NAME

caesar_bitmap – the ‘‘bitmap’’ library of OPEN/CAESAR

PURPOSE

The ‘‘bitmap’’ library provides primitives for implementing the ‘‘bit state space’’ verification technique proposed by Gerard Holzmann.

USAGE

The ‘‘bitmap’’ library consists of:

- a predefined header file **caesar_bitmap.h**;
- the precompiled library file **libcaesar.a**, which implements the features described in **caesar_bitmap.h**.

Note: The ‘‘bitmap’’ library is a software layer built above the primitives offered by the ‘‘standard’’ and ‘‘hash’’ libraries.

DESCRIPTION

A ‘‘bitmap’’ of size N is basically an array of N bits numbered from 0 to N-1. The value of N is usually large (e.g., some tenth million states).

Additionally, statistics are attached to each bitmap. These statistics consist of a ‘‘success counter’’ (which counts how many bits equal to 1 have been read) and a ‘‘failure counter’’ (which counts how many bits equal to 0 have been read).

FEATURES

.....

CAESAR_TYPE_BITMAP

```
typedef CAESAR_TYPE_ABSTRACT (...) CAESAR_TYPE_BITMAP;
```

This type denotes a pointer to the concrete representation of a bitmap, which is supposed to be ‘‘opaque’’.

.....

CAESAR_CREATE_BITMAP

```
void CAESAR_CREATE_BITMAP (CAESAR_B, CAESAR_SIZE, CAESAR_PRIME)
   CAESAR_TYPE_BITMAP *CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_BOOLEAN CAESAR_PRIME;
   { ... }
```

This procedure allocates a bitmap using **CAESAR_CREATE()** and assigns its address to ∗**CAESAR_B**. The size N of this bitmap is determined by the values of formal parameters **CAESAR_SIZE** and **CAESAR_PRIME**, and also by the amount of memory available.

Note: because **CAESAR_TYPE_BITMAP** is a pointer type, any variable **CAESAR_B** of type **CAESAR_TYPE_BITMAP** must be allocated before used, for instance using:

```
CAESAR_CREATE_BITMAP (&CAESAR_B, ...)
```

If the value of **CAESAR_SIZE** is different from 0, then the number N of bits in the bitmap will be

CAESAR_SIZE.

If the value of **CAESAR_SIZE** is equal to 0, then N will be given a default value as large as possible.

Note: in this case, the bitmap will fill most of the memory space available for the current process. Therefore, if **CAESAR_CREATE_BITMAP()** is to be called with **CAESAR_SIZE** = 0, it should be called only after having allocated all the other data structures (e.g., stacks, ...), otherwise there may be not enough memory for these data structures.

In both cases, the value of N can be reduced to a smaller value as to fit into the amount of available memory.

If the value of **CAESAR_PRIME** is different from 0, the value of N can also be reduced to the immediately smaller prime number (since some hash functions require prime modulus). If the value of **CAESAR_PRIME** is equal to 0, the value of N is not changed.

If (in spite of various attempts) the allocation fails, the **NULL** value is assigned to ∗**CAESAR_B**.

If the allocation succeeds, the final value of N can be known using the function **CAESAR_SIZE_BITMAP()** (see below).

If the allocation succeeds, the N bits of the bitmap are initialized to 0. The success and failure counters attached to the bitmap are also initialized to 0.

Note: since variable **CAESAR_SIZE** is a value of type **CAESAR_TYPE_NATURAL**, a bitmap can contain at most 2^{8n}-1 bits, where:

n = **sizeof (CAESAR_TYPE_NATURAL)**

It is assumed that n >= 4. For n = 4, this makes 4,294,967,295 bits, that is approximately 537 Megabytes of memory, which is currently enough.

.....

CAESAR_DELETE_BITMAP

```
void CAESAR_DELETE_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP *CAESAR_B;
   { ... }
```

This procedure frees the memory space corresponding to the bitmap pointed to by ∗**CAESAR_B** using **CAESAR_DELETE()**. Afterwards, the **NULL** value is assigned to ∗**CAESAR_B**.

.....

CAESAR_PURGE_BITMAP

```
void CAESAR_PURGE_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This procedure empties the bitmap pointed to by **CAESAR_B** without deleting it. Afterwards, this bitmap is exactly in the same state as after its creation using **CAESAR_CREATE_BITMAP()**. Its size remains unchanged.

.....

CAESAR_SIZE_BITMAP

```
CAESAR_TYPE_NATURAL CAESAR_SIZE_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This function returns the size (i.e., number of bits) of the bitmap pointed to by **CAESAR_B**. This size is determined when the bitmap is created using **CAESAR_CREATE_BITMAP()** and remains constant.

.....

CAESAR_SET_BITMAP

```
void CAESAR_SET_BITMAP (CAESAR_B, CAESAR_I)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_I;
   { ... }
```

This procedure sets to 1 the **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B**. The value of **CAESAR_I** is such that:

0 <= **CAESAR_I** <= **CAESAR_SIZE_BITMAP (CAESAR_B)**

It is usually the result of some hash-code computation.

.....

CAESAR_RESET_BITMAP

```
void CAESAR_RESET_BITMAP (CAESAR_B, CAESAR_I)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_I;
   { ... }
```

This procedure sets to 0 the **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B**. The value of **CAESAR_I** is such that:

0 <= **CAESAR_I** <= **CAESAR_SIZE_BITMAP (CAESAR_B)**

It is usually the result of some hash-code computation.

.....

CAESAR_TEST_BITMAP

```
CAESAR_TYPE_BOOLEAN CAESAR_TEST_BITMAP (CAESAR_B, CAESAR_I)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_I;
   { ... }
```

This function returns 0 if the **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B** is equal to 0, or a value different from 0 if this bit is equal to 1. The value of **CAESAR_I** is such that:

0 <= **CAESAR_I** <= **CAESAR_SIZE_BITMAP (CAESAR_B)**

It is usually the result of some hash-code computation.

A return value of 0 increments the failure counter attached to the bitmap pointed to by **CAESAR_B**, whereas a return value of 1 increments the success counter.

.....

CAESAR_TEST_AND_SET_BITMAP

```
CAESAR_TYPE_BOOLEAN CAESAR_TEST_AND_SET_BITMAP (CAESAR_B, CAESAR_I)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_I;
   { ... }
```

This function returns 0 if the **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B** is equal to 0, or a value different from 0 if this bit is equal to 1. The value of **CAESAR_I** is such that:

0 <= **CAESAR_I** <= **CAESAR_SIZE_BITMAP (CAESAR_B)**

It is usually the result of some hash-code computation.

The **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B** is set to 1 if it was equal to 0.

A return value of 0 increments the failure counter attached to the bitmap pointed to by **CAESAR_B**, whereas a return value of 1 increments the success counter.

.....

CAESAR_TEST_AND_RESET_BITMAP

```
CAESAR_TYPE_BOOLEAN CAESAR_TEST_AND_RESET_BITMAP (CAESAR_B, CAESAR_I)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_NATURAL CAESAR_I;
   { ... }
```

This function returns 0 if the **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B** is equal to 0, or a value different from 0 if this bit is equal to 1. The value of **CAESAR_I** is such that:

0 <= **CAESAR_I** <= **CAESAR_SIZE_BITMAP (CAESAR_B)**

It is usually the result of some hash-code computation.

The **CAESAR_I**-th bit of the bitmap pointed to by **CAESAR_B** is set to 0 if it was equal to 1.

A return value of 0 increments the failure counter attached to the bitmap pointed to by **CAESAR_B**, whereas a return value of 1 increments the success counter.

.....

CAESAR_ZERO_BITMAP

```
CAESAR_TYPE_NATURAL CAESAR_ZERO_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This function returns the number of bits which are equal to 0 in the bitmap pointed to by **CAESAR_B**.

.....

CAESAR_ONE_BITMAP

```
CAESAR_TYPE_NATURAL CAESAR_ONE_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This function returns the number of bits which are equal to 1 in the bitmap pointed to by **CAESAR_B**.

Note: for any bitmap **CAESAR_B**:

CAESAR_ZERO_BITMAP (CAESAR_B) + **CAESAR_ONE_BITMAP (CAESAR_B)**

is equal to:

CAESAR_SIZE_BITMAP (CAESAR_B)

.....

CAESAR_FAILURE_BITMAP

```
CAESAR_TYPE_NATURAL CAESAR_FAILURE_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This function returns the value of the failure counter of the bitmap pointed to by **CAESAR_B**, i.e., the number of searches that failed (see functions **CAESAR_TEST_BITMAP()**, **CAESAR_TEST_AND_SET_BITMAP()**, and **CAESAR_TEST_AND_RESET()** above).

.....

CAESAR_SUCCESS_BITMAP

```
CAESAR_TYPE_NATURAL CAESAR_SUCCESS_BITMAP (CAESAR_B)
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This function returns the value of the success counter of the bitmap pointed to by **CAESAR_B**, i.e., the number of searches that succeeded (see functions **CAESAR_TEST_BITMAP()**, **CAESAR_TEST_AND_SET_BITMAP()**, and **CAESAR_TEST_AND_RESET()** above).

.....

CAESAR_FORMAT_BITMAP

```
CAESAR_TYPE_FORMAT CAESAR_FORMAT_BITMAP (CAESAR_B, CAESAR_FORMAT)
   CAESAR_TYPE_BITMAP CAESAR_B;
   CAESAR_TYPE_FORMAT CAESAR_FORMAT;
   { ... }
```

This function allows to control the format under which the bitmap pointed to by **CAESAR_B** will be printed by the procedure **CAESAR_PRINT_BITMAP()** (see below). Currently, the following formats are available:

- With format 0, statistical information about the bitmap is displayed such as: the size in bytes, the number of bits, the number of bits equal to 0, the number of bits equal to 1, the success counter, the failure counter, etc.

- With format 1, the contents of the bitmap are printed in hexadecimal. This can be useful for debugging bitmaps of small size.
- With format 2, the list of bits which are equal to 0 is printed. This can be useful for debugging bitmaps with almost all bits equal to 1.
- With format 3, the list of bits which are equal to 1 is printed. This can be useful for debugging bitmaps with almost all bits equal to 0.
- (no other format available yet).

By default, the current format of each bitmap is initialized to 0.

When called with **CAESAR_FORMAT** between 0 and 3, this fonction sets the current format of **CAESAR_B** to **CAESAR_FORMAT** and returns an undefined result.

When called with another value of **CAESAR_FORMAT**, this function does not modify the current format of **CAESAR_B** but returns a result defined as follows. If **CAESAR_FORMAT** is equal to the constant **CAESAR_CURRENT_FORMAT**, the result is the value of the current format of **CAESAR_B**. If **CAESAR_FORMAT** is equal to the constant **CAESAR_MAXIMAL_FORMAT**, the result is the maximal format value (i.e., 3). In all other cases, the effect of this function is undefined.

.....

CAESAR_MAX_FORMAT_BITMAP

```
CAESAR_TYPE_FORMAT CAESAR_MAX_FORMAT_BITMAP ()
   { ... }
```

Caution! This function is deprecated. It should no longer be used, as it might be removed from future versions of the *OPEN/CAESAR*. Use function **CAESAR_FORMAT_BITMAP()** instead, called with argument **CAESAR_MAXIMAL_FORMAT**.

This function returns the maximal format value available for printing bitmaps.

.....

CAESAR_PRINT_BITMAP

```
void CAESAR_PRINT_BITMAP (CAESAR_FILE, CAESAR_B)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_BITMAP CAESAR_B;
   { ... }
```

This procedure prints to file **CAESAR_FILE** a character string containing information about the bitmap pointed to by **CAESAR_B**. The nature of the information is determined by the current format of the bitmap pointed to by **CAESAR_B**.

Before this procedure is called, **CAESAR_FILE** must have been properly opened, for instance using **fopen(3)**.

.....

AUTHOR(S)

Hubert Garavel

FILES

$CADP/incl/caesar_graph.h	interface of the graph module
$CADP/incl/caesar_*.h	interfaces of the storage module
$CADP/bin.‘arch‘/libcaesar.a	object code of the storage module

$CADP/src/open_caesar/*.c	source code of various exploration modules
$CADP/com/lotos.open	shell script to run OPEN/CAESAR

SEE ALSO

Reference Manuals of OPEN/CAESAR, CAESAR, and CAESAR.ADT, **lotos.open**(LOCAL), **caesar**(LOCAL), **caesar.adt**(LOCAL)

Additional information is available from the CADP Web page located at http://cadp.inria.fr

Directives for installation are given in files **$CADP/INSTALLATION_*.**

Recent changes and improvements to this software are reported and commented in file **$CADP/HISTORY.**

BUGS

Known bugs are described in the Reference Manual of OPEN/CAESAR. Please report new bugs to cadp@inria.fr