manual

Este módulo permite la representación de números en formato binario o hexadecimal, así como realizar determinadas operaciones sobre estos.

Los números serán representados de la forma: bind(0), hexd(0).

Ejemplos de uso:

1. Conversión de un número en formato hexadecimal a formato binario:

```
?- byte_conversion([hexd(a), hexd(1)], B).

B = [bind(1), bind(0), bind(1), bind(0), bind(0), bind(0),
bind(0), bind(1)] ? ;

no
?-
```

2. Obtención del enésimo bit de un byte:

```
?- get_nth_bit_from_byte(s(s(s(s(s(0))))), [hexd(a), hexd(c)],
B).

B = bind(1) ? ;

no
?-
```

Usage and interface

Library usage:
`:- use_module(/home/mlg/manual.pl).`
Exports:

- *Predicates:*
  `byte_list/1`, `byte_conversion/2`, `juntar_listas/3`, `byte_list_conversion/2`, `get_nth_bit_from_byte/3`, `resta/3`, `sacar_bit/3`, `byte_list_clsh/2`, `unificar/2`, `trasponer_izq/2`, `restaurar/3`, `longitud_lista/2`, `byte_list_conversion_2/2`, `byte_conversion_2/2`, `separar_lista/4`, `byte_list_crsh/2`, `trasponer_drch/3`, `byte_xor/3`, `operacion_xor/3`.
- *Properties:*
  `bind/1`, `hexd/1`, `binary_byte/1`, `hex_byte/1`, `byte/1`, `hex_equiv/2`, `igual/2`, `no_igual/2`.

Documentation on exports

PROPERTY bind/1

Usage: `bind(A)`

`A` es un digito binario.

```
bind(0).
bind(1).
```

PROPERTY hexd/1

Usage: `hexd(A)`

`A` es un digito hexadecimal.

```
hexd(0).
hexd(1).
hexd(2).
hexd(3).
hexd(4).
hexd(5).
hexd(6).
hexd(7).
hexd(8).
hexd(9).
hexd(a).
hexd(b).
hexd(c).
hexd(d).
hexd(e).
hexd(f).
```

PROPERTY binary_byte/1

Usage: `binary_byte(A)`

`A` es un byte binario.

```
binary_byte([bind(B7),bind(B6),bind(B5),bind(B4),bind(B3),bind(B2),bind(B1),bind(B0)]
    bind(B7),
    bind(B6),
    bind(B5),
    bind(B4),
    bind(B3),
    bind(B2),
    bind(B1),
    bind(B0).
```

PROPERTY hex_byte/1

Usage: `hex_byte(A)`

`A` es un byte hexadecimal.

```
hex_byte([hexd(H1),hexd(H0)]) :-
    hexd(H1),
    hexd(H0).
```

PROPERTY byte/1

Usage: `byte(A)`

`A` es un byte deciaml o hexadecimal.

```
byte(BB) :-
    binary_byte(BB).
byte(HB) :-
    hex_byte(HB).
```

PREDICATE byte_list/1

Usage: `byte_list(A)`

`A` es una lista de bytes binarios o hexadecimales.

```
byte_list([]).
byte_list([X|Xs]) :-
    byte(X),
    byte_list(Xs).
```

Other properties:

Test: `byte_list(A)`

- *If the following properties hold at call time:*
  `A=[[bind(0),bind(0),bind(0),bind(0),bind(0),bind(0),bind(0),bind(0)], [bind(0),bind(0),bind(0),bind(0),bind(0),bind(0),bind(0),bind(0)]]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `byte_list(A)` do not fail. (`not_fails/1`)

PREDICATE byte_conversion/2

Usage: `byte_conversion(A,B)`

`A` es un byte hexadecimal y `B` es el byte binario equivalente.

```
byte_conversion([X,Y],Z) :-
    hex_equiv(X,A),
    hex_equiv(Y,B),
    juntar_listas(A,B,Z).
```

Other properties:

Test: `byte_conversion(A,B)`

- *If the following properties hold at call time:*
  `A=[hexd(a),hexd(1)]` (`= /2`)
  then the following properties should hold upon exit:
  `B=[bind(1),bind(0),bind(1),bind(0),bind(0),bind(0),bind(0),bind(1)]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `byte_conversion(A,B)` do not fail. (`not_fails/1`)

PROPERTY hex_equiv/2

Usage: `hex_equiv(A,B)`

`A` es un bit hexadecimal y `B` es el bit binario equivalente.

```
hex_equiv(hexd(0),[bind(0),bind(0),bind(0),bind(0)]).
hex_equiv(hexd(1),[bind(0),bind(0),bind(0),bind(1)]).
hex_equiv(hexd(2),[bind(0),bind(0),bind(1),bind(0)]).
hex_equiv(hexd(3),[bind(0),bind(0),bind(1),bind(1)]).
hex_equiv(hexd(4),[bind(0),bind(1),bind(0),bind(0)]).
hex_equiv(hexd(5),[bind(0),bind(1),bind(0),bind(1)]).
hex_equiv(hexd(6),[bind(0),bind(1),bind(1),bind(0)]).
hex_equiv(hexd(7),[bind(0),bind(1),bind(1),bind(1)]).
hex_equiv(hexd(8),[bind(1),bind(0),bind(0),bind(0)]).
hex_equiv(hexd(9),[bind(1),bind(0),bind(0),bind(1)]).
hex_equiv(hexd(a),[bind(1),bind(0),bind(1),bind(0)]).
hex_equiv(hexd(b),[bind(1),bind(0),bind(1),bind(1)]).
hex_equiv(hexd(c),[bind(1),bind(1),bind(0),bind(0)]).
hex_equiv(hexd(d),[bind(1),bind(1),bind(0),bind(1)]).
hex_equiv(hexd(e),[bind(1),bind(1),bind(1),bind(0)]).
hex_equiv(hexd(f),[bind(1),bind(1),bind(1),bind(1)]).
```

PREDICATE juntar_listas/3

Usage: `juntar_listas(A,B,C)`

`C` es la unificación de las listas `A` y `B` .

```
juntar_listas([],X,X).
juntar_listas([X|Xs],Y,[X|Rs]) :-
    juntar_listas(Xs,Y,Rs).
```

PREDICATE byte_list_conversion/2

Usage: `byte_list_conversion(A,B)`

`A` es una lista de bits hexadecimales y `B` es la lista de bits binarios equivalentes.

```
byte_list_conversion([X],[Y]) :-
    byte_conversion(X,Y).
byte_list_conversion([X|Xs],[Y|Ys]) :-
    byte_conversion(X,Y),
    byte_list_conversion(Xs,Ys).
```

Other properties:

Test: `byte_list_conversion(A,B)`

- *If the following properties hold at call time:*
  `A=[[hexd(3),hexd(5)],[hexd(4),hexd(e)]]` (= /2)
  then the following properties should hold upon exit:
  `B=[[bind(0),bind(0),bind(1),bind(1),bind(0),bind(1),bind(0),bind(1)],[bind(0),bind(1),bind(0),bind(0),bind(1),bind(1),bind(1),bind(0)]]` (= /2)
  then the following properties should hold globally:
  All the calls of the form `byte_list_conversion(A,B)` do not fail. (not_fails/1)

PREDICATE get_nth_bit_from_byte/3

Usage: `get_nth_bit_from_byte(A,B,C)`

`C` es el bit del byte `B` situado en la posición `A` .

```
get_nth_bit_from_byte(X,Y,Z) :-
    ( hex_byte(Y),
      resta(s(s(s(s(s(s(s(0)))))),X,A),
      byte_conversion(Y,B),
      sacar_bit(A,B,Z)
    ; binary_byte(Y),
      resta(s(s(s(s(s(s(s(0)))))),X,A),
      sacar_bit(A,Y,Z)
    ).
```

Other properties:

Test: `get_nth_bit_from_byte(A,B,C)`

- *If the following properties hold at call time:*
  `A=s(s(s(s(s(0)))))` (= /2)
  `B=[bind(1),bind(0),bind(1),bind(0),bind(1),bind(1),bind(0),bind(0)]` (= /2)
  then the following properties should hold upon exit:
  `C=bind(1)` (= /2)

then the following properties should hold globally:

All the calls of the form `get_nth_bit_from_byte(A,B,C)` do not fail. (`not_fails/1`)

Test: `get_nth_bit_from_byte(A,B,C)`

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

  `A=s(s(s(s(s(0)))))` (`= /2`)

  `B=[hexd(a),hexd(c)]` (`= /2`)

  then the following properties should hold upon exit:

  `C=bind(1)` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `get_nth_bit_from_byte(A,B,C)` do not fail. (`not_fails/1`)

PREDICATE resta/3

Usage: `resta(A,B,C)`

`C` es el resultado de restar `A` menos `B` .

```
resta(X,0,X).
resta(0,_1,0).
resta(s(X),s(Y),Z) :-
    resta(X,Y,Z).
```

PREDICATE sacar_bit/3

Usage: `sacar_bit(A,B,C)`

`C` es el bit del byte `B` situado en la posicion siete menos `A` .

```
sacar_bit(0,[X|_1],X).
sacar_bit(s(X),[_1|Yz],Z) :-
    sacar_bit(X,Yz,Z).
```

PREDICATE byte_list_clsh/2

Usage: `byte_list_clsh(A,B)`

`B` es la lista de bytes `A` desplazada una posicion a la izquierda.

```
byte_list_clsh([X|Xs],Y) :-
    ( hex_byte(X),
      byte_list_conversion([X|Xs],A),
      unificar(A,B),
      trasponer_izq(B,C),
      restaurar(C,s(s(s(s(s(s(s(0)))))))),D),
      byte_list_conversion_2(D,Y)
```

```
    ; binary_byte(X),
      unificar([X|Xs],B),
      trasponer_izq(B,C),
      restaurar(C,s(s(s(s(s(s(s(s(0)))))))),Y)
    ).
```

Other properties:

Test: `byte_list_clsh(A,B)`

- *If the following properties hold at call time:*
  `A=[[bind(0),bind(1),bind(0),bind(1),bind(1),bind(0),bind(0),bind(1)],` (`= /2`)
  `[bind(0),bind(0),bind(1),bind(0),bind(0),bind(0),bind(1),bind(1)],`
  `[bind(0),bind(1),bind(0),bind(1),bind(0),bind(1),bind(0),bind(1)],`
  `[bind(0),bind(0),bind(1),bind(1),bind(0),bind(1),bind(1),bind(1)]]`
  then the following properties should hold upon exit:
  `B=[[bind(1),bind(0),bind(1),bind(1),bind(0),bind(0),bind(1),bind(0)],` (`= /2`)
  `[bind(0),bind(1),bind(0),bind(0),bind(0),bind(1),bind(1),bind(0)],`
  `[bind(1),bind(0),bind(1),bind(0),bind(1),bind(0),bind(1),bind(0)],`
  `[bind(0),bind(1),bind(1),bind(0),bind(1),bind(1),bind(1),bind(0)]]`
  then the following properties should hold globally:
  All the calls of the form `byte_list_clsh(A,B)` do not fail. (`not_fails/1`)

Test: `byte_list_clsh(A,B)`

- *If the following properties hold at call time:*
  `A=[[hexd(5),hexd(a)],[hexd(2),hexd(3)],[hexd(5),hexd(5)],[hexd(3),hexd(7)]]` (`= /2`)
  then the following properties should hold upon exit:
  `B=[[hexd(b),hexd(4)],[hexd(4),hexd(6)],[hexd(a),hexd(a)],[hexd(6),hexd(e)]]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `byte_list_clsh(A,B)` do not fail. (`not_fails/1`)

PREDICATE unificar/2

Usage: `unificar(A,B)`

`B` es la lista de listas `A` unificada en una unica lista.

```
unificar([X],X).
unificar([X|Y],Z) :-
    unificar(Y,A),
    juntar_listas(X,A,Z).
```

PREDICATE trasponer_izq/2

Usage: `trasponer_izq(A,B)`

`B` es la lista de bits `A` unificada en una unica lista.

```
trasponer_izq([X|Xs],Y) :-
    juntar_listas(Xs,[X],Y).
```

PREDICATE **restaurar/3**

Usage: `restaurar(A,B,C)`

`C` agrupa la lista `A` en una lista de listas de `B` elementos.

```
restaurar([],_1,[]).
restaurar(X,Y,[Z|Zs]) :-
    juntar_listas(Z,A,X),
    longitud_lista(Z,Y),
    restaurar(A,Y,Zs).
```

PREDICATE **longitud_lista/2**

Usage: `longitud_lista(A,B)`

`B` es la longitud de la lista `A` .

```
longitud_lista([],0).
longitud_lista([_1|Xs],s(Y)) :-
    longitud_lista(Xs,Y).
```

PREDICATE **byte_list_conversion_2/2**

Usage: `byte_list_conversion_2(A,B)`

`A` es una lista de bits binarios y `B` es la lista de bits hexadecimales equivalentes.

```
byte_list_conversion_2([X],[Y]) :-
    byte_conversion_2(X,Y).
byte_list_conversion_2([X|Xs],[Y|Ys]) :-
    byte_conversion_2(X,Y),
    byte_list_conversion_2(Xs,Ys).
```

PREDICATE **byte_conversion_2/2**

Usage: `byte_conversion_2(A,B)`

`A` es un byte binario y `B` es el byte hexadecimal equivalente.

```
byte_conversion_2(X,Y) :-
    separar_lista(X,s(s(s(s(0)))),A,B),
    hex_equiv(C,A),
    hex_equiv(D,B),
    juntar_listas([C],[D],Y).
```

PREDICATE separar_lista/4

Usage: `separar_lista(A,B,C,D)`

`C` es una lista que contiene los `B` primeros elementos de la lista `A` y D contiene el resto.

```
separar_lista([H|T],0,[],[H|T]).
separar_lista([X|Xs],s(Y),[X|Zs],Z) :-
    separar_lista(Xs,Y,Zs,Z).
```

PREDICATE byte_list_crsh/2

Usage: `byte_list_crsh(A,B)`

`B` es la lista de bytes `A` desplazada una posicion a la derecha.

```
byte_list_crsh([X|Xs],Y) :-
    ( hex_byte(X),
      byte_list_conversion([X|Xs],A),
      unificar(A,B),
      longitud_lista(B,s(C)),
      trasponer_drch(B,C,D),
      restaurar(D,s(s(s(s(s(s(s(s(0)))))))),E),
      byte_list_conversion_2(E,Y)
    ; binary_byte(X),
      unificar([X|Xs],A),
      longitud_lista(A,s(B)),
      trasponer_drch(A,B,C),
      restaurar(C,s(s(s(s(s(s(s(s(0)))))))),Y)
    ).
```

Other properties:

Test: `byte_list_crsh(A,B)`

- *If the following properties hold at call time:*
  `A=[[bind(1),bind(0),bind(1),bind(1),bind(0),bind(1),bind(0),bind(0)],` (= /2)
  `[bind(0),bind(1),bind(0),bind(0),bind(0),bind(1),bind(1),bind(0)],`
  `[bind(1),bind(0),bind(1),bind(0),bind(1),bind(0),bind(1),bind(0)],`
  `[bind(0),bind(1),bind(1),bind(0),bind(1),bind(1),bind(1),bind(0)]]`
  then the following properties should hold upon exit:
  `B=[[bind(0),bind(1),bind(0),bind(1),bind(1),bind(0),bind(1),bind(0)],` (= /2)
  `[bind(0),bind(0),bind(1),bind(0),bind(0),bind(0),bind(1),bind(1)],`

`[bind(0),bind(1),bind(0),bind(1),bind(0),bind(1),bind(0),bind(1)],`
`[bind(0),bind(0),bind(1),bind(1),bind(0),bind(1),bind(1),bind(1)]]`
then the following properties should hold globally:
All the calls of the form `byte_list_crsh(A,B)` do not fail. (`not_fails/1`)

Test: `byte_list_crsh(A,B)`

- *If the following properties hold at call time:*
  `A=[[hexd(b),hexd(4)],[hexd(4),hexd(6)],[hexd(a),hexd(a)],[hexd(6),hexd(e)]]` (`= /2`)
  then the following properties should hold upon exit:
  `B=[[hexd(5),hexd(a)],[hexd(2),hexd(3)],[hexd(5),hexd(5)],[hexd(3),hexd(7)]]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `byte_list_crsh(A,B)` do not fail. (`not_fails/1`)

PREDICATE trasponer_drch/3

Usage: `trasponer_drch(A,B,C)`

`C` es la lista de bytes `A` desplazada `B` posiciones a la izquierda.

```
trasponer_drch(X,0,X).
trasponer_drch(X,s(Y),Z) :-
    trasponer_izq(X,A),
    trasponer_drch(A,Y,Z).
```

PREDICATE byte_xor/3

Usage: `byte_xor(A,B,C)`

`C` es el resultado de aplicar la operación lógica XOR entre los bytes `A` y `B` .

```
byte_xor(X,Y,Z) :-
    ( hex_byte(X),
      byte_conversion(X,A),
      byte_conversion(Y,B),
      operacion_xor(A,B,C),
      byte_conversion_2(C,Z)
    ; operacion_xor(X,Y,Z)
    ).
```

Other properties:

Test: `byte_xor(A,B,C)`

- *If the following properties hold at call time:*
  `A=[bind(0),bind(1),bind(0),bind(1),bind(1),bind(0),bind(1),bind(0)]` (`= /2`)
  `B=[bind(0),bind(0),bind(1),bind(0),bind(0),bind(0),bind(1),bind(1)]` (`= /2`)
  then the following properties should hold upon exit:

`C=[bind(0),bind(1),bind(1),bind(1),bind(1),bind(0),bind(0),bind(1)]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `byte_xor(A,B,C)` do not fail. (`not_fails/1`)

Test: `byte_xor(A,B,C)`

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

  `A=[hexd(5),hexd(a)]` (`= /2`)

  `B=[hexd(2),hexd(3)]` (`= /2`)

  then the following properties should hold upon exit:

  `C=[hexd(7),hexd(9)]` (`= /2`)

  then the following properties should hold globally:

  All the calls of the form `byte_xor(A,B,C)` do not fail. (`not_fails/1`)

PREDICATE operacion_xor/3

Usage: `operacion_xor(A,B,C)`

`C` es el resultado de aplicar la operación lógica XOR entre los bytes `A` y `B` .

```
operacion_xor([],[],[]).
operacion_xor([X|Xs],[Y|Ys],[Z|Zs]) :-
    ( igual(X,Y),
      igual(Z,bind(0)),
      operacion_xor(Xs,Ys,Zs)
    ; no_igual(X,Y),
      igual(bind(1),Z),
      operacion_xor(Xs,Ys,Zs)
    ).
```

PROPERTY igual/2

Usage: `igual(A,B)`

`A` y `B` son iguales.

```
igual(X,X).
```

PROPERTY no_igual/2

Usage: `no_igual(A,B)`

`A` y `B` no son iguales.

```
no_igual(bind(0),bind(1)).
no_igual(bind(1),bind(0)).
```

Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.