CI-1441 Paradigmas computacionales

Grupo 01, I-2018, Prof. A. de la Ossa, Dr.rer.nat.

–Tarea programada 1 (Prolog)–

Objetivo

El objetivo principal de esta tarea es reforzar los conocimientos adquiridos en el curso relacionados con la programación lógica y el lenguaje PROLOG.

Instrucciones

La tarea consta de dos partes, con distinto peso en la nota: predicados para el procesamiento de árboles y conjuntos (40%) y la máquina encriptadora (60%).

La tarea debe hacerla en grupos de 2 o 3 estudiantes.

Haga sus **consultas** de la tarea al profesor, solamente **a través del foro del curso** en la plataforma de mediación virtual, de esa forma todos se enteran de las respuestas.

El momento límite de **entrega** de la tarea es el **viernes 16 de abril a la medianoche**, en la plataforma de mediación del curso. No se recibirán tareas por correo electrónico.

1. Predicados para el procesamiento de árboles y conjuntos

[40 puntos: 10 c/u]

Programe los siguientes predicados PROLOG de procesamiento de árboles y conjuntos.

Utilice `SWI-Prolog` (`swi-prolog.org`).

Debe entregar solamente un archivo `.pl` con el código y la documentación estándar (explicada en clase) de cada uno de los predicados programados, incluyendo cualquier predicado auxiliar.

bpp(+N,+A,-S): `S` es el subárbol de `A` cuya raíz es `N`; `S` debe ser NIL si el subárbol no existe

El predicado debe implementar el método de *búsqueda en profundidad primero*, y mostrar la secuencia de nodos visitados.

Validación: `N` debe ser un átomo y `A` un árbol.

Ejemplos:

- `bpp(d,[a,b,[c,d,e],[f,[g,h],i]],X)` → `X = d`; despliegue: `a b c d`
- `bpp(f,[a,b,[c,d,e],[f,[g,h],i]],X)` → `X =[f,[g,h],i]`; despliegue: `a b c d e f`
- `bpp(x,[a,b,[c,d,e],[f,[g,h],i]],X)` → `X = []`; despliegue: `a b c d e f g h i`

bap(+N,+A,-S): `S` es el subárbol de `A` cuya raíz es `N`, NIL si el subárbol no existe

El predicado debe implementar el método de *búsqueda en anchura primero*, y mostrar la secuencia de nodos visitados.

Validación: `N` debe ser un átomo y `A` un árbol.

Ejemplos:

- `bap(d,[a,b,[c,d,e],[f,[g,h],i]],X)` → `X = d`; despliegue: `a b c f d`
- `bap(f,[a,b,[c,d,e],[f,[g,h],i]],X)` → `X = [f,[g,h],i]`; despliegue: `a b c f`
- `bap(x,[a,b,[c,d,e],[f,[g,h],i]],X)` → X = []; despliegue: `a b c f d e g i h`

potencia(+C,-P): `P` es el conjunto potencia de `C`

El predicado debe implementar el cálculo del *conjunto potencia* de un conjunto.

Validación: `C` debe ser un conjunto.

Ejemplos:

- `potencia([a,b,c],X)` → `X = [nil,[a],[b],[c],[a,b],[a,c],[b,c],[a,b,c]]`
- `potencia(x,X)` → X = []; error: el argumento no es un conjunto

cartesiano(+A,+B,-C): `C` es el producto cartesiano de `A` y `B`

El predicado debe implementar el cálculo del *producto cartesiano* de dos conjuntos.

Validación: `A` y `B` deben ser conjuntos.

Ejemplos:

- `cartesiano([a,b,c],[d,e],X)` → `X = [[a,d],[a,e],[b,d],[b,e],[c,d],[c,e]]`
- `cartesiano(x,y,X)` → X = []; error: al menos uno de los argumentos no es un conjunto

2. La máquina encriptadora

[60 puntos: 45 `encripta`, 15 `decripta`]

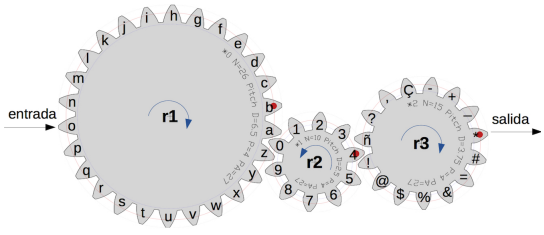
Construya una máquina *encriptadora*, es decir, una máquina que reciba de entrada una lista de símbolos y devuelva otra lista del mismo tamaño, que resulte de reemplazar cada símbolo de la primera (tomado de un alfabeto de *entrada*) con un símbolo de un alfabeto de *salida*, como se describe a continuación.

- `encripta(+He,+Ae,+As,-Hs,-Ef)`: `Hs` es el resultado de encriptar la hilera de entrada `He` con un engranaje formado por los alfabetos de entrada (`Ae`) y salida (`As`); `Ef` es el estado final de la máquina, formado por el par `[ae,as]`, donde `ae` y `as` son los símbolos de los alfabetos de entrada y salida, respectivamente, en los que quedó la máquina luego de encriptar `He`.

- El predicado inverso `decripta(+Hs,+Ae,+As,+Ef,-He)`, que decodifica la hilera encriptada `Hs` usando los alfabetos `Ae` y `As`, iniciando en la posición del engranaje descrita por el estado final de la máquina (`Ef`) cuando se encriptó la hilera que produjo la hilera `Hs`. `decripta` devuelve en `He` la hilera decodificada.

Para encriptar la hilera de entrada, su predicado `encripta/5` debe:

- Crear un *engranaje* con los alfabetos `Ae` de entrada y `As` de salida. La figura a continuación muestra un ejemplo de un engranaje de tres ruedas dentadas, cada una con sus dientes etiquetados con símbolos de un alfabeto específico. Su engranaje debe tener solo dos ruedas, una etiquetada con el alfabeto de entrada, y la otra con el alfabeto de salida.



- Como se muestra en la figura de ejemplo, se identifica una posición de la primera rueda como la `entrada` y una de la segunda como la `salida`. Cada vez que se rota una posición la rueda de entrada, la rueda de salida se rota también una posición pero en sentido contrario.

- Para cada símbolo de la lista de entrada:

  – Rotar la rueda de entrada hasta encontrar el símbolo de entrada. Esto hace que se mueva en sentido contrario la rueda de salida.

  – Anotar el símbolo en la posición de salida.

- Una vez que todos los símbolos de la lista de entrada han sido procesados, se devuelve la lista de símbolos de salida, y el par formado por los símbolos en las posiciones de entrada y salida del engranaje.

Note que para poder implementar las ruedas dentadas, es necesario contar con *listas circulares*, que son de fácil implementación en `Prolog`. Los predicados `circular` y `rotacircular` implementan una lista circular:

```
circular([a,b,c,d,e]).

rotacircular(L) :-
     retract(circular([Cabeza|Resto])),
     append(Resto,[Cabeza],L),
     assert(circular(L)).
```