Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Lenguajes de Programación (CI-3661)

Proyecto Prolog

Cocinando Sopas Atómicas

En la tierra de los mil árboles binarios, un grupo de aventureros (un computista, un físico, un químico y un biólogo) parten de la Ciudad Real de Ciencia con el propósito de despertar los cuatro structs que tienen consigo y así poder cumplir su destino de ser los elegidos por la Leyenda de los Cuatro Héroes.

Antes de comenzar a despertar los structs, el gran Rey de Ciencia le pide a los aventureros rescatar a su hija, raptada por el Lord y cuyo castillo está al norte de la Ciudad Real. Los aventureros, considerando que es una medida urgente, deciden partir al Castillo del Lord.

Al llegar al Castillo del Lord, se encuentran con el Guardián de la Puerta del Castillo, que evita su paso. Los aventureros deciden no atacarlo, ya que su nivel de poder sobrepasa 9000 y es imposible que puedan vencerlo. Intentando convencer al Guardián de que abandone el puesto, éste les dice su problema: ama las sopas de letras y todas las sopas que tenía a disposición las completó. No sería un problema exceptuando que a él le gustan ciertas palabras y desprecia otras palabras (todas las sopas de letras que el ha hecho tienen las palabras que ama y no tienen las palabras que desprecia).

Para resolver este problema, deciden dejar al computista (que es ud.) que resuelva el problema. Como no puede atacarlo, su otra opción es crear un programa que genere todas las posibles sopas de letras con las palabras que le gustan y que no tengan las que desprecia.

Y ahora para... completar... esto...

...VOLVIENDO A LA VIDA REAL...

Ud. debe implementar un programa, en Prolog, que genere todas las posibles sopas de letras que cumplan con lo indicado anteriormente.

1 Formalización del Problema

Como bien se presentó, el problema a atacar es una generalización de un rompecabeza conocido como la **sopa de letras**. Tal sopa de letras se define como una colección de letras, ofrecida en una malla cuadrángular de tamaño $n \times n$, como la presentada a continuación:

O G P S V R C W Ñ J
V A E Q J Ñ U A U W
G P L L F K V B T D
O L R N L N J H Y Y
J R K O A E Z A F U
H E U Q L Y K J V N
T P K A Y O Ñ S D A
P Y T H O N G Y A E
O J P D N Z Q F J H
O A Ñ X A V P O Ñ J

En esa colección de letras se encuentran esparcidas una colección de palabras, cada una ubicada en cualquier lado del tablero, siguiendo cualquiera de las 8 direcciones posibles (norte, noreste, este, sureste, sur, suroeste, oeste y noroeste). No existen palabras que se puedan conseguir en más de una dirección (por ejemplo, una palabra de 6 letras no puede tener 4 al norte y luego 2 al este, todas las 6 letras tienen que estar al norte o al este).

Por ejemplo, la sopa de letras indicada anteriormente tiene contenida las siguientes palabras (Y están invitados a resolver tal sopa de letras):

- HASKELL

- JAVA
- PERL
- PROLOG
- PYTHON
- RUBY

Por lo que se logra discernir en el enunciado motivante, el problema consiste en construir todas las posibles sopas de letras de un tamaño y un alfabeto determinado, donde todas las palabras de una lista dada deben aparecer, y además todas las palabras de otra lista dada no deben aparecer.

Su proyecto debe recibir las listas de palabras a aparecer (aceptadas) y de palabras sin aparecer (rechazadas) en dos archivos, cada uno conteniendo la lista en cuestión.

2 Estructuras de Datos a Utilizar

Para cada uno de los elementos discernidos, se usarán las siguientes estructuras de datos:

- El alfabeto es una lista de átomos de tamaño 1. Puede suponer que tales átomos tienen ese tamaño.
- La lista de palabras (tanto aceptadas como rechazadas) será una lista de átomos de tamaño mayor o igual a 1, donde cada caracter del átomo pertenece al alfabeto provisto.
- El tamaño (*tam*) es un número entero, sin límite alguno.
- La sopa de letras se representará con una lista de *tam* listas, cada una de *tam* átomos, donde cada átomo pertenece al alfabeto dado.

3 Predicados Recomendados a Implementar

Para este proyecto, se recomienda implementar los sigueintes predicados:

- El predicado `cargarListaPalabra`, que debe recibir el alfabeto a usar y la dirección de un archivo. La sintaxis del archivo es una lista de Prolog, con las palabras correspondientes (y debe finalizar con un punto). Este predicado debe satisfacerse si:
  - Los caracteres de todos los átomos pertenecen al alfabeto dado.
  - Se cargó la lista de palabras en el sistema. El almacenamiento de la lista de palabras en la aplicación se deja a discreción de ud.

  Es probable que, dependiendo de la decisión que ud. tomó para almacenar las palabras, deba implementar más de un predicado.
- El predicado `sopaLetra`, que debe construir todas las sopas de letras (una por una, vía *backtracking*), dado el alfabeto, el tamaño de la sopa de letras a construir y las palabras aceptadas y rechazadas (considerando la decisión que ud. tomó en el predicado anterior). Puede suponer que el alfabeto, el tamaño y las listas ya están instanciadas, mas la sopa de letras puede estar o no instanciadas.
- El predicado `mostrarSopa`, que recibe una sopa de letras ya instanciada y la imprime en pantalla. Dicha impresión debe ser legible para el usuario. Por ejemplo:

```
?- mostrarSopa([[a,b],[c,d]]).
 a b
 c d
```

4 Funcionalidad del Proyecto

Ud. debe implementar el predicado:

```
generadorSopa.
```

que debe funcionar de esta forma:

- Primero debe recibir, por entrada estandar, los siguientes datos:
  - El tamaño de la sopa de letras a generar (como un entero).
  - El alfabeto de la sopa de letras (como una lista de átomos).
  - Un nombre de archivo para la lista de palabras a aceptar. Si las palabras a aceptar no cumplen con el alfabeto dado, se debe detener el programa.
  - Un nombre de archivo para la lista de palabras a rechazar. Si las palabras a rechazar no cumplen con el alfabeto dado, se debe detener el programa.
- Después de recibidos los datos, el debe generar una sopa de letras y mostrarla en pantalla.
  - Si se desea generar más sopas de letras, se le da el comando `mas`.
  - Si no se desea generar más sopas, se le da cualquier otra instrucción y este terminará el proceso.

Como ejemplo, considere los siguientes archivos:

```
aceptados.txt:
[ha, sim].

rechazados.txt:
[lo, jo].
```

y se quieren generar sopas de letras de tamaño 3×3 con el alfabeto $\{h, a, s, i, m, l, o, j\}$. Entonces la siguiente es una traza del programa pidiendo dos sopas de letras distintas.

```
Tamaño? 3.
Alfabeto? [h, a, s, i, m, l, o, j].
Archivo de palabras aceptadas? 'aceptados.txt'.
Archivo de palabras rechazadas? 'rechazados.txt'.

h a s
l j i
m a m

Quieres mas? mas.

o a h
m i s
j l j

Quieres mas? no.

Gracias!
```

Algunos comentarios adicionales con respecto a su proyecto:

- No importa el orden en que se generen las distintas sopas de letras, siempre y cuando **se generen todas las posibles sopas**. Se pueden imprimir soluciones repetidas.
- Su implementación no puede usar el predicado `findall` (o similares) para obtener todas las sopas de letras. Debe de generarlas una por una vía *backtracking*, como fue dictado en clase.
- Su implementación debe correr en el modo intérprete de SWI-Prolog, por lo que no se necesita compilación alguna, ni un Makefile para compilarlo.

Detalles de Entrega

- Ud. debe entregar un archivo, llamado `sopaletra.pl`, con su implementación del proyecto, en adición a un `README` indicando el estado de su proyecto **(con ese nombre)**, en un archivo `.tar.gz` llamado `proy2-gXX.tar.gz`, donde `XX` corresponde a su número de grupo (con un 0 antecedido si el grupo es del 0 al 9). Por ejemplo, el grupo 00 debe entregar el archivo `proy2-g00.tar.gz`.

  - El contenido de su proyecto debe residir en un directorio llamado gXX, donde XX es el número de su grupo. Ud. debe comprimir el directorio. Si su proyecto su no cumple con esta norma, este no será evaluado y tendrá cero (0) puntos.

- Debe entregar el archivo respectivo a más tardar el día 22 de marzo a las 11.59 pm, tanto al prof. Pérez como al prep. Gómez. Cualquier entrega después de esa hora incurrirá en la no evaluación de su proyecto y tendrá una ponderación de cero (0) puntos.

2013. C. Pérez, M. Gómez