Facultad de Ciencias de la Administración
Universidad Nacional de Entre Ríos

PROYECTO: LÓGICA PARA LAS CIENCIAS INFORMÁTICAS

DEPENDENCIAS FUNCIONALES

Objetivo del Proyecto: El objetivo general del presente proyecto consiste en el diseño e implementación de un conjunto de predicados en Prolog, que permitan, dado un conjunto de dependencias funcionales y conjunto de atributos, obtener un buen cubrimiento.

Es decir, dado un esquema db(LDF, A), donde LDF es una lista de dependencias de la forma `X -> Y`, y A un conjunto de atributos, retorne un DFc, que sea un buen cubrimiento.

Por ejemplo, dado un hecho:

```
db([ab -> c, a -> d, bd -> c], abcd).
```

el programa debe imprimir:

```
dfc{a -> d, bd -> c}
```

opcionalmente, se pueden imprimir estados intermedios que al grupo crea de utilidad.

Modalidad: El presente trabajo puede realizarse en grupos de 2 o 3 alumnos, y luego de la entrega, se debe realizar la defensa del mismo.

Fecha de entrega: 4/11/2023

Materiales:

Para la realización de dicho proyecto, se les proveerá a los alumnos lo siguiente:

- Ejemplos de uso de predicados de segundo orden.
- Ejemplos de uso de operadores.
- Uso de predicados de entrada/salida y dinámicos
- Predicados útiles: name/2, sort/2, union/3, append/3, assertz/3, write/1, writeln/1,

Trabajo realizar:

Forma de entrega y defensa:

La entrega del trabajo debe contar con:

- El código **Prolog** (archivo **.pl**) con los predicados necesarios para obtener.
- Ejemplos de prueba de cada predicado con salidas.
- Un texto con la descripción de las actividades realizadas.

La defensa será una entrevista virtual de 20 minutos por grupo.

Ayuda. Esto es sugerido, el grupo puede presentar otra solución.

- El conjunto de DF y atributos colocarlos en un hecho, preferentemente utilizando minúsculas para que no lo tome como variables. De usar mayúsculas, debe poner los atributos entre comillas.

  Ejemplo 1:

  db([a->bcaeh, ab->dhi, i->c, e->h, aje->ak, aij->k], abcdehijk).

  Notar que las dependencias están en una lista.

- Definir el operador ->:
- Pasar las dependencias a listas utilizando el predicado name/2. Este predicado retorna una lista con los códigos ASCII de cada letra:
  - ?- name(abc, L).
  - L = [97, 98, 99]

  Tener en cuenta que hace la opción inversa:

  - ?- name(A, [97, 98, 99]).
  - A = abc

  o bien,

  - ?- name(A, [a, b, c]).
  - A = abc

  O sea, se puede realizar el proceso usando el código ASCII y al imprimir se usa el proceso inverso, o transformar a cada letra antes (las dos opciones son válidas).

- Se debe trabajar el conjunto de atributos como una lista de letras (o códigos ASCII) y las dependencias como una lista de pares de listas. Teniendo el caso del ejemplo uno sería:
  - Atributos: [97, 98, 99, 100, 101, 104, 105, 106, 107] o [a,c,d,e,l,m,n,p,s]
  - Dependencias: [[[97], [98, 99, 97, 101, 104]], [[97, 98], [100, 104, 105]], [[105], [99]], [[101], [104]], [[97, 106, 101], [97, 107]], ...

    o con letras…

  - [[[a], [b, c, a, e, h]], [[a, b], [d, h, i]], [[i], [c]], [[e], [h]], [[a, j, e], [a, k]], ...

Se dará más ayudas o guías a medida que se avance en el desarrollo del trabajo.