UTF-8

Memoria Bits, Nibbles & Bytes

Luis Dominguez Romero, z170298

Table of Contents

code

Este documento representa la memoria de la primera prctica.

Usage and interface

- **Library usage:**
  `:- use_module(/home/luis/Escritorio/Practica Prode/code.pl).`
- **Exports:**
  - *Predicates:*
    `author_data/4`, `bind/1`, `binary_byte/1`, `hexd/1`, `byte/1`, `hex_byte/1`, `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`, `binary_to_nibble/2`, `my_concat/3`, `indexOf/3`, `resta/3`, `is_binary_list/1`, `concat_matrix/2`, `extract_first/3`, `rotate/2`, `list_to_matrix/2`, `reverse/2`, `logic_operation/3`, `xor/3`.

Documentation on exports

author_data/4: PREDICATE

Usage: `author_data(Dominguez,Romero,Luis,Z170298)`

Estos son mis datos.

bind/1: PREDICATE

Usage: `bind(X)`

Este predicado es el encargado de definir un bit, el cual tomar el valor de la variable `X` [0,1].

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

binary_byte/1: PREDICATE

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

Este predicado define la estructura de un byte. Se trata de una lista de ocho bits donde `B7` ser el valor del bit ms significativo y `B0` ser el valor del bit menos significativo.

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

hexd/1: PREDICATE

Usage: `hexd(X)`

Este predicado es el encargado de definir un nibble. El valor de este nibble vendr asociado al de la `X`.

```
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).
```

byte/1: PREDICATE

Usage: `byte(Byte)`

Este predicado se encarga de la definicin de un byte, que puede ser representado como una lista de dos nibble o una lista de ocho bits.

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

hex_byte/1: PREDICATE

Usage: `hex_byte([hexd(H1),hexd(H0)])`

Este predicado se encarga de la definicin de un byte a partir de una lista de dos nibble. Los bits ms significativos sern los de la variable `H1`, mientras que los menos significativos sern los de `H0`.

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

byte_list/1: PREDICATE

Usage: `byte_list(ByteList)`

Este predicado define listas de bytes a partir del argumento `ByteList`, que puede ser una lista de bytes binarios o hexadecimales.

```
byte_list([]).
byte_list([H|T]) :-
    hex_byte(H),
```

```
    byte_list(T).
byte_list([H|T]) :-
    binary_byte(H),
    byte_list(T).
```

byte_conversion/2: PREDICATE

Usage: `byte_conversion(HexByte,BinByte)`

Este predicado es cierto si el `HexByte` es la representacin hexadecimal de `BinByte`.

```
byte_conversion([H1,H0],Byte) :-
    binary_to_nibble(H0,B0),
    binary_to_nibble(H1,B1),
    my_concat(B1,B0,Byte).
```

byte_list_conversion/2: PREDICATE

Usage: `byte_list_conversion(HL,BL)`

Este predicado es cierto si la lista de bytes hexadecimales `HL` tiene como representacion binaria la lista bytes binarios `BL`.

```
byte_list_conversion([],[]).
byte_list_conversion([H1|T1],[BL|TBL]) :-
    byte_conversion(H1,BL),
    byte_list_conversion(T1,TBL).
```

get_nth_bit_from_byte/3: PREDICATE

Usage: `get_nth_bit_from_byte(Index,Byte,Bit)`

Este predicado es cierto si, dada un byte `Byte` (en representacion binaria o hexadecimal), el bit `Bit` se encuentra en la posicin `Index`.

```
get_nth_bit_from_byte(N,HexList,BN) :-
    byte_conversion(HexList,Byte),
    get_nth_bit_from_byte(N,Byte,BN).
get_nth_bit_from_byte(N,Byte,Bit) :-
    binary_byte(Byte),
    resta(s(s(s(s(s(s(s(0)))))),N,N1),
    indexOf(N1,Byte,Bit).
```

byte_list_clsh/2: PREDICATE

Usage: `byte_list_clsh(L,CL)`

Este predicado es cierto si la lista de bytes (en representacin binaria o hexadecimal) `CL`,es el resultado de aplicar un desplazamiento circular hacia la izquierda a la lista de bytes `L`.

```
byte_list_clsh(L,CL) :-
    byte_list_conversion(L,ByteList),
    concat_matrix(ByteList,Vector),
    rotate(Vector,VectorRes),
    list_to_matrix(VectorRes,ByteListRes),
    byte_list_conversion(CL,ByteListRes).
```

```
byte_list_clsh(L,CL) :-
    is_binary_list(L),
    concat_matrix(L,Vector),
    rotate(Vector,VectorRes),
    list_to_matrix(VectorRes,CL).
```

byte_list_crsh/2: PREDICATE

Usage: `byte_list_crsh(L,CL)`

Este predicado es cierto si la lista de bytes (en representacin binaria o hexadecimal) `CL`,es el resultado de aplicar un desplazamiento circular hacia la derecha a la lista de bytes `L`.

```
byte_list_crsh(L,CL) :-
    byte_list_conversion(L,ByteList),
    concat_matrix(ByteList,Vector),
    reverse(Vector,VectorRever),
    rotate(VectorRever,VectorRes),
    reverse(VectorRes,VectorFinal),
    list_to_matrix(VectorFinal,ByteListRes),
    byte_list_conversion(CL,ByteListRes).
byte_list_crsh(L,CL) :-
    is_binary_list(L),
    concat_matrix(L,Vector),
    reverse(Vector,VectorRever),
    rotate(VectorRever,VectorRes),
    reverse(VectorRes,VectorFinal),
    list_to_matrix(VectorFinal,CL).
```

byte_xor/3: PREDICATE

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

Este predicado es cierto si el byte `B3` es el resultado de efectuar la operacin lgica xor e los bytes `B1` y `B2`. La representacin de estos bytes puede ser binaria o hexadecimal.

```
byte_xor(B1,B2,B3) :-
    byte_conversion(B1,Byte1),
    byte_conversion(B2,Byte2),
    logic_operation(Byte1,Byte2,Byte3),
    byte_conversion(B3,Byte3).
byte_xor(B1,B2,B3) :-
    logic_operation(B1,B2,B3).
```

binary_to_nibble/2: PREDICATE

Usage: `binary_to_nibble(Hex,Nibble)`

Este predicado se encarga de definir la equivalencia entre el valor de un `Nibble` y su valor en hexadecimal `Hex`.

```
binary_to_nibble(hexd(0),[bind(0),bind(0),bind(0),bind(0)]).
binary_to_nibble(hexd(1),[bind(0),bind(0),bind(0),bind(1)]).
binary_to_nibble(hexd(2),[bind(0),bind(0),bind(1),bind(0)]).
binary_to_nibble(hexd(3),[bind(0),bind(0),bind(1),bind(1)]).
```

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

my_concat/3: PREDICATE

Usage: `my_concat(List1,List2,Res)`

Este predicado es cierto si la lista `Res` es el resultado de concatenar `List1` con `List2`.

```
my_concat([],ListaPost,ListaPost).
my_concat([H1|T1],ListaPost,[H1|T2]) :-
    my_concat(T1,ListaPost,T2).
```

indexOf/3: PREDICATE

Usage: `indexOf(Index,List,Elem)`

Este predicado es cierto si, dada una lista `List`, el elemento `Elem` se encuentra en la posicin `Index`.

```
indexOf(0,[H|T],H) :-
    list(T).
indexOf(s(X),[_1|T],H) :-
    indexOf(X,T,H).
```

resta/3: PREDICATE

Usage: `resta(N1,N2,Res)`

Este predicado es cierto si `Res` es el resultado de restar `N2` a `N1`.

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

is_binary_list/1: PREDICATE

Usage: `is_binary_list(ByteList)`

Este predicado es cierto si el argumento `ByteList` es una lista de bytes en representacin binaria.

```
is_binary_list([]).
is_binary_list([H|T]) :-
    binary_byte(H),
    is_binary_list(T).
```

concat_matrix/2: PREDICATE

Usage: `concat_matrix(Matrix,List)`

Este predicado es cierto si `List` es el resultado de concatenar todas las listas que contenga `Matrix`.

```
concat_matrix([],[]).
concat_matrix([H|T],ListaRes) :-
    concat_matrix(T,ListaRes1),
    my_concat(H,ListaRes1,ListaRes).
```

extract_first/3: PREDICATE

Usage: `extract_first(Elem,RestoLista,Lista)`

Este predicado es cierto si el resultado de extraer el primer elemento `Elem` a la lista `Lista` es `RestoLista`.

```
extract_first(X,T,[X|T]).
```

rotate/2: PREDICATE

Usage: `rotate(L,CL)`

Este predicado es cierto si `CL` es el resultado de aplicar un desplazamiento hacia la izquierda a la lista `L`.

```
rotate([],[]).
rotate(ListaEnt,ListaRes) :-
    extract_first(Elem,L1,ListaEnt),
    my_concat(L1,[Elem],ListaRes).
```

list_to_matrix/2: PREDICATE

Usage: `list_to_matrix(List,Matrix)`

Este predicado es cierto si la matriz `Matrix` es el resultado de agrupar varias listas de ocho elementos de la lista `List`.

```
list_to_matrix([],[]).
list_to_matrix([B7,B6,B5,B4,B3,B2,B1,B0|T1],[[B7,B6,B5,B4,B3,B2,B1,B0]|T2]) :-
    list_to_matrix(T1,T2).
```

reverse/2: PREDICATE

Usage: `reverse(L,LR)`

Este predicado es cierto si la lista `LR` es el resultado de invertir la lista `L` .

```
reverse([],[]).
reverse([H|T],Res) :-
    reverse(T,Res1),
    my_concat(Res1,[H],Res).
```

logic_operation/3: PREDICATE

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

Este predicado es cierto si el byte `B3` es el resultado de efectuar la operacin lgica xor e los bytes `B1` y `B2`. La representacin de estos bytes SOLO PUEDE SER BINARIA.

```
logic_operation([],[],[]).
logic_operation([Bit1|T1],[Bit2|T2],Res) :-
    xor(Bit1,Bit2,BitRes),
    logic_operation(T1,T2,Res1),
    my_concat([BitRes],Res1,Res).
```

xor/3: PREDICATE

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

Este predicado es el encargado de definir los valores de la operacin XOR con dos bits. `B3` ser el resultado de esta operacin entre los bits `B1` y `B2`.

```
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)).
```

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`, `regtypes`.

References

(this section is empty)