NAME

caesar_hash – the ‘‘hash’’ library of OPEN/CAESAR

PURPOSE

The ‘‘hash’’ library provides primitives for computing various hashing functions on byte strings, states and labels.

USAGE

The ‘‘hash’’ library consists of:

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

Note: The ‘‘hash’’ library is a software layer built above the primitives offered by the ‘‘standard’’ library and by the *OPEN/CAESAR* graph module.

FEATURES

....

CAESAR_PRIME_HASH

```
CAESAR_TYPE_BOOLEAN CAESAR_PRIME_HASH (CAESAR_I)
  CAESAR_TYPE_NATURAL CAESAR_I;
  { ... }
```

This function returns a value different from 0 if **`CAESAR_I`** is a prime number, or 0 otherwise.

....

CAESAR_0_HASH

```
CAESAR_TYPE_NATURAL CAESAR_0_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the byte string of length **`CAESAR_SIZE`** pointed to by **`CAESAR_P`**, and returns this value, which is in the range 0..(**`CAESAR_MODULUS`**-1). If either **`CAESAR_SIZE`** or **`CAESAR_MODULUS`** is equal to 0, the result is undefined.

Note: The string is exactly **`CAESAR_SIZE`** bytes long, starting from **`CAESAR_P`** [0] to **`CAESAR_P`** [**`CAESAR_SIZE`** - 1]. It is not required that the byte string is terminated by a null byte (which is not the case of character strings in C). The same remark applies to the other functions defined below.

....

CAESAR_1_HASH

```
CAESAR_TYPE_NATURAL CAESAR_1_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
```

```
    CAESAR_TYPE_NATURAL CAESAR_MODULUS;
    { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: This function is derived from the hash function **s_hash()** formerly used in the SPIN validation system (see Gerard Holzmann's book ''Design and Validation of Computer Protocols'', 1991, page 307). The result of **s_hash()** is also the same as the one returned in global variable **J1** by the other hash function **d_hash()**. The **CAESAR_1_HASH()** function is more general than **s_hash()** since it was extended to 64-bit machines and since it does not assume that **CAESAR_SIZE** is a power of four and that **CAESAR_MODULUS** is a power of two. It also attempts to solve portability issues that occur in **s_hash()**. Yet, it may be slower than **s_hash()**.

Note: The result returned by this function may differ accross different machines, since it depends on machine endianness.

....

CAESAR_2_HASH

```
CAESAR_TYPE_NATURAL CAESAR_2_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
    CAESAR_TYPE_POINTER CAESAR_P;
    CAESAR_TYPE_NATURAL CAESAR_SIZE;
    CAESAR_TYPE_NATURAL CAESAR_MODULUS;
    { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: This function is derived from the hash function **d_hash()** formerly used in the SPIN validation system (see Gerard Holzmann's book ''Design and Validation of Computer Protocols'', 1991, page 307) with respect to the result returned in global variable **J2**. The **CAESAR_2_HASH()** function is more general than **d_hash()** since it was extended to 64-bit machines and since it does not assume that **CAESAR_SIZE** is a power of four and that **CAESAR_MODULUS** is a power of two. It also attempts to solve portability issues that occur in **d_hash()**. Yet, it may be slower than **d_hash()**.

Note: The result returned by this function may differ accross different machines, since it depends on machine endianness.

....

CAESAR_3_HASH

```
CAESAR_TYPE_NATURAL CAESAR_3_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
    CAESAR_TYPE_POINTER CAESAR_P;
    CAESAR_TYPE_NATURAL CAESAR_SIZE;
    CAESAR_TYPE_NATURAL CAESAR_MODULUS;
    { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

....

CAESAR_4_HASH

```
CAESAR_TYPE_NATURAL CAESAR_4_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: This function is derived from the PJW algorithm given in "Compilers: Principles, Techniques, and Tools" by Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, [pp. 434-438], which was used primarly for compressing character strings. The **CAESAR_4_HASH()** function removes a limitation of the original PJW algorithm, which always returns a hash value on 24 bits only. If **CAESAR_MODULUS** is larger than 24 bits, the result of **CAESAR_4_HASH()** might be also larger than 24 bits.

....

CAESAR_5_HASH

```
CAESAR_TYPE_NATURAL CAESAR_5_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: This function performs simple computations based upon XOR operations and bit shifts.

....

CAESAR_6_HASH

```
CAESAR_TYPE_NATURAL CAESAR_6_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

....

CAESAR_7_HASH

```
CAESAR_TYPE_NATURAL CAESAR_7_HASH (CAESAR_P, CAESAR_SIZE, CAESAR_MODULUS)
   CAESAR_TYPE_POINTER CAESAR_P;
   CAESAR_TYPE_NATURAL CAESAR_SIZE;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the byte string of length **CAESAR_SIZE** pointed to by **CAESAR_P**, and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If either **CAESAR_SIZE** or **CAESAR_MODULUS** is equal to 0, the result is undefined.

Note: This function is derived from the hash functions proposed by Bob Jenkins, namely **lookup2()** for 32-bit machines and **lookup8()** for 64-bit machines.

....

CAESAR_STATE_0_HASH

```
CAESAR_TYPE_NATURAL CAESAR_STATE_0_HASH (CAESAR_S, CAESAR_MODULUS)
   CAESAR_TYPE_STATE CAESAR_S;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the state pointed to by **CAESAR_S** and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If **CAESAR_MODULUS** is equal to 0, the result is undefined.

This function is defined by the following equality:

```
CAESAR_STATE_0_HASH (CAESAR_S, CAESAR_MODULUS) =
CAESAR_HASH_STATE (CAESAR_S, CAESAR_MODULUS)
```

where **CAESAR_HASH_STATE()** is the hashing function exported by the graph module.

....

CAESAR_STATE_1_HASH

```
CAESAR_TYPE_NATURAL CAESAR_STATE_1_HASH (CAESAR_S, CAESAR_MODULUS)
   CAESAR_TYPE_STATE CAESAR_S;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the state pointed to by **CAESAR_S** and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If **CAESAR_MODULUS** is equal to 0, the result is undefined.

This function is defined by the following equality:

```
CAESAR_STATE_1_HASH (CAESAR_S, CAESAR_MODULUS) =
CAESAR_1_HASH (CAESAR_S, CAESAR_HASH_SIZE_STATE (), CAESAR_MODULUS)
```

Additional functions

```
CAESAR_STATE_2_HASH (CAESAR_S, CAESAR_MODULUS)
CAESAR_STATE_3_HASH (CAESAR_S, CAESAR_MODULUS)
CAESAR_STATE_4_HASH (CAESAR_S, CAESAR_MODULUS)
CAESAR_STATE_5_HASH (CAESAR_S, CAESAR_MODULUS)
CAESAR_STATE_6_HASH (CAESAR_S, CAESAR_MODULUS)
CAESAR_STATE_7_HASH (CAESAR_S, CAESAR_MODULUS)
```

are defined in the same way as **CAESAR_STATE_1_HASH()**.

....

CAESAR_LABEL_0_HASH

```
CAESAR_TYPE_NATURAL CAESAR_LABEL_0_HASH (CAESAR_L, CAESAR_MODULUS)
   CAESAR_TYPE_LABEL CAESAR_L;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the label pointed to by **CAESAR_L** and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If **CAESAR_MODULUS** is equal to 0, the result is undefined.

This function is defined by the following equality:

```
CAESAR_LABEL_0_HASH (CAESAR_L, CAESAR_MODULUS) =
CAESAR_HASH_LABEL (CAESAR_L, CAESAR_MODULUS)
```

where **CAESAR_HASH_LABEL()** is the hashing function exported by the graph module.

....

CAESAR_LABEL_1_HASH

```
CAESAR_TYPE_NATURAL CAESAR_LABEL_1_HASH (CAESAR_L, CAESAR_MODULUS)
   CAESAR_TYPE_LABEL CAESAR_L;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the label pointed to by **CAESAR_L** and returns this value, which is in the range 0..(**CAESAR_MODULUS**-1). If **CAESAR_MODULUS** is equal to 0, the result is undefined.

This function is defined by the following equality:

```
CAESAR_LABEL_1_HASH (CAESAR_L, CAESAR_MODULUS) =
CAESAR_1_HASH (CAESAR_L, CAESAR_HASH_SIZE_LABEL (), CAESAR_MODULUS)
```

Additional functions

```
CAESAR_LABEL_2_HASH (CAESAR_L, CAESAR_MODULUS)
```

```
    CAESAR_LABEL_3_HASH (CAESAR_L, CAESAR_MODULUS)
    CAESAR_LABEL_4_HASH (CAESAR_L, CAESAR_MODULUS)
    CAESAR_LABEL_5_HASH (CAESAR_L, CAESAR_MODULUS)
    CAESAR_LABEL_6_HASH (CAESAR_L, CAESAR_MODULUS)
    CAESAR_LABEL_7_HASH (CAESAR_L, CAESAR_MODULUS)
```

are defined in the same way as **`CAESAR_LABEL_1_HASH()`**.

....

`CAESAR_STRING_0_HASH`

```
CAESAR_TYPE_NATURAL CAESAR_STRING_0_HASH (CAESAR_S, CAESAR_MODULUS)
   CAESAR_TYPE_STRING CAESAR_S;
   CAESAR_TYPE_NATURAL CAESAR_MODULUS;
   { ... }
```

This function computes a hash-value for the (variable-length, null-terminated) character string pointed to by **`CAESAR_S`** and returns this value, which is in the range 0..(**`CAESAR_MODULUS`**-1). If **`CAESAR_MODULUS`** is equal to 0, the result is undefined.

....

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