Trabajo Práctico 2
Programación Lógica

Paradigmas de Lenguajes de Programación
1er cuatrimestre, 2013

Fecha de entrega: 6 de junio

1. Introducción

El objetivo de este trabajo es escribir funciones para ejemplificar y probar distintas nociones de distancia entre dos cadenas. En el contexto de este trabajo práctico, una cadena es una secuencia finita de ceros y unos, representada en prolog como una lista en la que cada elemento es uno de los átomos `0` o `1`.

Una función de distancia es una función dist que, dadas dos cadenas, devuelve un entero no negativo. Recordamos que, utilizando la notación $\cdot^*$ vista en el trabajo práctico anterior, Σ^* es el conjunto de todas las secuencias finitas de elementos de Σ, en particular, $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, ...\}$ donde ϵ es la secuencia vacía. Utilizando esa notación $\text{dist} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Z}_{\geq 0}$.

Lo que haremos en cada caso es pedir implementar un predicado asociado `dist(+S, ?T, ?D)` que sea verdadero exactamente cuando `S` y `T` representan cadenas (es decir, son listas y sus elementos son exclusivamente los átomos `0` y `1`) y `D` es un entero no negativo de manera que $\text{dist}(s,t) = d$, donde s y t son las cadenas representadas por `S` y `T` respectivamente y d es el entero no negativo representado por `D`.

Notar que `T` y `D` no necesariamente vienen instanciados, con lo cual podemos usar el predicado para "calcular" la distancia entre dos cadenas dadas (instanciando `S` y `T` pero no `D`), para listar una a una todas las cadenas a una distancia dada de una particular (instanciando `S` y `D` pero no `T`) o listar todas las cadenas cada una con su distancia a una particular (instanciando solo `S`) o incluso instanciar parcialmente `T` para establecer un filtrado sobre la lista reportada. Parte de la dificultad del trabajo práctico es poder implementar los predicados de forma prolija de manera que maneje todas las combinaciones de instanciación correctamente con un mismo código.

2. Ejercicios

2.1. Distancia de Hamming

La distancia de Hamming es definida entre cadenas de la misma longitud como la cantidad de posiciones en que las cadenas tienen diferentes símbolos.

$$\begin{aligned} \text{distHam}(\epsilon, \epsilon) &= 0 \\ \text{distHam}(cs, dt) &= \text{distHam}(s,t) + |c - d|. \end{aligned}$$

Por ejemplo, las cadenas `0010` y `0101` están a distancia 3, `00` y `11` a distancia 2 y `00010` y `10010` a distancia 1. Dos cadenas idénticas están a distancia 0.

Escribir un predicado de Prolog `distHam(+S, ?T, ?D)` que sea verdadero exactamente cuando `S` y `T` son cadenas que están a distancia de Hamming `D`.

Ejemplos (el orden de las soluciones puede variar):

```
?- distHam([],[0],N).
false.

?- distHam([0,1],[0,1],N).
N = 0 ;
false.

?- distHam([0,1,0],[0,0,1],N).
N = 2 ;
false.

?- distHam([0,1,0],[X,Y,1],N).
X = 0, Y = N, N = 1 ;
X = Y, Y = 0, N = 2 ;
X = Y, Y = 1, N = 2 ;
X = 1, Y = 0, N = 3 ;
false.

?- distHam([0,1,0],X,2).
X = [0, 0, 1] ;
X = [1, 1, 1] ;
X = [1, 0, 0] ;
false.

?- distHam([plp],[plp],N).
false.
```

2.2. Distancia de prefijos

La distancia de prefijos es definida entre cadenas como la cantidad de símbolos de cualquiera de ellas que no pertenece a un prefijo común. Otra forma de verlo es la suma de las longitudes luego de borrar el prefijo común mas largo de ambas.

$$\begin{aligned} \text{distPref}(s, \epsilon) &= \text{distPref}(\epsilon, s) = |s| \\ \text{distPref}(cs, ct) &= \text{distPref}(s, t) \\ \text{distPref}(cs, (1-c)t) &= |s| + |t| + 2. \end{aligned}$$

Por ejemplo, las cadenas `0010` y `010` están a distancia 5, `00` y `11` a distancia 4 y `0010` y `00101` a distancia 1. Dos cadenas idénticas están a distancia 0.

Escribir un predicado de Prolog `distPref(+S, ?T, ?D)` que sea verdadero exactamente cuando `S` y `T` son cadenas que están a distancia de prefijos `D`.

Ejemplos (el orden de las soluciones puede variar):

```
?- distPref([0,1,1],[0,1,1,1],N).
N = 1 ;
false.

?- distPref([0,1,1],[0,1,X,Y],N).
X = 1, Y = 0, N = 1 ;
X = Y, Y = N, N = 1 ;
X = Y, Y = 0, N = 3 ;
X = 0, Y = 1, N = 3 ;
false.

?- distPref([0,1,1],X,2).
X = [0] ;
X = [0, 1, 1, 0, 0] ;
X = [0, 1, 1, 0, 1] ;
X = [0, 1, 1, 1, 0] ;
X = [0, 1, 1, 1, 1] ;
X = [0, 1, 0] ;
false.

?- distPref([plp],X,N).
false.
```

2.3. Distancia de edición

La distancia edición es definida entre cadenas de no necesariamente la misma longitud como la mínima cantidad de ediciones a realizar desde una de ellas para llegar hasta la otra. Una edición consiste en agregar un símbolo en cualquier parte, remover un símbolo o cambiar un símbolo por su complemento.

$$\begin{aligned} \text{distEd}(\epsilon, s) &= \text{distEd}(s, \epsilon) = |s| \\ \text{distEd}(cs, dt) &= \text{mín}(\text{distEd}(s, t) + |c - d|, \text{distEd}(s, dt) + 1, \text{distEd}(cs, t) + 1) \end{aligned}$$

Notar que el primer parámetro del mín considera el caso en que se modifica el primer símbolo de cs, sólo en caso de ser necesario, para que resulte igual en cs y en dt, el segundo parámetro considera la posibilidad de eliminar el primer símbolo de cs y el último considera la posibilidad de agregar el símbolo d al inicio de cs de manera que el primer símbolo del resultado dcs se pueda igualar al primer símbolo de dt.

Por ejemplo, las cadenas `0010` y `0101` están a distancia 3, `00` y `11` a distancia 2 y `00010` y `10010` a distancia 1. Dos cadenas idénticas están a distancia 0.

Escribir un predicado de Prolog `distEd(+S, ?T, ?D)` que sea verdadero exactamente cuando `S` y `T` son cadenas que están a distancia de edición `D`.

Ejemplos (el orden de las soluciones puede variar):

```
 ?- distEd([0,0,1,1],[1,1,0],N).
N = 3 ;
false.

 ?- distEd([0,0,1],[1,1,X,Y],N).
X = Y, Y = 0, N = 3 ;
X = 0, Y = 1, N = 2 ;
X = 1, Y = 0, N = 3 ;
X = Y, Y = 1, N = 3 ;
false.

 ?- distEd([0,1,1,1],[1|Xs],2).
Xs = [1] ;
Xs = [0, 1] ;
Xs = [1, 0] ;
Xs = [0, 1, 1] ;
Xs = [1, 0, 1] ;
Xs = [1, 1, 0] ;
Xs = [0, 0, 1, 1] ;
Xs = [0, 1, 0, 1] ;
Xs = [0, 1, 1, 0] ;
Xs = [1, 0, 1, 1] ;
Xs = [1, 1, 0, 1] ;
Xs = [1, 1, 1, 0] ;
Xs = [1, 1, 1, 1] ;
Xs = [0, 0, 1, 1, 1] ;
Xs = [0, 1, 0, 1, 1] ;
Xs = [0, 1, 1, 0, 1] ;
Xs = [0, 1, 1, 1, 0] ;
Xs = [0, 1, 1, 1, 1] ;
Xs = [1, 0, 1, 1, 1] ;
false.

?- distEd(plp, X, N).
false.

?- distEd([0,1], [X,plp|Xs], N).
false.

?- distEd([0,1], [X,0|Xs], 0).
false.

?- distEd([0,1], [X,1|Xs], 0).
X = 0, Xs = [] ;
false.
```

Nota importante Cada ejercicio presenta dificultades bastante mayores al anterior. Se recomienda encararlos todos con tiempo. El TP está pensado para que puedan encontrarse con dificultades y zanjarlas pensando un poco y consultando. Es posible que si intentan hacer todo solos sin tiempo para consultar como pasar alguna traba la dificultad les resulte excesiva.

3. Pautas de Entrega

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado debe contar con un comentario donde se explique su funcionamiento. 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 (puede adjuntarse un `.zip` o `.tar.gz`).

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
- Uso de unificación, backtracking, generate and test y reversibilidad de los predicados que correspondan.
- Salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas. 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 principal referencias del lenguaje de programación Prolog, mencionamos:

- http://www.swi-prolog.org/

Se recomienda fuertemente usar los predicados que SWI-Prolog ya trae definidos, en particular para generar/instanciar variables. Recomendamos especialmente examinar los predicados reversibles de listas `append`, `elem` y `length` y de chequeo de instanciación `ground` y `var`, así como recordar los predicados para generate and test dados en clase `entre` y `desde` y el metapredicado `not`, para ver si alguno les puede ser útil (es probable que no necesiten usar todos los nombrados, depende el camino que elijan para cada implementación).

Sugerencias particulares Recomendamos repasar estas sugerencias de ser necesario luego de haber pensado un poco la forma de resolver cada ejercicio.

1. Hacer una versión que funcione con todo instanciado y luego utilizarla para considerar las diferentes posibles instanciaciones.

2. Distinguir los casos con finitas e infinitas soluciones para evitar "cuelgues". En algunos casos es posible resolver ambos casos con el mismo código usando las funciones reversibles de prolog, pero dependiendo el caso puede resultar muy díficil o incluso imposible hacerlo así y se necesita separar en casos.

3. Considerar el predicado `ground` para separar los casos en los que algo viene instanciado de los que no, teniendo en cuenta que no instanciado no quiere decir que es una variable, puede estar parcialmente instanciado y una instanciación parcial puede marcar la diferencia entre finitas e infinitas soluciones de una forma compleja (por ejemplo, se puede fijar la longitud de una lista sin fijar su contenido pasando `[X1,X2,X3,...,Xn]` donde cada `Xi` es una variable distinta).

Por último, acá está el código fuente de este archivo propiamente dicho.