.txt").
true.
% diccionario("casa").
% diccionario("miento").
% diccionario("de").
% diccionario("flor").

?- ej(3, S), descifrar_sin_espacios(S, M).
S = [rombo, cuadrado, perro, cuadrado, sol,
     luna, triangulo, estrella, arbol, gato],
M = "casa miento";
M = "casa de flor";
false.
```

Ejercicio 10

Escribir el predicado `mensajes_mas_parejos(S, M)` que, dada una lista de símbolos sin espacios `S`, instancie en `M` los mensajes resultantes de descifrar `S` intercalando espacios de todas las formas posibles de manera tal que la desviación estándar de la longitud las palabras de `M` sea mínima (esto no significa que sea 0, sino que sea menor o igual que la de cualquier otra decodificación posible).
La desviación estándar se calcula como:

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$$

Donde x_i es la longitud de cada palabra, $\overline{x}$ la media de las longitudes de las palabras y n la cantidad de palabras.

```
?- cargar("dicc1.txt").
true.

?- ej(3, S), mensajes_mas_parejos(S, M).
S = [rombo, cuadrado, perro, cuadrado, sol, luna, triangulo, estrella, arbol, gato],
M = "casa de flor";
false.
```

Como se ve en el ejercicio anterior, los mensajes que se instancian a partir de S insertando los espacios en cualquier posición son:

```
M = "casa miento";
M = "casa de flor";
```

Sin embargo `mensajes_mas_parejos` devuelve solo la instancia con el segundo mensaje debido a que el primero posee una desviación estándar mayor.

La desviación estándar para `casa miento` es

$$\sqrt{\frac{\sum_{i=1}^{2}(x_i-5)^2}{2}} = \sqrt{\frac{(4-5)^2+(6-5)^2}{2}} = \sqrt{\frac{1+1}{2}} = \sqrt{1} = 1$$

mientras que para `casa de flor` es

$$\sqrt{\frac{\sum_{i=1}^{3}(x_i-3{,}33)^2}{3}} = \sqrt{\frac{(4-3{,}33)^2+(2-3{,}33)^2+(4-3{,}33)^2}{3}} = \sqrt{\frac{0{,}45+1{,}77+0{,}45}{3}} = \sqrt{0{,}89} = 0{,}94$$

En caso de haber más de una solución, el predicado debe instanciar todas (una por vez).

3. 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.

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

4. 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.