Practica 2 - COMPRESION DE SECUENCIAS

ISO-Prolog

David de Frutos, B190372

Table of Contents

codigo

Este modulo define los predicados de la practica 2 para la que se quiere realizar la compresion de una secuencia de caracter

Usage and interface

- **Library usage:**
  `:- use_module(/home/defru/prolog2/codigo.pl).`
- **Exports:**
  - *Predicates:*
    `memo/2`, `comprimir/2`, `limpia_memo/0`, `compresion_recursiva/2`, `partir/3`, `parentesis/3`, `se_repite/4`, `repeticion/2`, `compresion/2`, `division/2`, `mejor_compresion/2`, `sol_optima/2`, `mejor_compresion_memo/2`.
  - *Properties:*
    `alumno_prode/4`.
  - *Multifiles:*
    `Σcall_in_module/2`.

Documentation on exports

memo/2: PREDICATE

No further documentation available for this predicate. The predicate is of type *dynamic*.

alumno_prode/4: PROPERTY

Usage:

Datos del alumno que realiza la entrega.

```
alumno_prode('De Frutos','Zafra','David','B190372').
```

comprimir/2: PREDICATE

Usage: `comprimir(Inicial,Comprimida)`

donde `Inicial` es la secuencia original de caracteres y `Comprimida` es el resultado de aplicar las operaciones necesarias a la cadena `Inicial` para comprimirla.

```
comprimir(Inicial,Comprimida) :-
    limpia_memo,
    compresion_recursiva(Inicial,Comprimida).
```

limpia_memo/0: PREDICATE

Usage:

es el encargado de limpiar los lemas almacenados en la memoria.

```
limpia_memo :-
    retractall(memo(_1,_2)).
```

compresion_recursiva/2: PREDICATE

Usage: `compresion_recursiva(Inicial,Comprimida)`

es el predicado al que se llama para realizar la compresion de manera recursiva.

```
compresion_recursiva(Inicial,Comprimida) :-
    mejor_compresion_memo(Inicial,Comprimida).
```

partir/3: PREDICATE

Usage: `partir(Todo,Parte1,Parte2)`

este predicado se verifica si `Parte1` y `Parte2` son dos subsecuencias no vacias que concatenadas forman la secuencia `Todo`.

```
partir(Todo,Parte1,Parte2) :-
    append(Parte1,Parte2,Todo),
    length(Parte1,N1),
    length(Parte2,N2),
    N1>0,
    N2>0.
```

Other properties:

Test: `partir(Todo,Parte1,Parte2)`

Prueba 1 - Se divide en dos listas no vacias que son subsecuencias.

– *If the following properties hold at call time:*

`Todo=[1,2,3,4]` (`= /2`)

then the following properties should hold upon exit:

`Parte1=[1],Parte2=[2,3,4];Parte1=[1,2],Parte2=[3,4]` (undefined property)

then the following properties should hold globally:

All the calls of the form `partir(Todo,Parte1,Parte2)` do not fail. (`not_fails/1`)

parentesis/3: PREDICATE

Usage: `parentesis(Parte,Num,ParteNum)`

donde `ParteNum` es la lista de caracteres resultado de componer la lista `Parte` con el numero de repeticiones `Num`, aadiendo parentesis solamente si `Parte` tiene 2 elementos o mas.

```
parentesis(Parte,Num,ParteNum) :-
    integer(Num),
    length(Parte,N1),
    ( N1<2 ->
        append(Parte,[Num],ParteNum)
    ; append(['('|Parte],[')',Num],ParteNum)
    ),
    !.
```

Other properties:

Test: `parentesis(Parte,Num,ParteNum)`

Prueba 2 - Parte contiene 3 elementos.

– *If the following properties hold at call time:*

`Parte=[a,b,c]` (= /2)

`Num=3` (= /2)

then the following properties should hold upon exit:

`ParteNum=[(,a,b,c,),3]` (= /2)

then the following properties should hold globally:

All the calls of the form `parentesis(Parte,Num,ParteNum)` do not fail. (`not_fails/1`)

Test: `parentesis(Parte,Num,ParteNum)`

Prueba 3 - Parte contiene 1 elemento.

– *If the following properties hold at call time:*

`Parte=[a]` (= /2)

`Num=2` (= /2)

then the following properties should hold upon exit:

`ParteNum=[a,2]` (= /2)

then the following properties should hold globally:

All the calls of the form `parentesis(Parte,Num,ParteNum)` do not fail. (`not_fails/1`)

se_repite/4: PREDICATE

Usage: `se_repite(Cs,Parte,Num0,Num)`

tiene exito si `Cs` se obtiene por repetir N veces la secuencia `Parte`. El argumento `Num` incrementa `Num0` en N.

```
se_repite([],_1,Num0,Num0).
se_repite(Cs,Cs,Num0,Num) :-
    Num is Num0+1,
    !.
se_repite(Cs,Parte,Num0,Num) :-
    append(Parte,Rest,Cs),
    se_repite(Rest,Parte,Num0,N),
    Num is N+1,
    !.
```

Other properties:

Test: `se_repite(Cs,Parte,Num0,Num)`

Prueba 4 - [a,b,c] se repite una vez en [a,b,c].

– *If the following properties hold at call time:*

`Cs=[a,b,c]` (= /2)

`Parte=[a,b,c]` (= /2)

`Num0=0` (= /2)

then the following properties should hold upon exit:

`R=1` (= /2)

then the following properties should hold globally:

All the calls of the form `se_repite(Cs,Parte,Num0,Num)` do not fail. (`not_fails/1`)

Test: `se_repite(Cs,Parte,Num0,Num)`

Prueba 5 - [a,b,c,a,b,c,a,b,c] se repite tres veces en [a,b,c].

– *If the following properties hold at call time:*

`Cs=[a,b,c,a,b,c,a,b,c]` (`= /2`)

`Parte=[a,b,c]` (`= /2`)

`Num0=0` (`= /2`)

then the following properties should hold upon exit:

`R=3` (`= /2`)

then the following properties should hold globally:

All the calls of the form `se_repite(Cs,Parte,Num0,Num)` do not fail. (`not_fails/1`)

Test: `se_repite(Cs,Parte,Num0,Num)`

Prueba 6 - [a,b,c,a,c,a,b,c] no se puede obtener repitiendo [a,b,c].

– *If the following properties hold at call time:*

`Cs=[a,b,c,a,c,a,b,c]` (`= /2`)

`Parte=[a,b,c]` (`= /2`)

`Num0=0` (`= /2`)

then the following properties should hold upon exit:

`R=0` (`= /2`)

then the following properties should hold globally:

Calls of the form `se_repite(Cs,Parte,Num0,Num)` fail. (`fails/1`)

Test: `se_repite(Cs,Parte,Num0,Num)`

Prueba 7 - [] se obtiene repitiendo 0 veces [a,b,c].

– *If the following properties hold at call time:*

`Cs=[]` (`= /2`)

`Parte=[a,b,c]` (`= /2`)

`Num0=0` (`= /2`)

then the following properties should hold upon exit:

`R=0` (`= /2`)

then the following properties should hold globally:

All the calls of the form `se_repite(Cs,Parte,Num0,Num)` do not fail. (`not_fails/1`)

repeticion/2: PREDICATE

Usage: `repeticion(Inicial,Comprimida)`

basandose en los predicados partir/3 y se_repite/4 debe identificar un prefijo (o parte) que por repeticion permita obtener la secuencia inicial. Esta parte sera posteriormente comprimida.

```
repeticion(Inicial,ParteNum) :-
    partir(Inicial,Parte1,_Parte2),
    se_repite(Inicial,Parte1,0,Num),
    compresion_recursiva(Parte1,Comprimida),
    parentesis(Comprimida,Num,ParteNum),
    !.
```

Other properties:

Test: `repeticion(Inicial,ParteNum)`

Prueba 8 - [a,a,a,a,a,a,a] se obtiene repitiendo 7 veces [a].

- *If the following properties hold at call time:*
  `Inicial=[a,a,a,a,a,a,a]` (= /2)
  then the following properties should hold upon exit:
  `R=[a,7]` (= /2)
  then the following properties should hold globally:
  All the calls of the form `repeticion(Inicial,ParteNum)` do not fail. (`not_fails/1`)

Test: `repeticion(Inicial,ParteNum)`

Prueba 9 - [a,b,a,b,a,b] se obtiene repitiendo 3 veces [a,b].

- *If the following properties hold at call time:*
  `Inicial=[a,b,a,b,a,b]` (= /2)
  then the following properties should hold upon exit:
  `R=[(,a,b,),3]` (= /2)
  then the following properties should hold globally:
  All the calls of the form `repeticion(Inicial,ParteNum)` do not fail. (`not_fails/1`)

Test: `repeticion(Inicial,ParteNum)`

Prueba 10 - [a,b,a,b,a] no se puede obtener repitiendo ninguna secuencia.

- *If the following properties hold at call time:*
  `Inicial=[a,b,a,b,a]` (= /2)
  then the following properties should hold upon exit:
  `R=0` (= /2)
  then the following properties should hold globally:
  Calls of the form `repeticion(Inicial,ParteNum)` fail. (`fails/1`)

compresion/2: PREDICATE

Usage: `compresion(Inicial,Comprimida)`

tendra dos alternativas: llamar a repeticion/2 o a division/2 para obtener todas las posibles compresiones por backtracking, tanto optimas como no optimas.

```
compresion(Inicial,Comprimida) :-
    repeticion(Inicial,Comprimida).
compresion(Inicial,Comprimida) :-
    division(Inicial,Comprimida).
```

division/2: PREDICATE

Usage: `division(Inicial,Comprimida)`

debe partir la lista inicial en dos partes y llamar a compresion_recursiva/2 de forma recursivas con cada una de ellas para finalmente concatenar los resultados. Esto dara mas posibilidad a encontrar repeticiones

```
division(Inicial,Final) :-
    partir(Inicial,Parte1,Parte2),
    compresion_recursiva(Parte1,Comprimida1),
    compresion_recursiva(Parte2,Comprimida2),
    append(Comprimida1,Comprimida2,Final).
```

mejor_compresion/2: PREDICATE

Usage: `mejor_compresion(Inicial,Comprimida)`

intentara encontrar compresiones que reduzcan el tamano, es decir, el resultado final de `Comprimida` sera la que tenga el menor tamao, la solucion mas optima

```
mejor_compresion([X],[X]).
mejor_compresion(Inicial,Comprimida) :-
    findall(LComp,compresion(Inicial,LComp),LAcum),
    sol_optima(LAcum,Comprimida).
```

Other properties:

Test: `mejor_compresion(Inicial,Comprimida)`

Prueba 11 - se obtiene la mejor compresion posible de [a,b,a,b,a,b].

- *If the following properties hold at call time:*

  `Inicial=[a,b,a,b,a,b]` (`= /2`)

  then the following properties should hold upon exit:

  `Comprimida=[(,a,b,),3]` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `mejor_compresion(Inicial,Comprimida)` do not fail. (`not_fails/1`)

sol_optima/2: PREDICATE

Usage: `sol_optima(Posible_sol,Mejor)`

debe encontrar la solucion con la menor longitud entre una serie de soluciones obtenidas con la llamada al predicado findall `Posible_sol` es la lista en la que se han ido almacenando las diferentes listas comprimidas

```
sol_optima([X],X).
sol_optima([X|Xs],Mejor) :-
    sol_optima(Xs,Mejor1),
    length(X,N1),
    length(Mejor1,N2),
    ( N1<N2 ->
        Mejor=X
    ; Mejor=Mejor1
    ).
```

Other properties:

Test: `sol_optima(Posible_sol,Mejor)`

Prueba 12 - La solucion optima entre las posibles soluciones ([a,b]) es [a,b].

- *If the following properties hold at call time:*

  `Posible_sol=[[a,b]]` (`= /2`)

  then the following properties should hold upon exit:

  `Mejor=[a,b]` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `sol_optima(Posible_sol,Mejor)` do not fail. (`not_fails/1`)

Test: `sol_optima(Posible_sol,Mejor)`

Prueba 13 - La solucion optima entre las posibles soluciones ([[a,b],[a,b,c],[a,b,c,d],[a]) es [a].

– *If the following properties hold at call time:*

`Posible_sol=[[a,b],[a,b,c],[a,b,c,d],[a]]` (`= /2`)

then the following properties should hold upon exit:

`Mejor=[a]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `sol_optima(Posible_sol,Mejor)` do not fail. (`not_fails/1`)

mejor_compresion_memo/2: PREDICATE

Usage: `mejor_compresion_memo(Inicial,Comprimida)`

debe obtener la compresion mas optima posible, utilizando la memorizacion de lemas, lo que hara esta solucion aun mas eficiente al poder recordar aquellas que ya ha comprobado

```
mejor_compresion_memo(Inicial,Comprimida) :-
    memo(Inicial,Comprimida),
    !.
mejor_compresion_memo(Inicial,Comprimida) :-
    mejor_compresion(Inicial,Comprimida),
    assert(memo(Inicial,Comprimida)).
```

Documentation on multifiles

Σcall_in_module/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

Documentation on imports

This module has the following direct dependencies:

– *Application modules:*

`operators`, `dcg_phrase_rt`, `datafacts_rt`, `dynamic_rt`, `classic_predicates`.

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `hiord_rt`, `stream_basic`, `io_basic`, `runtime_control`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `classic`, `runtime_ops`, `dcg`, `dcg/dcg_phrase`, `dynamic`, `datafacts`, `assertions`, `assertions/assertions_basic`, `regtypes`.

References

(this section is empty)