UTF-8

Primera practica (Programacion Logica Pura)

Carlos Pozo Serrano, B190234

Table of Contents

code

Pruebas:

1. byte_list/1:

```
?- byte_list([[bind(0),bind(0),bind(0),bind(0),
        bind(0),bind(0),bind(0),bind(0)]
,[hexd(a),hexd(1)]]).

yes
?- byte_list([bind(0)]).

no
?- byte_list(L).

L = [] ? ;

L = [[bind(0),bind(0),bind(0),bind(0),
     bind(0),bind(0),bind(0),bind(0)]] ? ;

L = [[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)]] ?

yes
?-
```

2. byte_conversion/2:

```
?- byte_conversion([hexd(a),hexd(2)],R).

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

no
?- byte_conversion([hexd(a)],R).

no
?- byte_conversion(H,
    [bind(1),bind(1),bind(0),bind(0),
    bind(1),bind(1),bind(1),bind(0)]).

H = [hexd(c),hexd(e)] ? ;

no
?-
```

3. byte_list_conversion/2

```
?- byte_list_conversion([[hexd(1),hexd(a)],
    [hexd(0),hexd(c)]],B).

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

no
?- byte_list_conversion(H,
    [[bind(1),bind(0),bind(1),bind(1),
    bind(1),bind(0),bind(0),bind(0)],
    [bind(0),bind(1),bind(1),bind(0),
    bind(0),bind(1),bind(0),bind(1)]]).

H = [[hexd(b),hexd(8)],[hexd(6),hexd(5)]] ? ;

no
?- byte_list_conversion([[hexd(1),hexd(a)],
    [hexd(0)]],B).

no
?-
```

4. get_nth_bit_from_byte/3:

```
?- get_nth_bit_from_byte(N,
    [bind(1),bind(0),bind(1),bind(0),
    bind(1),bind(0),bind(1),bind(1)],NB).

N = s(s(s(s(s(s(s(0)))))))),
NB = bind(1) ? ;

N = s(s(s(s(s(s(0))))))),
NB = bind(1) ? ;

N = s(s(s(s(s(0))))),
NB = bind(0) ? ;

N = s(s(s(s(0)))),
NB = bind(1) ?

yes

?- get_nth_bit_from_byte(0,B,bind(1)).

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

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

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

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

yes
?-
```

5. byte_list_clsh/2

```
?- byte_list_clsh([[hexd(a),hexd(1)],
    [hexd(b),hexd(5)]],CLShL).

CLShL = [[hexd(4),hexd(3)],[hexd(6),hexd(b)]] ? ;

no

?- byte_list_clsh([[hexd(3),hexd(2)],
    [hexd(e),hexd(6)]],
    [[hexd(6),hexd(5)],
    [hexd(c),hexd(c)]]).

yes
?-
```

6. byte_list_crsh/2

```
?- byte_list_crsh([[hexd(4),hexd(3)],
    [hexd(6),hexd(b)]],CRShL).

CRShL = [[hexd(a),hexd(1)],[hexd(b),hexd(5)]] ? ;

no

?- byte_list_crsh([[hexd(3),hexd(1)],
    [hexd(4),hexd(2)]],
    [[hexd(1),hexd(8)],
    [hexd(a),hexd(1)]]).

yes
?-
```

7. byte_xor/3

```
?- byte_xor([hexd(a),hexd(1)],[hexd(0),hexd(7)],C).

C = [hexd(a),hexd(6)] ? ;
```

```
no
?- byte_xor([bind(1),bind(1),bind(0),bind(1),
    bind(0),bind(0),bind(1),bind(1)],
    [bind(1),bind(1),bind(0),bind(1),
    bind(1),bind(0),bind(1),bind(1)],C).

C = [bind(0),bind(0),bind(0),bind(0),
    bind(1),bind(0),bind(0),bind(0)] ?

yes
?-
```

Usage and interface

- **Library usage:**
  `:- use_module(/mnt/c/Users/carlo/Desktop/Prolog/code.pl).`
- **Exports:**
  - *Predicates:*
    `binary_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`, `append/3`, `invert/2`, `aux4/3`, `hex_byte_list/1`, `binary_byte_list/1`, `byte_to_bit/2`, `xor_list/3`.
  - *Properties:*
    `bind/1`, `hexd/1`, `hextobind/2`, `xor/3`.

Documentation on exports

bind/1: PROPERTY

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

Usage: `bind(B)`

Dado un elemento `B` determina si es un digito binario.

hexd/1: PROPERTY

```
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).
```

Usage: `hexd(H)`

Dado un elemento `H` determina si es un digito hexadecimal.

binary_byte/1: PREDICATE

```
binary_byte([bind(A),bind(B),bind(C),bind(D),bind(E),bind(F),bind(G),bind(H)])
    bind(A),
    bind(B),
    bind(C),
    bind(D),
    bind(E),
    bind(F),
    bind(G),
    bind(H).
```

Usage: `binary_byte(B)`

`B`: lista a comprobar si forma un byte binario.

hex_byte/1: PREDICATE

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

Usage: `hex_byte(H)`

`H`: lista a comprobar si forma un byte hexadecimal.

hextobind/2: PROPERTY

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

Usage: `hextobind(H,R)`

Dado un valor hexadecimal `H` devuelve su valor `R` binario asociado.

byte_list/1: PREDICATE

Predicado 1: Extrae elementos de la lista, comprueba si son bytes binarios o hexadecimales y se hace una llamada recursiva al predicado con el resto de la lista. El caso base es que la lista este vacia.

```
byte_list([]).
byte_list([A|L]) :-
    binary_byte(A),
    byte_list(L).
byte_list([A|L]) :-
    hex_byte(A),
    byte_list(L).
```

Usage: `byte_list(L)`

`L`: lista a comprobar si esta formada por bytes binarios o hexadecimales.

byte_conversion/2: PREDICATE

Predicado 2: Comprueba que el primer argumento es un byte hexadecimal y despues lo convierte a un byte binario.

```
byte_conversion([X,Y],R) :-
    hextobind(X,L1),
    hextobind(Y,L2),
    append(L1,L2,R).
```

Usage: `byte_conversion(H,R)`

`H`: byte en hexadecimal, `R`: byte en binario.

byte_list_conversion/2: PREDICATE

Predicado 3: 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.

```
byte_list_conversion([],[]).
byte_list_conversion([X|Xs],[L|R]) :-
    byte_conversion(X,L),
    byte_list_conversion(Xs,R).
```

Usage: `byte_list_conversion(HL,BL)`

`HL`: lista de bytes hexadecimales, `BL`: lista de bytes binarios.

get_nth_bit_from_byte/3: PREDICATE

Predicado 4: Comprueba que el segundo argumento es un byte, si es hexadecimal lo convierte a binario llamando al predicado 2, a continuacion invierte la lista de bits y llama al predicado aux4/3 para buscar el elemento de la posicion que se indica en el primer argumento en la lista.

```
get_nth_bit_from_byte(N,B,NB) :-
    binary_byte(B),
    aux4(N,B,NB).
get_nth_bit_from_byte(N,B,NB) :-
    hex_byte(B),
    byte_conversion(B,L),
    invert(L,R),
    aux4(N,R,NB).
```

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

`N`: posicion del bit, `B`; byte del que extraer el bit, `L`: valor del bit.

byte_list_clsh/2: PREDICATE

Predicado 5: Comprueba que el primer argumento es una lista de bytes binarios o hexadecimales, si es una lista de hexadecimales los convierte a binario llamando al predicado 3. A continuacion une las listas de bytes en una lista de bits, llamando a byte_to_bit/2. Posteriormente se mueve el primer elemento de la lista a la ultima posicion.Por ultimo transforman los bits en bytes usando de nuevo el predicado byte_to_bit/2 y si es necesario se convierte el resultado a hexadecimal.

```
byte_list_clsh(L,CLShL) :-
    hex_byte_list(L),
    byte_list_conversion(L,R),
    byte_to_bit(R,[B1|Bits]),
    append(Bits,[B1],R1),
    byte_to_bit(Bytes,R1),
    byte_list_conversion(CLShL,Bytes).
byte_list_clsh(L,CLShL) :-
    binary_byte_list(L),
    byte_to_bit(L,[B1|Bits]),
    append(Bits,[B1],D),
    byte_to_bit(CLShL,D).
```

Usage: `byte_list_clsh(L,CLShL)`

`L`: lista de bytes, `CLShL`: lista con un bit rotado a la izquierda.

byte_list_crsh/2: PREDICATE

Predicado 6: Comprueba que el primer argumento es una lista de bytes binarios o hexadecimales, si es una lista de hexadecimales los convierte a binario llamando al predicado 3. A continuacion une las listas de bytes en una lista de bits, llamando a byte_to_bit/2. Se invierte la lista llmando a invert/2 y posteriormente se mueve el primer elemento de la lista a la ultima posicion. Por ultimo se invierte la lista de nuevo y se transforman los bits en bytes usando de nuevo el predicado byte_to_bit/2 y si es necesario se convierte el resultado a hexadecimal.

```
byte_list_crsh(L,CRShL) :-
    hex_byte_list(L),
    byte_list_conversion(L,R),
    byte_to_bit(R,I),
    invert(I,[B1|Bits]),
    append(Bits,[B1],I1),
    invert(I1,R1),
```

```
        byte_to_bit(Bytes,R1),
        byte_list_conversion(CRShL,Bytes).
    byte_list_crsh(L,CRShL) :-
        binary_byte_list(L),
        byte_to_bit(L,I),
        invert(I,[B1|Bits]),
        append(Bits,[B1],I1),
        invert(I1,D),
        byte_to_bit(CRShL,D).
```

Usage: `byte_list_crsh(L,CRShL)`

`L`: lista de bytes, `CRShL`: lista con un bit rotado a la derecha.

byte_xor/3: PREDICATE

Predicado 7: Comprueba que los dos primeros operados son bytes, si son hexadecimales los transforma a binario llamando al predicado 2. Llama al predicadoxor_list/3 para calcular la operacion xor, por ultimo, en caso de ser necesario, transforma el resultado binario en hexadecimal.

```
    byte_xor(A,B,C) :-
        binary_byte(A),
        binary_byte(B),
        binary_byte(C),
        xor_list(A,B,C).
    byte_xor(A,B,C) :-
        hex_byte(A),
        hex_byte(B),
        byte_conversion(A,AB),
        byte_conversion(B,BB),
        xor_list(AB,BB,CB),
        byte_conversion(C,CB).
```

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

`B1`: primer operando, `B2`: segundo operando, `B3`: resultado del xor.

append/3: PREDICATE

Anade el primer elemento de la primera lista y lo pone al inicio de la lista resultado, a continuacion se llama de forma recursiva a la misma funcion con parametros el resto de la primera lista, la segunda lista entera y la lista resultado. El caso base es que la primera lista este vacia, en ese caso la lista resultado es la segunda lista.

```
    append([],Ys,Ys) :-
        list(Ys).
    append([X|Xs],Ys,[X|Zs]) :-
        append(Xs,Ys,Zs).
```

Usage: `append(X,Y,Z)`

`Z` es el resultado de concatenar `X` y `Y`

invert/2: PREDICATE

Se extrae el primer elemento de la lista y se hace una llamada recursiva, a continuacion la lista obtenida como resultadose concatena con el elemento extraido. El caso base son las dos listas vacias.

```
invert([],[]).
invert([X|L],R) :-
    invert(L,L1),
    append(L1,[X],R).
```

Usage: `invert(L,R)`

`R` es el resultado de invertir `L`

aux4/3: PREDICATE

Reduce en uno el valor del numero del primer argumento, descarta el primer bit de la lista y a continuacion se vuelve a llamr de forma recursiva. El caso base es el primer argumento con valor 0, en esta situacion se equipara el primer elemento de la lista de bits con el tercer argumento.

```
aux4(s(N),[_1|A],B) :-
    aux4(N,A,B).
aux4(0,[X|_1],X).
```

Usage: `aux4(N,B,BN)`

`N`: posicion del bit, `B`: byte del que extraer el bit, `L`: valor del bit.

hex_byte_list/1: PREDICATE

Extrae elementos de la lista, comprueba si son bytes hexadecimales y se hace una llamada recursiva al predicado con el resto de la lista. El caso base es que la lista este vacia.

```
hex_byte_list([]).
hex_byte_list([A|L]) :-
    hex_byte(A),
    hex_byte_list(L).
```

Usage: `hex_byte_list(L)`

`L`: lista a comprobar si esta formada por bytes hexadecimales.

binary_byte_list/1: PREDICATE

Extrae elementos de la lista, comprueba si son bytes binarios y se hace una llamada recursiva al predicado con el resto de la lista. El caso base es que la lista este vacia.

```
binary_byte_list([]).
binary_byte_list([A|L]) :-
    binary_byte(A),
    binary_byte_list(L).
```

Usage: `binary_byte_list(L)`

`L`: lista a comprobar si esta formada por bytes binarios.

byte_to_bit/2: PREDICATE

Transforma una lista de bytes en una unica lista de bits o viceversa.

```
byte_to_bit([],[]).
byte_to_bit([[A,B,C,D,E,F,G,H]|By],[A,B,C,D,E,F,G,H|Bi]) :-
    byte_to_bit(By,Bi).
```

Usage: `byte_to_bit(Bytes,Bits)`

`Bytes`: Lista de bytes, `Bits`: Lista de bits.

xor_list/3: PREDICATE

Extrae elementos de las dos primeras listas, realiza la operacion xor sobre ellos, se almacena el resultado en el tercer argumento y se llama recursivamente a la funcion. El caso base es las una llamada con las tres listas vacias.

```
xor_list([E1|A],[E2|B],C) :-
    xor(E1,E2,R),
    xor_list(A,B,Acc),
    append([R],Acc,C).
xor_list([],[],[]).
```

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

`A`: Lista de bytes del primer operando, `B`: Lista de bytes del segundo operando, `C`: lista de bytes del resultado.

xor/3: PROPERTY

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

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

Dado un bit `A` y un bit `B` el resultado de la operacion xor en `C`.

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)