UTF-8

BITS, NIBBLES & BYTES in Prolog

Nicolas Cossio Miravalles, B190082

Table of Contents

code

This module defines the structure for bits, nibbles and bytes, and some operations

Definitions of structures:

Usage and interface

- **Library usage:**
  `:- use_module(/home/nicocossio/UPM/Prolog/code.pl).`
- **Exports:**
  - *Predicates:*
    `author_data/4`, `bind/1`, `binary_byte/1`, `hexd/1`, `hex_byte/1`, `byte/1`, `byte_list/1`, `nibble_bits/2`, `byte_conversion/2`, `byte_list_conversion/2`, `get_nth_bit_from_byte/3`, `byte_list_clsh/2`, `clsh/2`, `my_append/3`, `my_flattener/2`, `rotate_left/2`, `byte_list_crsh/2`, `crsh/2`, `conversor_xor/3`, `byte_xor/3`, `bin_xor/3`.

Documentation on exports

author_data/4: PREDICATE

No further documentation available for this predicate.

bind/1: PREDICATE

Usage: `bind(X)`

Bit representation where `X` is either 1 or 0.

```
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)])`

Checks whether the given list is a list of 8 binary digits.

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

hexd/1: PREDICATE

Usage: `hexd(X)`

Hex digit representation where `X` is ranges from 0 to 9 or a to f.

```
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).
```

hex_byte/1: PREDICATE

Usage: `hex_byte(X)`

Hexadecimal byte representation where `X` is a list containing two elements that are hexd/1.

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

byte/1: PREDICATE

Usage: `byte(X)`

Byte representation where `X` is either hexd/1 or X is true for binary_byte/1.

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

byte_list/1: PREDICATE

Usage: `byte_list(X)`

True when `X` is a list whose elements are true for byte/1.

```
byte_list([]).
byte_list([Lh|Lt]) :-
    byte(Lh),
    byte_list(Lt).
```

nibble_bits/2: PREDICATE

Usage: `nibble_bits(X,Y)`

True when `X` is the hexd/1 representation for `Y` which is list of 4 elements bind/1.

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

byte_conversion/2: PREDICATE

Usage: `byte_conversion(X,Y)`

True when `X` is a list of 2 hexd/1 elements and which is the equivalent for `Y` which is list of 8 elements bind/1.

```
byte_conversion([hexd(H1),hexd(H2)],[bind(B1),bind(B2),bind(B3),bind(B4),bind(B5
    byte([hexd(H1),hexd(H2)]),
    nibble_bits(hexd(H1),[bind(B1),bind(B2),bind(B3),bind(B4)]),
    nibble_bits(hexd(H2),[bind(B5),bind(B6),bind(B7),bind(B8)]).
```

byte_list_conversion/2: PREDICATE

Usage: `byte_list_conversion(X,Y)`

True when `X` is a list of lists whose elements are or lists of 2 hexd/1 elements, `Y` is a list of lists whose elements are lists of 8 bind/1 elements which are the equivalent of `X` .

```
byte_list_conversion([],[]).
byte_list_conversion([FirstList|Rest1],[SecondList|Rest2]) :-
    byte_conversion(FirstList,SecondList),
    byte_list_conversion(Rest1,Rest2).
```

get_nth_bit_from_byte/3: PREDICATE

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

True when `Nth` is the bind/1, in the position given by `N` which is in peano number format, starting counting from the least significant bit in the `B` byte/1.

```
get_nth_bit_from_byte(s(s(s(s(s(s(s(0))))))),[Nth|Tail],Nth).
get_nth_bit_from_byte(N,B,Nth) :-
    hex_byte(B),
    byte_conversion(B,X),
    get_nth_bit_from_byte(N,X,Nth).
get_nth_bit_from_byte(N,[Head|Tail],Nth) :-
    get_nth_bit_from_byte(s(N),Tail,Nth).
```

byte_list_clsh/2: PREDICATE

Usage: `byte_list_clsh(X,Y)`

True when `X` is the left circularly shifted list of lists of byte/1 of `Y`.

```
    byte_list_clsh([First|ByteList],Res) :-
        hex_byte(First),
        byte_list_conversion([First|ByteList],X),
        clsh(X,X2),
        byte_list_conversion(Res,X2).
    byte_list_clsh([First|ByteList],Res) :-
        binary_byte(First),
        clsh([First|ByteList],Res).
```

clsh/2: PREDICATE

Usage: `clsh(X,Y)`

```
    clsh(ByteList,Res) :-
        my_flattener(ByteList,Flattened),
        rotate_left(Flattened,Rotated),
        my_flattener(Res,Rotated).
```

my_append/3: PREDICATE

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

True when `Z` is the list composed by `Y` whose last element is `X`.

my_flattener/2: PREDICATE

Usage: `my_flattener(X,Y)`

True when `Y` is the list composed by all elements inside the lists of lists which composes `Y`.

rotate_left/2: PREDICATE

Usage: `rotate_left(X,Y)`

True when `Y` is the leftly circularly shifted list `X`.

```
    rotate_left([Head|Tail],Rotated) :-
        my_append(Tail,[Head],Rotated).
```

byte_list_crsh/2: PREDICATE

Usage: `byte_list_crsh(X,Y)`

True when `X` is the right circularly shifted list of lists of byte/1 of `Y`.

```
byte_list_crsh([First|ByteList],Res) :-
    hex_byte(First),
    byte_list_conversion([First|ByteList],X),
    crsh(X,X2),
    byte_list_conversion(Res,X2).
byte_list_crsh([First|ByteList],Res) :-
    binary_byte(First),
    crsh([First|ByteList],Res).
```

crsh/2: PREDICATE

Usage: `crsh(X,Y)`

```
crsh(ByteList,Res) :-
    my_flattener(ByteList,Flattened),
    rotate_left(Rotated,Flattened),
    my_flattener(Res,Rotated).
```

conversor_xor/3: PREDICATE

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

True when `Z` is the result of the binary XOR operation between `X` and `Y` which are bind/1.

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

byte_xor/3: PREDICATE

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

True when `Z` is the result of the binary XOR operation between `X` and `Y` which are bytes/1.

```
byte_xor(B1,B2,B) :-
    binary_byte(B1),
    binary_byte(B2),
    bin_xor(B1,B2,B).
byte_xor(B1,B2,H) :-
    hex_byte(B1),
    hex_byte(B2),
    byte_conversion(B1,X1),
    byte_conversion(B2,X2),
    bin_xor(X1,X2,B3),
    byte_conversion(H,B3).
```

bin_xor/3: PREDICATE

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

```
byte_xor(B1,B2,B) :-
    binary_byte(B1),
    binary_byte(B2),
    bin_xor(B1,B2,B).
byte_xor(B1,B2,H) :-
    hex_byte(B1),
    hex_byte(B2),
    byte_conversion(B1,X1),
    byte_conversion(B2,X2),
    bin_xor(X1,X2,B3),
    byte_conversion(H,B3).
```

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)