DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Paradigma Lógico.

Paradigmas de Lenguajes

Grupo Two and a Half Blondes

Integrante	LU	Correo electrónico
De Sousa Bispo, Germán	359/12	germandesousa@gmail.com
Fernandez, Esteban	691/12	esteban.pmf@gmail.com
Wright, Carolina	876/12	wright.carolina@gmail.com

Reservado para la cátedra

Instancia	Fecha	Docente	Nota
Primera entrega			
Segunda entrega			

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Aclaración

El ejercicio 8 posee errores a la hora de descifrar. Trae repetidos bajo ciertas condiciones y no siempre descifra bien. Se lo anexo para obtener un feedback al respecto para corregirlo posteriormente.

2. Código

2.1. Ejercicio 1

- diccionario_lista(?Lcode): la reversibilidad depende de la reversibilidad de string_codes, la cual es string_codes(?Strin ?Codes).

```
diccionario_lista(Lcode) :- diccionario(PalabraDelDicc),
                            string_codes(PalabraDelDicc, Lcode).
```

2.2. Ejercicio 2

- juntar_con(+S, ?J , ?R): Si no se instancia S, el predicado no funciona correctamente (aunque no se cuelga) por la forma en que se realizan los Appends. J puede ser una variable que, si R está instanciado, terminará unificándose. Si no, appendeará a la variable J.

```
juntar_con([L], _, L).
juntar_con([Ls | Lss], J, R) :- juntar_con(Lss, J, Rrec),
                                append(RPref, Rrec, R),
                                append(Ls, [J], RPref),!.
```

2.3. Ejercicio 3

- palabras(+S, ?P): Los motivos por los que S debe estar instanciado se muestran en el análisis de split_por_caracter. El mismo se encuentra en el ejercicio 10.

```
palabras(S, P) :- split_por_caracter(S, espacio, P).
```

2.4. Ejercicio 4

- asignar_var(?A, +MI, ?MF): Si MI no está instanciado, se cuelga cuando trata de encontrar otra solución más que MI = [] y MF = A, debido a que genera combinaciones con listas infinitas. A o MF deben estar instanciados. No pueden no estar instanciados al mismo tiempo.
- get_keys(?Tuples, ?TMapped): Si ambas no están instanciadas, genera infinitos resultados.
- Este ejercicio funciona gracias a la capacidad de ProLog de generar variables frescas bajo demanda utilizando variables anonimas generadas con la keyword "_", ademas, la manera de representar dichas variables frescas ayuda para que estas puedan ser manipulables dentro de los predicados. Si se nos diera como variable fresca una que ya hemos utilizado antes nuestro predicado asignar_var se volveria inconsistente.

```
asignar_var(A, MI, MF) :- get_keys(MI, Keys),
                          not(member(A, Keys)),
                          append(MI, [(A, _)], MF).
asignar_var(A, MI, MI) :- get_keys(MI, Keys),
                          member(A, Keys).

get_keys([], []).
get_keys([(Key, _) | Ts], TMapped) :- get_keys(Ts, Trec),
                                      append([Key], Trec, TMapped).
```

2.5. Ejercicio 5

- palabras_con_variables(+L, ?V): Si L no estuviera instanciado, no se rompe pero genera infinitos resultados inservibles con combinaciones de listas vacias. Esto sucede debido a la utilización de member().
- palabras_con_var_y_mapa(+L, ?V, +Mi, -Mf)
- pares_definidos_en_mapa(+L, ?V, +Mi, -Mf)

```
palabras_con_variables(L, V) :- palabras_con_var_y_mapa(L, V, [], _).

palabras_con_var_y_mapa([], [], Mf, Mf).
palabras_con_var_y_mapa([Ls | Lss], [Vs | Vss], M, Mf1) :-
                                pares_definidos_en_mapa(Ls, Vs, M, Mf0),
                                palabras_con_var_y_mapa(Lss, Vss, Mf0, Mf1).

pares_definidos_en_mapa([], [], Mf, Mf).
pares_definidos_en_mapa([X | Ls], [V | Vs], Mi0, Mf) :-
                                asignar_var(X, Mi0 , Mi1),
                                member((X,V), Mi1),
                                pares_definidos_en_mapa(Ls, Vs, Mi1, Mf).
```

2.6. Ejercicio 6

- quitar(?E, +Ls, ?R): E puede no estar instanciado ya que el algoritmo también elimina variables en caso de estar. Ls deberá estar instanciada para evitar la generación de listas infinitas. Si no esta instanciado, devuelve una solución trivial y luego se cuelga cuando se pide otra (debido al uso de append).

```
quitar(_, [], R) :- R = [].
quitar(E, [L|Ls], R) :- E == L, quitar(E, Ls, R).
quitar(E, [L|Ls], R) :- E \== L, quitar(E, Ls, Rrec),
                        append([L], Rrec, R).
```

2.7. Ejercicio 7

- cant_distintos(+Ls, ?S): Ls deberá estar instanciado para evitar la generación de listas infinitas. Si no esta instanciado, devuelve una solución trivial y luego se cuelga cuando se pide otra (debido al uso de quitar(), quien termina llamando a append()).

```
cant_distintos([], S) :- S = 0.
cant_distintos([L | Ls], S) :- quitar(L, Ls, RQuitado),
                               cant_distintos(RQuitado, CuentaRec),
                               S is (1 + CuentaRec).
```

2.8. Ejercicio 8

- Aclaración: este ejercicio devuelve incorrectamente repetidos y soluciones inválidas.

```
descifrar(S, M) :- palabras(S, P),
                   palabras_con_variables(P, V),
                   descifrarPalabras(V, Mvar),
                   juntar_con(Mvar, 32, PalabrasSeparadas),
                   string_codes(M, PalabrasSeparadas).

descifrarPalabras([], []).
descifrarPalabras([Vs | Vss], Mvar) :- diccionario_lista(PalabraDelDicc),
```

```
                                        palabra_valida(Vs, PalabraDelDicc, Mp),
                                        descifrarPalabras(Vss, Mrec),
                                        append([Mp], Mrec, Mvar).

palabra_valida([], [], []).
palabra_valida([Var | Vars], [P | Ps], M) :- length(Vars, Lv),
                                              length(Ps, Lp),
                                              Lv == Lp,
                                              palabra_valida(Vars, Ps, Mrec),
                                              esta_libre(Var, P, Ps),
                                              P = Var,
                                              append([P], Mrec, M).

esta_libre(_, _, []).
esta_libre(Var, _, _) :- nonvar(Var).
esta_libre(Var, P, [Pp | Ps]) :- var(Var), Pp \== P, esta_libre(Var, P, Ps).
```

2.9. Ejercicio 9

- descifrar_sin_espacios(+S, ?M), necesariamente S debe estar instanciado para generar las posibles intercalaciones con espacio resultantes, que luego deberan ser descifradas. Si no estuviera instanciada la S podria instanciarse en secuencias, potencialmente infinitas, que nunca daran una oracion valida considerando el diccionario cargado actual.
- con_espacios_intercalados(+S, ?R)
- intercalar_o_no(+P, +S, ?R)

```
descifrar_sin_espacios(S, M) :- con_espacios_intercalados(S, SWithSpaces),
                                descifrar(SWithSpaces, M).

con_espacios_intercalados([], []).
con_espacios_intercalados(S, SWithSpaces) :-
                        append(SPref, SSuf, S), SPref \== [],
                        intercalar_o_no(SPref, SSuf, SPrefIntercalado),
                        con_espacios_intercalados(SSuf, SWithSpacesSuf),
                        append(SPrefIntercalado, SWithSpacesSuf, SWithSpaces).

intercalar_o_no(Pref, [], Pref) :- !.
intercalar_o_no(Pref, _, PrefNuevo) :- append(Pref, [espacio], PrefNuevo).
```

2.10. Ejercicio 10

- mensajes_mas_parejos(+S, ?M): S debe estar instanciada en particular, por el uso de descifrar_sin_espacios().
- hay_uno_menor(+DesvioM, +S).
- calcular_desvio(+Mensaje, ?Desvio).
- split_por_caracter(?Sentencia, +Caracter, ?ListaDePalabras): En caso de que ListaDePalabras esté instanciado, Sentencia debe estar instanciado.
- leer_hasta_caracter(+Caracteres, +CaracterSeparador, ?Palabra).
- palabra_hasta_caracter(+Cs, +CaracterSeparador, ?Accum, ?Palabra): En caso de instanciarse Accum, debe tener relacion con lo puesto en Cs para que el algoritmo tenga sentido. Hay casos para los que funciona tener ?Cs, pero en otros se cuelga. Al igual que para Accum, esos casos deben tener sentido en el algoritmo. No se recomienda.

- borrar_hasta_caracter(?Palabra, ?Caracter, ?Sentencia, ?SentenciaSinPalabra): Siempre brinda una sola solución.
- calcular_desvio_sobre_lista_de_palabras(+Palabras, ?Desvio).
- calcular_longitud_media(+P, ?LongMedia).
- length_list(+Ls, ?LList).
- average(+List, ?Average).
- binomios_cuadrados(+Ps, +LongMedia, -BCuadrados).
- binomio_cuadrado(?P, +LongMedia, -BCuadrado).

```
mensajes_mas_parejos(S, M) :- descifrar_sin_espacios(S, M),
                              string_codes(M, L),
                              calcular_desvio(L, DesvioM),
                              not(hay_uno_menor(DesvioM, S)).

hay_uno_menor(DesvioM, S) :- descifrar_sin_espacios(S, MComparador),
                             string_codes(MComparador, LComparador),
                             calcular_desvio(LComparador, DesvioComp),
                             DesvioM > DesvioComp.

calcular_desvio(Mensaje, Desvio) :-
        split_por_caracter(Mensaje, 32, MsjSeparadosPorEspacio),
        calcular_desvio_sobre_lista_de_palabras(MsjSeparadosPorEspacio, Desvio).

split_por_caracter([], _, []).
split_por_caracter(Sentencia, Caracter, ListaDePalabras) :-
                leer_hasta_caracter(Sentencia, Caracter, Palabra),
                borrar_hasta_caracter(Palabra, Caracter, Sentencia, SSinPalabra),
                split_por_caracter(SSinPalabra, Caracter, RecListaDePalabras),
                append([Palabra], RecListaDePalabras, ListaDePalabras), !.

leer_hasta_caracter(Caracteres, CaracterSeparador, Palabra) :-
                palabra_hasta_caracter(Caracteres, CaracterSeparador, [], Palabra).

palabra_hasta_caracter([], _, Palabra, Palabra).
palabra_hasta_caracter([C|Cs], CaracterSeparador, Accum, Palabra) :-
        C\=CaracterSeparador,
        append(Accum, [C], AccumConCaracter),
        palabra_hasta_caracter(Cs, CaracterSeparador, AccumConCaracter, Palabra).
palabra_hasta_caracter([C|_], CaracterSeparador, Palabra, Palabra) :-
        C==CaracterSeparador.

borrar_hasta_caracter(Palabra, Caracter, Sentencia, SentenciaSinPalabra) :-
                append(Palabra, [Caracter|SentenciaSinPalabra], Sentencia), !.
borrar_hasta_caracter(Palabra, _, Sentencia, SentenciaSinPalabra) :-
                append(Palabra, SentenciaSinPalabra, Sentencia).

calcular_desvio_sobre_lista_de_palabras(Palabras, Desvio) :-
                calcular_longitud_media(Palabras, LongMedia),
                binomios_cuadrados(Palabras, LongMedia, BCuadrados),
                sum_list(BCuadrados, Sumatoria),
                length(Palabras, LPalabras), Division is Sumatoria / LPalabras,
                Desvio is sqrt(Division).

calcular_longitud_media(P, LongMedia) :- length_list(P, LengthList),
                                         average(LengthList, LongMedia).
```

```
length_list([L | Ls], LList) :- append([LLength], LListRec, LList),
                                length_list(Ls, LListRec),
                                length(L, LLength),!.
length_list([L], LList) :- length(L, LLength), LList = [LLength].

average(List, Average):- sum_list(List, Sum),
                         length(List, Length),
                         Length > 0,
                         Average is (Sum / Length).


binomios_cuadrados([P], LongMedia, [BCuadrado]) :-
                binomio_cuadrado(P, LongMedia, BCuadrado).
binomios_cuadrados([P | Ps], LongMedia, BCuadrados) :-
                append([BCuadrado], RecBCuadrados, BCuadrados),
                binomios_cuadrados(Ps, LongMedia, RecBCuadrados),
                binomio_cuadrado(P, LongMedia, BCuadrado),!.

binomio_cuadrado(P, LongMedia, BCuadrado) :- length(P, LongP),
                                             Resta is (LongP-LongMedia),
                                             BCuadrado is (Resta^2).
```