Compresión de secuencias

Xiao Peng Ye, A180321

Table of Contents

codigo

Usage and interface

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

Documentation on exports

alumno_prode/4: PREDICATE

No further documentation available for this predicate.

comprimir/2: PREDICATE

Usage: `comprimir(Inicial,Comprimida)`

`Comprimida` es el resultado de comprimir `Inicial`, una lista de caracteres, utilizado la técnica de memoria dinámica.

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

Other properties:

Test: `comprimir(Inicial,Comprimida)`

- *If the following properties hold at call time:*
  `Inicial=[a,a,a,a,a,b,b,b,b]` (`= /2`)
  `Comprimida=[a,5,b,4]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `comprimir(Inicial,Comprimida)` do not fail. (`not_fails/1`)

Test: `comprimir(Inicial,Comprimida)`

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

Test: `comprimir(Inicial,Comprimida)`

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

`Inicial=[97,98,99,97,98,99]` (`= /2`)

`Comprimida=[97,98,99,97,98,99]` (`= /2`)

then the following properties should hold globally:

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

memo/2: PREDICATE

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

compresion_recursiva/2: PREDICATE

Usage: `compresion_recursiva(Inicial,Comprimida)`

Predicado auxiliar de compresion.

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

limpia_memo/0: PREDICATE

Usage:

Borrar todos los predicados de memo de la memoria dinámica.

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

minimo_lista/2: PREDICATE

Usage: `minimo_lista(Lista,Minimo)`

`Minimo` es la lista con menor longitud de la lista de listas `Lista`.

```
minimo_lista(Lista,Minimo) :-
    member(Minimo,Lista),
    length(Minimo,N),
    \+ (member(X,Lista),length(X,M),N>M),
    !.
```

Other properties:

Test: `minimo_lista(Lista,Minimo)`

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

`Lista=[[49],[50,50],[51,51,51],[52,52,52,52]]` (`= /2`)

then the following properties should hold upon exit:

`Minimo=[49]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `minimo_lista(Lista,Minimo)` do not fail. (`not_fails/1`)

Test: `minimo_lista(Lista,Minimo)`

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

`Lista=[[50,50],[50,50],[51,51,51],[52,52,52,52],[52,52,52,52]]` (`= /2`)

then the following properties should hold upon exit:

`Minimo=[50,50]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `minimo_lista(Lista,Minimo)` do not fail. (`not_fails/1`)

mejor_compresion/2: PREDICATE

Usage: `mejor_compresion(Inicial,Comprimida)`

Versión mejorado del predicado compresión utilizando findall para obtener la lista de todos las posibles sentencias comprimidas quedando solo con la de longitud menor.

```
mejor_compresion(Inicial,Comprimida) :-
    findall(X,compresion(Inicial,X),L),
    minimo_lista(L,Comprimida).
```

Other properties:

Test: `mejor_compresion(Inicial,Comprimida)`

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

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

`Comprimida=[(,a,4,b,3,c,4,),2]` (`= /2`)

then the following properties should hold globally:

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

mejor_compresion_memo/2: PREDICATE

Usage: `mejor_compresion_memo(Inicial,Comprimida)`

Predicado auxiliar para salvar las sentencias comprimidas ya conocidas en la memoria dinámica, evitando así duplicar el trabajo.

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

partir/3: PREDICATE

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

`Todo` es la lista formado al concatenar las listas no vacias `Parte1` y `Parte2`.

```
partir(Todo,Parte1,Parte2) :-
    append(Parte1,Parte2,Todo),
    Parte1\=[],
    Parte2\=[].
```

Other properties:

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

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

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

`Parte1=[]` (`= /2`)

`Parte2=[a,b,c]` (`= /2`)

then the following properties should hold globally:

Calls of the form `partir(Todo,Parte1,Parte2)` fail. (`fails/1`)

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

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

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

`Parte1=[a,b]` (`= /2`)

then the following properties should hold upon exit:

`Parte2=[c,d,e]` (`= /2`)

then the following properties should hold globally:

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

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

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

`Parte1=[104,111,108,97]` (`= /2`)

`Parte2=[32,109,117,110,100,111]` (`= /2`)

then the following properties should hold upon exit:

`Todo=[104,111,108,97,32,109,117,110,100,111]` (`= /2`)

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,ParteNumm)`

`ParteNum` es la lista de caracteres que es el resultado de componer la lista `Parte` con el número de repeticiones `Num`, anadiendo paréntesis solo si `Parte` tiene 2 elementos o más.

```
parentesis([X],Num,[X,Num]) :-
    number(Num).
parentesis(Parte,Num,ParteNum) :-
    number(Num),
    append(['('|Parte],[')',Num],ParteNum),
    length(Parte,N),
    N>1.
```

Other properties:

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

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

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

`Num=3` (`= /2`)

then the following properties should hold upon exit:

`ParteNum=[a,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)`

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

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

`Num=9` (`= /2`)

then the following properties should hold upon exit:

`ParteNum=[(,a,b,c,),9]` (`= /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)`

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

`Num=4` (`= /2`)

`ParteNum=[(,t,e,s,t,),4]` (`= /2`)

then the following properties should hold upon exit:

`Parte=[t,e,s,t]` (`= /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)`

La lista `Cs` es el resultado de repetir `Num - Num0` veces la lista `Parte`.

```
se_repite([],_1,Num0,Num) :-
    Num is Num0.
se_repite(Cs1,Parte,Num0,Num1) :-
    append(Parte,Cs,Cs1),
    se_repite(Cs,Parte,Num0,Num),
    Num1 is Num+1.
```

Other properties:

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

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

`Cs=[97,98,99,97,98,99,97,98,99,97,98,99]` (`= /2`)

`Parte=[97,98,99]` (`= /2`)

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

then the following properties should hold upon exit:

`Num=4` (`= /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)`

– *If the following properties hold at call time:*
`Cs=[97,98,99,97,99,97,98,99]` (`= /2`)
`Parte=[97,98,99]` (`= /2`)
`Num0=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)`
– *If the following properties hold at call time:*
`Cs=[]` (`= /2`)
`Parte=[118,97,99,105,111]` (`= /2`)
`Num0=0` (`= /2`)
then the following properties should hold upon exit:
`Num=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)`

La sentencia `Comprimida` es el resultado de comprimir la sentencia `Inicial` por la repetición de su subsecuencia.

```
repeticion(Inicial,Comprimida) :-
    partir(Inicial,Parte,_1),
    se_repite(Inicial,Parte,0,Num),
    compresion_recursiva(Parte,ParteComprimida),
    parentesis(ParteComprimida,Num,Comprimida).
```

Other properties:

Test: `repeticion(Inicial,Comprimida)`
– *If the following properties hold at call time:*
`Inicial=[a,a,a,a,a,a,a,a,a,a]` (`= /2`)
`Comprimida=[a,10]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `repeticion(Inicial,Comprimida)` do not fail. (`not_fails/1`)

Test: `repeticion(Inicial,Comprimida)`
– *If the following properties hold at call time:*
`Inicial=[a,b,a,b,a,b,a,b,a,b]` (`= /2`)
`Comprimida=[(,a,b,),5]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `repeticion(Inicial,Comprimida)` do not fail. (`not_fails/1`)

Test: `repeticion(Inicial,Comprimida)`
– *If the following properties hold at call time:*
`Inicial=[115,97,102,106,102,106,101,114,119]` (`= /2`)
then the following properties should hold globally:
Calls of the form `repeticion(Inicial,Comprimida)` fail. (`fails/1`)

division/2: PREDICATE

Usage: `division(Incial,Comprimida)`

La sentencia `Comprimida` es el resultado de comprimir la sentencia `Inicial` por división.

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

Other properties:

Test: `division(Inicial,Comprimida)`

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

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

`Comprimida=[a,4,b,5]` (`= /2`)

then the following properties should hold globally:

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

Test: `division(Inicial,Comprimida)`

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

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

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

then the following properties should hold globally:

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

compresion/2: PREDICATE

Usage: `compresion(Inicial,Comprimida)`

Devuelve todos las posibles sentencias comprimidas del `Inicial` tanto por repeticion como división.

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