UTF-8

Practica 1 Programacion logica y declarativa

Juan Jimenez Perez 190204

Table of Contents

code **1**
Pruebas de ejecucion 1
Usage and interface 2
Documentation on exports 2
hex_a_byte/2 (prop) 2
my_nth_bit_from_byte/3 (pred) 3
invertir/2 (pred) 3
concatenar/3 (pred) 3
byte_bit/2 (pred) 3
binary_list/1 (pred) 4
hex_list/1 (pred) 4
xor/3 (prop) 4
byte_xor_aux/3 (pred) 4
byte_list/1 (pred) 5
byte_conversion/2 (pred) 5
byte_list_conversion/2 (pred) 5
get_nth_bit_from_byte/3 (pred) 5
byte_list_clsh/2 (pred) 6
byte_list_crsh/2 (pred) 6
byte_xor/3 (pred) 7
Documentation on imports 7

References **9**

code

Pruebas de ejecucion

1. Predicado 1 byte_list/1:

```
?- byte_list (L).

L = [[bind(0), bind(0), bind(0), bind(0),
bind(0), bind(0), bind(0), bind(0)]] ?
yes
?-
```

2. Predicado 2 byte_conversion/2:

```
?- 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
?-
```

3. Predicado 3 byte_list_conversion/2:

```
?- byte_list_conversion ([[hexd (3), hexd (5)],
[hexd (4), hexd(e)]], BL).

BL = [[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)]] ?;
no
?-
```

4. Predicado 4 get_nth_bit_from_byte/3:

```
?- get_nth_bit_from_byte (s(s(s(s(s(0))))),
[bind (1), bind (0), bind (1), bind (0),
bind (1), bind (1), bind (0), bind (0)], B).

B = bind (1) ?;
no
?-
```

5. Predicado 5 byte_list_clsh/2:

```
?- byte_list_clsh ([[hexd (5), hexd(a)],
[hexd (2), hexd (3)], [hexd (5), hexd (5)],
[hexd (3), hexd (7)]], L).

L = [[hexd(b), hexd (4)], [hexd (4), hexd (6)],
[hexd(a), hexd(a)], [hexd (6), hexd(e)]] ?;
no
```

```
?-
```

6. Predicado 6 byte_list_crsh/2:

```
?- byte_list_crsh ([[hexd(b), hexd (4)],
[hexd (4), hexd (6)], [hexd(a), hexd(a)],
[hexd (6), hexd(e)]], L).

L = [[hexd (5), hexd(a)], [hexd (2), hexd (3)],
[hexd (5), hexd (5)], [hexd (3), hexd (7)]] ?;
no
?-
```

7. Predicado 7 byte_xor/3:

```
?- byte_xor ([hexd (5) ,hexd(a)],
[hexd (2) ,hexd (3)], B3).

B3 = [hexd (7), hexd (9)] ?;
no
?-
```

Usage and interface

- **Library usage:**
  `:- use_module(/home/juan/Documents/Prolog/code.pl).`
- **Exports:**
  - *Predicates:*
    `my_nth_bit_from_byte/3`, `invertir/2`, `concatenar/3`, `byte_bit/2`, `binary_list/1`, `hex_list/1`, `byte_xor_aux/3`, `byte_list/1`, `byte_conversion/2`, `byte_list_conversion/2`, `get_nth_bit_from_byte/3`, `byte_list_clsh/2`, `byte_list_crsh/2`, `byte_xor/3`.
  - *Properties:*
    `hex_a_byte/2`, `xor/3`.

Documentation on exports

hex_a_byte/2: PROPERTY

Convierte de hexadecimal a binario al reves:

```
hex_a_byte(hexd(0),[bind(0),bind(0),bind(0),bind(0)]).
hex_a_byte(hexd(1),[bind(0),bind(0),bind(0),bind(1)]).
hex_a_byte(hexd(2),[bind(0),bind(0),bind(1),bind(0)]).
hex_a_byte(hexd(3),[bind(0),bind(0),bind(1),bind(1)]).
hex_a_byte(hexd(4),[bind(0),bind(1),bind(0),bind(0)]).
hex_a_byte(hexd(5),[bind(0),bind(1),bind(0),bind(1)]).
hex_a_byte(hexd(6),[bind(0),bind(1),bind(1),bind(0)]).
hex_a_byte(hexd(7),[bind(0),bind(1),bind(1),bind(1)]).
hex_a_byte(hexd(8),[bind(1),bind(0),bind(0),bind(0)]).
```

```
hex_a_byte(hexd(9),[bind(1),bind(0),bind(0),bind(1)]).
hex_a_byte(hexd(a),[bind(1),bind(0),bind(1),bind(0)]).
hex_a_byte(hexd(b),[bind(1),bind(0),bind(1),bind(1)]).
hex_a_byte(hexd(c),[bind(1),bind(1),bind(0),bind(0)]).
hex_a_byte(hexd(d),[bind(1),bind(1),bind(0),bind(1)]).
hex_a_byte(hexd(e),[bind(1),bind(1),bind(1),bind(0)]).
hex_a_byte(hexd(f),[bind(1),bind(1),bind(1),bind(1)]).
```

Usage: `hex_a_byte(HB,BB)`

`BB` conversion del numero hexadecimal `HB` en binario.

my_nth_bit_from_byte/3: PREDICATE

Recorre la lista hasta encontrar el byte pasado como primer parametro:

```
my_nth_bit_from_byte(0,[X|_1],X).
my_nth_bit_from_byte(s(X),[_1|C],B) :-
    my_nth_bit_from_byte(X,C,B).
```

Recorre la lista reduciendo el indice hasta llegar al 0. Al llegar al 0 devuelve el elemento en dicha posicion

Usage: `my_nth_bit_from_byte(s(X),L,R)`

`R` bit situado en el indice `s(X)` de la lista `L`.

invertir/2: PREDICATE

Invierte la lista pasada como argumento:

```
invertir([],[]).
invertir([X|Y],L) :-
    invertir(Y,L1),
    concatenar(L1,[X],L).
```

Recorre la lista con llamadas recursivas hasta llegar a la lista vacia, una vez ahi, concatena los distintos elementos de la lista.

Usage: `invertir(L,R)`

`R` resultado de invertir la lista `L`.

concatenar/3: PREDICATE

Concatena las listas pasadas como argumento:

```
concatenar([],L,L).
concatenar([X|Y],Z,[X|U]) :-
    concatenar(Y,Z,U).
```

Aade los elementos de la primera lista en la lista resultado y llama recursivamente al predicado, cuando no hay mas elementos en la lista inicial devuelve la lista

Usage: `concatenar(L1,L2,R)`

`R` resultado de concatenar las listas `L1` y `L2`.

byte_bit/2: PREDICATE

Convierte de byte a bit o al reves:

```
byte_bit([],[]).
byte_bit([[B7,B6,B5,B4,B3,B2,B1,B0]|Bytes],[B7,B6,B5,B4,B3,B2,B1,B0|Bits]) :-
    byte_bit(Bytes,Bits).
```

Transforma una lista de listas de ocho elementos en una unica lista.

Usage: `byte_bit(L,R)`

`R` resultado de transformar la lista de bytes `L` en una lista de bits.

binary_list/1: PREDICATE

Comprueba si la lista esta compuesta solo de elementos binarios:

```
binary_list([]).
binary_list([X|Y]) :-
    binary_byte(X),
    binary_list(Y).
```

Para comprobarlo recorre la lista comprobando si el primer elemento de la lista es binario, si lo es llama al predicado recursivamente con el resto de la lista hasta que no quedan mas elementos.

Usage: `binary_list(L)`

`L` lista pasada para comprobar si es de digitos binarios.

hex_list/1: PREDICATE

Comprueba si la lista esta compuesta solo de elementos hexadecimales:

```
hex_list([]).
hex_list([X|Y]) :-
    hex_byte(X),
    hex_list(Y).
```

Para comprobarlo recorre la lista comprobando si el primer elemento de la lista es hexadecimal, si lo es llama al predicado recursivamente con el resto de la lista hasta que no quedan mas elementos.

Usage: `hex_list(L)`

`L` lista pasada para comprobar si es de digitos hexadecimales.

xor/3: PROPERTY

Operaciones xor:

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

Usage: `xor(B1,B2,B3)`

`B3` resultado de hacer la operacion xor de `B1` y de `B2`.

byte_xor_aux/3: PREDICATE

Dadas dos listas realiza la operacion xor elemento a elemento:

```
byte_xor_aux([],[],_1).
byte_xor_aux([X|Y],[Z|T],[A|B]) :-
    xor(X,Z,A),
    byte_xor_aux(Y,T,B).
```

Selecciona el primer elemento de cada lista y realiza la operacion xor. Repite la operacion de forma recursiva con el resto de la lista hasta que ambas son vacias.

Usage: `byte_xor_aux(B1,B2,B3)`

`B3` lista resultado de operar las listas `B1` y `B2`.

byte_list/1: PREDICATE

Predicado 1:

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

Recorre los elementos de la lista comprobando si el primer elemento es binario o hexadecimal y llama recursivamente al predicado con el resto de la lista hasta que esta sea vacia.

Usage: `byte_list(L)`

`L` lista pasada para comprobar si es de digitos hexadecimales o binarios.

byte_conversion/2: PREDICATE

Predicado 2:

```
byte_conversion([H1,H0],[B7,B6,B5,B4,B3,B2,B1,B0]) :-
    hex_byte([H1,H0]),
    hex_a_byte(H1,[B7,B6,B5,B4]),
    hex_a_byte(H0,[B3,B2,B1,B0]).
```

Comprueba que el argumento pasado es hexadecimal y despues los convierte a binario.

Usage: `byte_conversion(HB,B)`

`B` resultado de convertir a binario `HB`.

byte_list_conversion/2: PREDICATE

Predicado 3:

```
byte_list_conversion([],[]).
byte_list_conversion([X|Y],[Z|T]) :-
    byte_conversion(X,Z),
    byte_list_conversion(Y,T).
```

Recorre los elementos de la lista convirtiendo el primer elemento de hexadecimal a binario y llama recursivamente al predicado con el resto de la lista hasta que esta sea vacia.

Usage: `byte_list_conversion(HL,BL)`

`BL` lista resultado de convertir a binario la lista `HL`.

get_nth_bit_from_byte/3: PREDICATE

Predicado 4:

```
get_nth_bit_from_byte(A,[H1,H0],B) :-
    hex_byte([H1,H0]),
    byte_conversion([H1,H0],C),
    invertir(C,R),
    my_nth_bit_from_byte(A,R,B).
get_nth_bit_from_byte(A,BB,B) :-
    binary_byte(BB),
    invertir(BB,R),
    my_nth_bit_from_byte(A,R,B).
```

Comprueba que los datos pasados son validos, y si son hexadecimales los convierte a binarios llamando al predicado 2. Invierte la lista y llama al predicado my_nth_bit_from_byte/3 para buscar el elemento en la lista.

Usage: `get_nth_bit_from_byte(N,B,BN)`

`BN` bit situado en el indice `N` de la lista `B`.

byte_list_clsh/2: PREDICATE

Predicado 5:

```
byte_list_clsh(L,CLShL) :-
    hex_list(L),
    byte_list_conversion(L,R),
    byte_bit(R,[X|Y]),
    concatenar(Y,[X],A),
    byte_bit(B,A),
    byte_list_conversion(CLShL,B).
byte_list_clsh(L,CLShL) :-
    binary_list(L),
    byte_bit(L,[X|Y]),
    concatenar(Y,[X],R),
    byte_bit(CLShL,R).
```

Comprueba que los elementos pasados son validos, y si estan en hexadecimal los convierte a binario llamando al predicado 3. Transforma la lista de bytes a una lista de bits, llamando a byte_bit/2, en la que mueve el primer elemento a la ultima posicion. Convierte de nuevo la lista de bits a bytes y si es hexadecimal la transforma de nuevo a hexadecimal.

Usage: `byte_list_clsh(L,CLShL)`

`CLShL` lista resultado de rotar la lista `L` a la izquierda.

byte_list_crsh/2: PREDICATE

Predicado 6:

```
byte_list_crsh(L,CRShL) :-
    hex_list(L),
    byte_list_conversion(L,A),
    byte_bit(A,B),
    invertir(B,[X|Y]),
    concatenar(Y,[X],C),
    invertir(C,D),
```

```
    byte_bit(R,D),
    byte_list_conversion(CRShL,R).
byte_list_crsh(L,CRShL) :-
    binary_list(L),
    byte_bit(L,A),
    invertir(A,[X|Y]),
    concatenar(Y,[X],B),
    invertir(B,R),
    byte_bit(CRShL,R).
```

Comprueba que los elementos pasados son validos, y si estan en hexadecimal los convierte a binario llamando al predicado 3. Transforma la lista de bytes a una lista de bits llamando a byte_bit/2 y la invierte llamando a invertir/2. Mueve el primer elemento a la ultima posicion y la invierte de nuevo. Convierte de nuevo la lista de bits a bytes y si es hexadecimal la transforma de nuevo a hexadecimal.

Usage: `byte_list_crsh(L,CRShL)`

`CRShL` lista resultado de rotar la lista `L` a la derecha.

byte_xor/3: PREDICATE

Predicado 7:

```
byte_xor(H1,H2,H3) :-
    hex_byte(H1),
    hex_byte(H2),
    byte_conversion(H1,A),
    byte_conversion(H2,B),
    byte_xor_aux(A,B,C),
    byte_conversion(H3,C).
byte_xor(B1,B2,B3) :-
    binary_byte(B1),
    binary_byte(B2),
    byte_xor_aux(B1,B2,B3),
    binary_byte(B3).
```

Comprueba que ambos datos pasados son validos, y en caso de ser hexadecimales los transforma a binario llamando al predicado 2. Llama al predicado byte_xor_aux/3 para calcular la operacion xor y por ultimo en caso de ser binario comprueba que el resultado es correcto y en caso de ser hexadecimal lo vuelve a convertir a hexadecimal.

Usage: `byte_xor(B1,B2,B3)`

`B3` resultado de hacer la operacion xor de `B2` y `B1`.

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`.

References

(this section is empty)