The Bit-vector Library

0.3.0

Generated by Doxygen 1.7.6.1

Tue Apr 23 2013 22:31:44

Contents

Chapter 1

C Data Structure Library: Bit-vector Library

Version

0.3.0

Author

Jun Woong (woong.jun at gmail.com)

Date

last modified on 2013-04-19

1.1 Introduction

This document specifies the Bit-vector Library which belongs to the C Data Structure Library. The basic structure is from David Hanson's book, "C Interfaces and Implementations." I modifies the original implementation to make it more appropriate for my other projects and to enhance its readibility; for example a prefix is used more strictly in order to avoid the user namespace pollution.

Since the book explains its design and implementation in a very comprehensive way, not to mention the copyright issues, it is nothing but waste to repeat it here, so I finish this document by giving introduction to the library; how to use the facilities is deeply explained in files that define them.

The Bit-vector Library reserves identifiers starting with `bitv_` and `BITV_`, and imports the Assertion Library (which requires the Exception Handling Library) and the Memory Management Library.

1.2 How to Use The Library

The Bit-vector Library implements a bit-vector that is a set of integers. A unsigned integer type like `unsigned long` or a bit-field in a struct or union is often used to represent such a set, and various bitwise operators serve set operations; for example, the bitwise OR operator effectively provides a union operation. This approach, however, imposes a restriction that the size of a set should be limited by that of the primitive integer type chosen. This Bit-vector Library gets rid of such a limit and allows users to use a set of integers with an arbitrary size at the cost of dynamic memory allocation and a more complex data structure than a simple integer type.

Similarly for other data structure libraries, use of the Bit-vector Library follows this sequence: create, use and destroy.

In general, a null pointer given to an argument expecting a bit-vector is considered an error which results in an assertion failure, but the functions for set operations (bitv_union(), bitv_inter(), bitv_minus() and bitv_diff()) take a null pointer as a valid argument and treat it as representing an empty (all-bits-cleared) bit-vector. Also note that they always produce a distinct bit-vector; none of them alters the original set.

1.3 Boilerplate Code

Using a bit-vector starts with creating one using bitv_new(). There are other ways to create bit-vectors from an existing one with bitv_union(), bitv_inter(), bitv_minus() and bitv_diff() (getting a union, intersection, difference, symmetric difference of bit-vectors, respectively); all bit-vector creation functions allocate storage for a bit-vector to create and if no allocation is possible, an exception is raised instead of returning a failure indicator like a null pointer.

Once a bit-vector created, a bit in the vector can be set and cleared using bitv_put(). A sequence of bits also can be handled in group using bitv_set(), bitv_clear(), bitv_not() and bitv_setv(); unlike a generic set, the concept of a universe can be defined for integral sets, thus we can have a function for complement. set_get() inspects if a certain bit is set in a bit-vector, and bitv_length() gives the size (or the length) of a bit-vector while bitv_count() counts the number of bits set in a given bit-vector.

bitv_map() offers a way to apply some operations on every bit in a bit-vector; it takes a user-defined function and calls it for each of bits.

bitv_free() takes a bit-vector (to be precise, a pointer to a bit-vector) and releases the storage used to maintain it.

As an example, the following code creates two bit-vectors each of which has 60 bits. It then obtains two more from getting a union and intersection of them after setting bits in different ways. It ends with printing bits in each set using a user-provided function and destroying all vectors.

```
int len;
bitv_t *s, *t, *u, *v;
unsigned char a[] = { 0x42, 0x80, 0x79, 0x29, 0x54, 0x19, 0xFF };

s = bitv_new(60);
```

```
t = bitv_new(60);
len = bitv_length(s);

bitv_set(s, 10, 50);
bitv_setv(t, a, 7);

u = bitv_union(s, t);
v = bitv_inter(s, t);

bitv_map(s, print, &len);
bitv_map(t, print, &len);
bitv_map(u, print, &len);
bitv_map(v, print, &len);

bitv_free(&s);
bitv_free(&t);
bitv_free(&u);
bitv_free(&v);
```

where print() is defined as follows:

```
static void print(size_t n, int v, void *cl)
{
    printf("%s%d", (n > 0 && n % 8 == 0)? " ": "", v);
    if (n == *(int *)cl - 1)
        puts("");
}
```

1.4 Contact Me

Visit http://code.woong.org to get the lastest version of this library. Only a small portion of my homepage (http://www.woong.org) is maintained in English, thus one who is not good at Korean would have difficulty when navigating most of other pages served in Korean. If you think the information you are looking for is on pages written in Korean, do not hesitate to send me an email to ask for help.

Any comments about the library are welcomed. If you have a proposal or question on the library just email me, and I will reply as soon as possible.

1.5 Copyright

I do not wholly hold the copyright of this library; it is mostly held by David Hanson as stated in his book, "C: Interfaces and Implementations:"

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 bitv.c File Reference

Source for Bit-vector Library (CDSL)

`#include <stddef.h>` `#include <string.h>` `#include "cbl/memory.-h"` `#include "cbl/assert.h"` `#include "bitv.h"` Include dependency graph for bitv.c:

Functions

- bitv_t *() bitv_new (size_t len)

  creates a new bit-vector.
- void() bitv_free (bitv_t **pset)

  destroys a bit-vector.
- size_t() bitv_length (const bitv_t *set)

  returns the length of a bit-vector.
- size_t() bitv_count (const bitv_t *set)

  returns the number of bits set.
- int() bitv_get (const bitv_t *set, size_t n)

  gets a bit in a bit-vector.
- int() bitv_put (bitv_t *set, size_t n, int bit)

  changes the value of a bit in a bit-vector.
- void() bitv_set (bitv_t *set, size_t l, size_t h)

  sets bits to 1 in a bit-vector.
- void() bitv_clear (bitv_t *set, size_t l, size_t h)

  clears bits in a bit-vector.
- void() bitv_not (bitv_t *set, size_t l, size_t h)

  complements bits in a bit-vector.
- void() bitv_setv (bitv_t *set, unsigned char *v, size_t n)

sets bits in a bit-vector with bit patterns.

- void() bitv_map (bitv_t *set, void apply(size_t, int, void *), void *cl)

  calls a user-provided function for each bit in a bit-vector.
- int() bitv_eq (const bitv_t *s, const bitv_t *t)

  compares two bit-vectors for equality.
- int() bitv_leq (const bitv_t *s, const bitv_t *t)

  compares two bit-vectors for subset.
- int() bitv_lt (const bitv_t *s, const bitv_t *t)

  compares two bit-vectors for proper subset.
- bitv_t *() bitv_union (const bitv_t *t, const bitv_t *s)

  returns a union of two bit-vectors.
- bitv_t *() bitv_inter (const bitv_t *t, const bitv_t *s)

  returns an intersection of two bit-vectors.
- bitv_t *() bitv_minus (const bitv_t *t, const bitv_t *s)

  returns a difference of two bit-vectors.
- bitv_t *() bitv_diff (const bitv_t *t, const bitv_t *s)

  returns a symmetric difference of two bit-vectors.

3.1.1 Detailed Description

Source for Bit-vector Library (CDSL)

3.1.2 Function Documentation

3.1.2.1 void() bitv_clear (bitv_t * *set*, size_t *l*, size_t *h*)

clears bits in a bit-vector.

bitv_clear() clears bits in a specified range of a bit-vector. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector to set
`in`	*l*	lower bound of range (inclusive)
`in`	*h*	upper bound of range (inclusive)

Returns

nothing

3.1.2.2 size_t() bitv_count (const bitv_t * *set*)

returns the number of bits set.

bitv_count() returns the number of bits set in a bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector to count

Returns

number of bits set

3.1.2.3 bitv_t*() bitv_diff (const bitv_t * *t*, const bitv_t * *s*)

returns a symmetric difference of two bit-vectors.

bitv_diff() returns a symmetric difference of two bit-vectors of the same length; the bit in the resulting bit-vector is set if only one of the corresponding bits in the operands is set. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_diff() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	operand of difference operation
in	*t*	operand of difference operation

Returns

symmetric difference of bit-vectors

Here is the call graph for this function:

3.1.2.4 int() bitv_eq (const bitv_t * *s*, const bitv_t * *t*)

compares two bit-vectors for equality.

bitv_eq() compares two bit-vectors to check whether they are equal. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether or not two bit-vectors compare equal

Return values

0	not equal
1	equal

3.1.2.5 void() bitv_free (bitv_t ** *pset*)

destroys a bit-vector.

bitv_free() destroys a bit-vector by deallocating the storage for it and set a given pointer to a null pointer.

Possible exceptions: assert_exceptfail

Unchecked errors: pointer to foreign data structure given for `pset`

Parameters

in,out	*pset*	pointer to bit-vector to destroy

Returns

nothing

3.1.2.6 int() bitv_get (const bitv_t * *set*, size_t *n*)

gets a bit in a bit-vector.

bitv_get() inspects whether a bit in a bit-vector is set or not. The position of a bit to inspect, `n` starts at 0 and must be smaller than the length of the bit-vector to inspect.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector to inspect
in	*n*	bit position

Returns

bit value (0 or 1)

3.1.2.7 bitv_t*() bitv_inter (const bitv_t * t, const bitv_t * s)

returns an intersection of two bit-vectors.

bitv_inter() creates an intersection of two bit-vectors of the same length and returns it. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_inter() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	operand of intersection operation
in	*t*	operand of intersection operation

Returns

intersection of bit-vectors

Here is the call graph for this function:

3.1.2.8 size_t() bitv_length (const bitv_t * set)

returns the length of a bit-vector.

bitv_length() returns the length of a bit-vector which is the number of bits in a bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector whose legnth returned

Returns

length of bit-vector

3.1.2.9 int() bitv_leq (const bitv_t * s, const bitv_t * t)

compares two bit-vectors for subset.

bitv_leq() compares two bit-vectors to check whether a bit-vector is a subset of the other; note that two bit-vectors have a subset relationship for each other when they compare equal. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether `s` is a subset of `t`

Return values

0	`s` is not a subset of `t`
1	`s` is a subset of `t`

3.1.2.10 int() bitv_lt (const bitv_t * *s*, const bitv_t * *t*)

compares two bit-vectors for proper subset.

bitv_lt() compares two bit-vectors to check whether a bit-vector is a proper subset of the other. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether `s` is a proper subset of `t`

Return values

0	`s` is not a proper subset of `t`
1	`s` is a proper subset of `t`

3.1.2.11 void() bitv_map (bitv_t * *set*, void *applysize_t, int, void* *, void * *cl*)

calls a user-provided function for each bit in a bit-vector.

For each bit in a bit-vector, bitv_map() calls a user-provided callback function; it is useful when doing some common task for each bit. The pointer given in `cl` is passed to the third parameter of a user callback function, so can be used as a communication channel between the caller of bitv_map() and the callback. Differently from mapping functions for other data structures (e.g., tables and sets), changes made in an earlier invocation to `apply` are visible to later invocations.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector with which `apply` called
`in`	*apply*	user-provided function (callback)
`in`	*cl*	passing-by argument to `apply`

Returns

nothing

3.1.2.12 bitv_t*() bitv_minus (const bitv_t * *t*, const bitv_t * *s*)

returns a difference of two bit-vectors.

bitv_minus() returns a difference of two bit-vectors of the same length; the bit in the resulting bit-vector is set if and only if the corresponding bit in the first operand is set and the corresponding bit in the second operand is not set. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_minus() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

`in`	*s*	operand of difference operation
`in`	*t*	operand of difference operation

Returns

difference of bit-vectors, `s - t`

Here is the call graph for this function:

3.1.2.13 bitv_t*() bitv_new (size_t *len*)

creates a new bit-vector.

bitv_new() creates a new bit-vector. Since a bit-vector has a much simpler data structure than a set (provided by `cdsl/set`) does, the only information that bitv_new() needs to create a new instance is the length of the bit vector it will create; bitv_new() will create a bit-vector with `length` bits. The length cannot be changed after creation.

Possible exceptions: mem_exceptfail

Parameters

`in`	*len*	length of bit-vector to create

new bit-vector created

Here is the caller graph for this function:

3.1.2.14 void() bitv_not (bitv_t * *set*, size_t *l*, size_t *h*)

complements bits in a bit-vector.

bitv_not() flips bits in a specified range of a bit-vector. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector to set
`in`	*l*	lower bound of range (inclusive)
`in`	*h*	upper bound of range (inclusive)

Returns

nothing

3.1.2.15 int() bitv_put (bitv_t * *set*, size_t *n*, int *bit*)

changes the value of a bit in a bit-vector.

bitv_put() changes the value of a bit in a bit-vector to 0 or 1 and returns its previous value. The position of a bit to change, `n` starts at 0 and must be smaller than the length of the bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in,out	*set*	bit-vector to set
in	*n*	bit position
in	*bit*	value

Returns

previous value of bit

3.1.2.16 void() bitv_set (bitv_t * *set*, size_t *l*, size_t *h*)

sets bits to 1 in a bit-vector.

bitv_set() sets bits in a specified range of a bit-vector to 1. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in,out	*set*	bit-vector to set
in	*l*	lower bound of range (inclusive)
in	*h*	upper bound of range (inclusive)

Returns

nothing

3.1.2.17 void() bitv_setv (bitv_t * *set*, unsigned char * *v*, size_t *n*)

sets bits in a bit-vector with bit patterns.

bitv_setv() copies bit patterns from an array of bytes to a bit vector. Because only 8 bits in a byte are used to represent a bit-vector for table-driven approaches, any excess bits are masked out before copying, which explains why bitv_setv() needs to modify the array, `v`.

Be careful with how to count bit positions in a bit vector. Within a byte, the first bit (the bit position 0) indicates the least significant bit of a byte and the last bit (the bit position 7) does the most significant bit. Therefore, the array

```
{ 0x01, 0x02, 0x03, 0x04 }
```

can be used to set bits of a bit-vector as follows:

```
0        8        16       24     31
```

```
|        |        |        |      |
10000000 01000000 11000000 00100000
```

where the bit position is shown on the first line.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`, invalid pointer given for `v`, invalid size given for `n`

Parameters

`in,out`	*set*	bit-vector to set
`in,out`	*v*	bit patterns to use
`in`	*n*	size of `v` in bytes

Returns

nothing

3.1.2.18 bitv_t*() bitv_union (const bitv_t * *t*, const bitv_t * *s*)

returns a union of two bit-vectors.

bitv_union() creates a union of two given bit-vectors of the same length and returns it. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_union() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its results even when one of the operands is empty. This property can be used to make a copy of a bit-vector as follows:

```
bitv_t *v *w;
v = bitv_new(n);
...
w = bitv_union(v, NULL);
```

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

`in`	*s*	operand of union operation
`in`	*t*	operand of union operation

Returns

union of bit-vectors

3.2 bitv.h File Reference

Header for Bit-vector Library (CDSL)

`#include <stddef.h>` Include dependency graph for bitv.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef struct bitv_t bitv_t
  represents a bit-vector.

Functions

bit-vector creating/destroying functions:

- bitv_t * bitv_new (size_t)
  creates a new bit-vector.
- void bitv_free (bitv_t **)
  destroys a bit-vector.

data/information retrieving functions:

- size_t bitv_length (const bitv_t *)
  returns the length of a bit-vector.
- size_t bitv_count (const bitv_t *)
  returns the number of bits set.
- int bitv_get (const bitv_t *, size_t)
  gets a bit in a bit-vector.
- int bitv_put (bitv_t *, size_t, int)
  changes the value of a bit in a bit-vector.
- void bitv_set (bitv_t *, size_t, size_t)
  sets bits to 1 in a bit-vector.
- void bitv_clear (bitv_t *, size_t, size_t)
  clears bits in a bit-vector.
- void bitv_not (bitv_t *, size_t, size_t)
  complements bits in a bit-vector.
- void bitv_setv (bitv_t *, unsigned char *, size_t)
  sets bits in a bit-vector with bit patterns.

bit-vector handling functions:

- void **bitv_map** (bitv_t *, void(size_t, int, void *), void *)

bit-vector comparison functions:

- int bitv_eq (const bitv_t *, const bitv_t *)
  compares two bit-vectors for equality.
- int bitv_leq (const bitv_t *, const bitv_t *)
  compares two bit-vectors for subset.
- int bitv_lt (const bitv_t *, const bitv_t *)
  compares two bit-vectors for proper subset.

set operation functions:

- bitv_t * bitv_union (const bitv_t *, const bitv_t *)
  returns a union of two bit-vectors.
- bitv_t * bitv_inter (const bitv_t *, const bitv_t *)
  returns an intersection of two bit-vectors.
- bitv_t * bitv_minus (const bitv_t *, const bitv_t *)
  returns a difference of two bit-vectors.
- bitv_t * bitv_diff (const bitv_t *, const bitv_t *)
  returns a symmetric difference of two bit-vectors.

3.2.1 Detailed Description

Header for Bit-vector Library (CDSL) Documentation for Bit-vector Library (CDSL)

3.2.2 Function Documentation

3.2.2.1 void bitv_clear (bitv_t * *set*, size_t *l*, size_t *h*)

clears bits in a bit-vector.

bitv_clear() clears bits in a specified range of a bit-vector. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector to set
`in`	*l*	lower bound of range (inclusive)
`in`	*h*	upper bound of range (inclusive)

Returns

nothing

3.2.2.2 size_t bitv_count (const bitv_t * *set*)

returns the number of bits set.

bitv_count() returns the number of bits set in a bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector to count

Returns

number of bits set

3.2.2.3 bitv_t* bitv_diff (const bitv_t * *t*, const bitv_t * *s*)

returns a symmetric difference of two bit-vectors.

bitv_diff() returns a symmetric difference of two bit-vectors of the same length; the bit in the resulting bit-vector is set if only one of the corresponding bits in the operands is set. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_diff() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	operand of difference operation
in	*t*	operand of difference operation

Returns

symmetric difference of bit-vectors

Here is the call graph for this function:

3.2.2.4 int bitv_eq (const bitv_t * *s*, const bitv_t * *t*)

compares two bit-vectors for equality.

bitv_eq() compares two bit-vectors to check whether they are equal. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether or not two bit-vectors compare equal

Return values

0	not equal
1	equal

3.2.2.5 void bitv_free (bitv_t ** *pset*)

destroys a bit-vector.

bitv_free() destroys a bit-vector by deallocating the storage for it and set a given pointer to a null pointer.

Possible exceptions: assert_exceptfail

Unchecked errors: pointer to foreign data structure given for `pset`

Parameters

in,out	*pset*	pointer to bit-vector to destroy

Returns

nothing

3.2.2.6 int bitv_get (const bitv_t * *set*, size_t *n*)

gets a bit in a bit-vector.

bitv_get() inspects whether a bit in a bit-vector is set or not. The position of a bit to inspect, `n` starts at 0 and must be smaller than the length of the bit-vector to inspect.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector to inspect
in	*n*	bit position

Returns

bit value (0 or 1)

3.2.2.7 bitv_t* bitv_inter (const bitv_t * t, const bitv_t * s)

returns an intersection of two bit-vectors.

bitv_inter() creates an intersection of two bit-vectors of the same length and returns it. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_inter() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	operand of intersection operation
in	*t*	operand of intersection operation

Returns

intersection of bit-vectors

Here is the call graph for this function:

3.2.2.8 size_t bitv_length (const bitv_t * set)

returns the length of a bit-vector.

bitv_length() returns the length of a bit-vector which is the number of bits in a bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in	*set*	bit-vector whose legnth returned

Returns

length of bit-vector

3.2.2.9 int bitv_leq (const bitv_t * s, const bitv_t * t)

compares two bit-vectors for subset.

bitv_leq() compares two bit-vectors to check whether a bit-vector is a subset of the other; note that two bit-vectors have a subset relationship for each other when they compare equal. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether `s` is a subset of `t`

Return values

0	`s` is not a subset of `t`
1	`s` is a subset of `t`

3.2.2.10 int bitv_lt (const bitv_t * *s*, const bitv_t * *t*)

compares two bit-vectors for proper subset.

bitv_lt() compares two bit-vectors to check whether a bit-vector is a proper subset of the other. Two bit-vectors must be of the same length.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	bit-vector to compare
in	*t*	bit-vector to compare

Returns

whether `s` is a proper subset of `t`

Return values

0	`s` is not a proper subset of `t`
1	`s` is a proper subset of `t`

3.2.2.11 bitv_t* bitv_minus (const bitv_t * *t*, const bitv_t * *s*)

returns a difference of two bit-vectors.

bitv_minus() returns a difference of two bit-vectors of the same length; the bit in the resulting bit-vector is set if and only if the corresponding bit in the first operand is set and the corresponding bit in the second operand is not set. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_minus() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its result even when one of the operands is empty.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

`in`	*s*	operand of difference operation
`in`	*t*	operand of difference operation

Returns

difference of bit-vectors, `s - t`

Here is the call graph for this function:

3.2.2.12 bitv_t* bitv_new (size_t *len*)

creates a new bit-vector.

bitv_new() creates a new bit-vector. Since a bit-vector has a much simpler data structure than a set (provided by `cdsl/set`) does, the only information that bitv_new() needs to create a new instance is the length of the bit vector it will create; bitv_new() will create a bit-vector with `length` bits. The length cannot be changed after creation.

Possible exceptions: mem_exceptfail

Parameters

`in`	*len*	length of bit-vector to create

new bit-vector created

Here is the caller graph for this function:

3.2.2.13 void bitv_not (bitv_t * *set*, size_t *l*, size_t *h*)

complements bits in a bit-vector.

bitv_not() flips bits in a specified range of a bit-vector. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector to set
`in`	*l*	lower bound of range (inclusive)
`in`	*h*	upper bound of range (inclusive)

Returns

nothing

3.2.2.14 int bitv_put (bitv_t * *set*, size_t *n*, int *bit*)

changes the value of a bit in a bit-vector.

bitv_put() changes the value of a bit in a bit-vector to 0 or 1 and returns its previous value. The position of a bit to change, `n` starts at 0 and must be smaller than the length of the bit-vector.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

`in,out`	*set*	bit-vector to set
`in`	*n*	bit position
`in`	*bit*	value

Returns

previous value of bit

3.2.2.15 void bitv_set (bitv_t * *set*, size_t *l*, size_t *h*)

sets bits to 1 in a bit-vector.

bitv_set() sets bits in a specified range of a bit-vector to 1. The inclusive lower bound `l` and the inclusive upper bound `h` specify the range. `l` must be equal to or smaller than `h` and `h` must be smaller than the length of the bit-vector to set.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`

Parameters

in,out	*set*	bit-vector to set
in	*l*	lower bound of range (inclusive)
in	*h*	upper bound of range (inclusive)

Returns

nothing

3.2.2.16 void bitv_setv (bitv_t * *set*, unsigned char * *v*, size_t *n*)

sets bits in a bit-vector with bit patterns.

bitv_setv() copies bit patterns from an array of bytes to a bit vector. Because only 8 bits in a byte are used to represent a bit-vector for table-driven approaches, any excess bits are masked out before copying, which explains why bitv_setv() needs to modify the array, `v`.

Be careful with how to count bit positions in a bit vector. Within a byte, the first bit (the bit position 0) indicates the least significant bit of a byte and the last bit (the bit position 7) does the most significant bit. Therefore, the array

```
{ 0x01, 0x02, 0x03, 0x04 }
```

can be used to set bits of a bit-vector as follows:

```
0        8        16       24      31
|        |        |        |       |
10000000 01000000 11000000 00100000
```

where the bit position is shown on the first line.

Possible exceptions: assert_exceptfail

Unchecked errors: foreign data structure given for `set`, invalid pointer given for `v`, invalid size given for `n`

Parameters

in,out	*set*	bit-vector to set
in,out	*v*	bit patterns to use
in	*n*	size of `v` in bytes

Returns

nothing

3.2.2.17 bitv_t* bitv_union (const bitv_t * t, const bitv_t * s)

returns a union of two bit-vectors.

bitv_union() creates a union of two given bit-vectors of the same length and returns it. One of those may be a null pointer, in which case it is considered an empty (all-cleared) bit-vector. bitv_union() constitutes a distinct bit-vector from its operands as a result, which means it always allocates storage for its results even when one of the operands is empty. This property can be used to make a copy of a bit-vector as follows:

```
bitv_t *v *w;
v = bitv_new(n);
...
w = bitv_union(v, NULL);
```

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: foreign data structure given for `s` or `t`

Parameters

in	*s*	operand of union operation
in	*t*	operand of union operation

Returns

union of bit-vectors