NAME

caesar_edge – the ‘‘edge’’ library of OPEN/CAESAR

PURPOSE

The ‘‘edge’’ library provides primitives for computing the edges going out of a given state.

Although the *OPEN/CAESAR* graph module already provides an iterator macro for this purpose (the **CAESAR_ITERATE_STATE()** function), higher-level primitives may be useful and easier to use. This is especially the case when a depth-first traversal of the state graph is necessary (e.g., on-the-fly verification, interactive simulation, ...).

From our experience, the **CAESAR_ITERATE_STATE()** function is often used as follows: for a given state S_1, one wants to compute all outgoing edges of the form ‘‘(S_1, L, S_2)’’; the labels L and states S_2 are stored in a data structure (usually a linked list).

USAGE

The ‘‘edge’’ library consists of:

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

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

DESCRIPTION

An ‘‘edge’’ is basically a tuple with 5 fields:

- (1) a field containing a ‘‘previous’’ state;
- (2) a field containing a label;
- (3) a field containing a ‘‘next’’ state;
- (4) a ‘‘mark’’ field, that is a byte string whose size and contents are freely determined by the user. It can be used to mark states while depth-first exploring the state graph. The size of the mark field is the same for all edges; it must be greater or equal than zero. Pointers to mark fields will be considered as values of type **CAESAR_TYPE_POINTER**; ‘‘mark’’ fields are always aligned on appropriate boundaries so that the user can put any information in these fields without alignment problem;
- (5) a pointer to a ‘‘successor’’ edge, which is used to build linked lists of edges.

Fields (1), (2), (3), and (4) are optional, depending on the initialization parameters (see function **CAESAR_INIT_EDGE()** below).

Edges are represented as byte strings of fixed size (see function **CAESAR_SIZE_EDGE()** below) with definite alignment constraints (see function **CAESAR_ALIGNMENT_EDGE()** below). All edges have the same size.

FEATURES

.....

CAESAR_TYPE_EDGE

```
typedef CAESAR_TYPE_ABSTRACT (...) CAESAR_TYPE_EDGE;
```

This type denotes a pointer to the concrete representation of an edge. The edge representation is supposed

to be ‘‘opaque’’.

.....

CAESAR_INIT_EDGE

```
void CAESAR_INIT_EDGE (CAESAR_PREVIOUS_STATE, CAESAR_LABEL, CAESAR_NEXT_STATE,
                       CAESAR_SIZE_MARK, CAESAR_ALIGNMENT_MARK)
CAESAR_TYPE_BOOLEAN CAESAR_PREVIOUS_STATE;
CAESAR_TYPE_BOOLEAN CAESAR_LABEL;
CAESAR_TYPE_BOOLEAN CAESAR_NEXT_STATE;
CAESAR_TYPE_NATURAL CAESAR_SIZE_MARK;
CAESAR_TYPE_NATURAL CAESAR_ALIGNMENT_MARK;
{ ... }
```

This initialization procedure must be called before using any other primitive of the ‘‘edge’’ library. It should be called only once.

Each edge will contain a ‘‘previous’’ state iff **CAESAR_PREVIOUS_STATE** is equal to true.

Each edge will contain a label iff **CAESAR_LABEL** is equal to true.

Each edge will contain a ‘‘next’’ state iff **CAESAR_NEXT_STATE** is equal to true.

Each edge will contain a mark field iff **CAESAR_SIZE_MARK** is different from zero. If so, the value of **CAESAR_SIZE_MARK** determines the (constant) size (in bytes) of the mark field, and the value of **CAESAR_ALIGNMENT_MARK** determines the alignment factor (in bytes) of the mark field. The alignment factor must be a power of two. Any mark field will be aligned on a boundary that is an even multiple of the alignment factor. **CAESAR_ALIGNMENT_MARK** is equal to zero iff **CAESAR_SIZE_MARK** is equal to zero; otherwise, the effect of **CAESAR_INIT_EDGE()** is undefined.

If **CAESAR_PREVIOUS_STATE**, **CAESAR_LABEL**, and **CAESAR_NEXT_STATE** are equal to false, and if **CAESAR_SIZE_MARK** is equal to zero, the effect of **CAESAR_INIT_EDGE()** is undefined.

.....

CAESAR_SIZE_EDGE

```
CAESAR_TYPE_NATURAL CAESAR_SIZE_EDGE ()
{ ... }
```

This function returns the edge size (in bytes).

.....

CAESAR_ALIGNMENT_EDGE

```
CAESAR_TYPE_NATURAL CAESAR_ALIGNMENT_EDGE ()
   { ... }
```

This function returns the alignment factor (in bytes) for edges. The alignment factor is always a power of two, usually 1, 2, 4, or 8. Any byte string representing a edge must be aligned on a boundary that is an even multiple of the alignment factor.

.....

CAESAR_PREVIOUS_STATE_EDGE

```
CAESAR_TYPE_STATE CAESAR_PREVIOUS_STATE_EDGE (CAESAR_E)
CAESAR_TYPE_EDGE CAESAR_E;
{ ... }
```

This function returns a pointer to the ''previous'' state field of the edge pointed to by **CAESAR_E**. If there is no such field (due to the initialization parameters supplied to **CAESAR_INIT_EDGE()**) the result is undefined.

.....

CAESAR_LABEL_EDGE

```
CAESAR_TYPE_LABEL CAESAR_LABEL_EDGE (CAESAR_E)
CAESAR_TYPE_EDGE CAESAR_E;
{ ... }
```

This function returns a pointer to the label field of the edge pointed to by **CAESAR_E**. If there is no such field (due to the initialization parameters supplied to **CAESAR_INIT_EDGE()**) the result is undefined.

.....

CAESAR_NEXT_STATE_EDGE

```
CAESAR_TYPE_STATE CAESAR_NEXT_STATE_EDGE (CAESAR_E)
CAESAR_TYPE_EDGE CAESAR_E;
{ ... }
```

This function returns a pointer to the ''next'' state field of the edge pointed to by **CAESAR_E**. If there is no such field (due to the initialization parameters supplied to **CAESAR_INIT_EDGE()**) the result is undefined.

.....

CAESAR_MARK_EDGE

```
CAESAR_TYPE_POINTER CAESAR_MARK_EDGE (CAESAR_E)
CAESAR_TYPE_EDGE CAESAR_E;
{ ... }
```

This function returns a pointer to the mark field of the edge pointed to by **CAESAR_E**. If there is no such field (due to the initialization parameters supplied to **CAESAR_INIT_EDGE()**) the result is undefined.

.....

CAESAR_SUCCESSOR_EDGE

```
CAESAR_TYPE_EDGE *CAESAR_SUCCESSOR_EDGE (CAESAR_E)
CAESAR_TYPE_EDGE CAESAR_E;
{ ... }
```

This function returns a pointer to the ''successor'' edge (pointer) field of the edge pointed to by **CAESAR_E**.

.....

CAESAR_CREATE_EDGE

```
void CAESAR_CREATE_EDGE (CAESAR_E)
   CAESAR_TYPE_EDGE *CAESAR_E;
   { ... }
```

This procedure allocates a byte string of length **CAESAR_SIZE_EDGE()** using **CAESAR_CREATE()** and assigns its address to ***CAESAR_E**. If the allocation fails, the **NULL** value is assigned to ***CAESAR_E**.

When the allocation succeeds, the mark field (if any) of **CAESAR_E** is initialized to a bit string of 0's and the ''successor'' edge field is initialized to the **NULL** pointer. The state field and the label field are left undefined.

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

```
CAESAR_CREATE_EDGE (&CAESAR_E);
```

However, it is not necessary to use **CAESAR_CREATE_EDGE()** to perform the allocation. Instead, users can allocate edges into their own data structures (tables, lists, ...)

.....

CAESAR_DELETE_EDGE

```
void CAESAR_DELETE_EDGE (CAESAR_E)
   CAESAR_TYPE_EDGE *CAESAR_E;
   { ... }
```

This procedure frees the byte string of length **CAESAR_SIZE_EDGE()** pointed to by ***CAESAR_E** using **CAESAR_DELETE()**. Afterwards, the **NULL** value is assigned to ***CAESAR_E**.

.....

CAESAR_COPY_EDGE

```
void CAESAR_COPY_EDGE (CAESAR_E1, CAESAR_E2)
   CAESAR_TYPE_EDGE CAESAR_E1;
   CAESAR_TYPE_EDGE CAESAR_E2;
   { ... }
```

This procedure copies the edge pointed to by **CAESAR_E2** onto the edge pointed to by **CAESAR_E1**.

.....

CAESAR_FORMAT_EDGE

```
CAESAR_TYPE_FORMAT CAESAR_FORMAT_EDGE (CAESAR_FORMAT)
   CAESAR_TYPE_FORMAT CAESAR_FORMAT;
   { ... }
```

This function allows to control the format under which edges are printed by procedures **CAESAR_PRINT_EDGE()** and **CAESAR_PRINT_EDGE_LIST()** (see below). Currently, the following formats are available:

- With format 0, the edge is printed as a portion of text. This is mainly intended for debugging purpose.
- (no other format available yet)

By default, the current edge format is initialized to 0.

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

When called with another value of **CAESAR_FORMAT**, this function does not modify the current edge format 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 edge format. If **CAESAR_FORMAT** is equal to the constant **CAESAR_MAXIMAL_FORMAT**, the result is the maximal format value (i.e., 0). In all other cases, the effect of this function is undefined.

.....

CAESAR_MAX_FORMAT_EDGE

```
CAESAR_TYPE_FORMAT CAESAR_MAX_FORMAT_EDGE ()
   { ... }
```

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_EDGE()** instead, called with argument **CAESAR_MAXIMAL_FORMAT**.

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

.....

CAESAR_PRINT_EDGE

```
void CAESAR_PRINT_EDGE (CAESAR_FILE, CAESAR_E)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_EDGE CAESAR_E;
   { ... }
```

This procedure prints to file **CAESAR_FILE** information about the contents of the edge pointed to by **CAESAR_E**. The nature of the information is determined by the current edge format (see procedure **CAESAR_FORMAT_EDGE()** above).

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

.....

CAESAR_CREATE_EDGE_LIST

```
void CAESAR_CREATE_EDGE_LIST (CAESAR_S1, CAESAR_E1_En, CAESAR_ORDER)
CAESAR_TYPE_STATE CAESAR_S1;
CAESAR_TYPE_EDGE *CAESAR_E1_En;
CAESAR_TYPE_NATURAL CAESAR_ORDER;
```

This procedure computes all couples (**CAESAR_L**, **CAESAR_S2**) such that ‘‘(**CAESAR_S1**, **CAESAR_L**, **CAESAR_S2**)’’ is an edge of the labelled transition system (this is done using the **CAESAR_ITERATE_STATE()** procedure of the graph module).

This procedure also builds a linked list whose items are values of type **CAESAR_TYPE_EDGE**, linked together using the ‘‘successor’’ edge field. The ‘‘successor’’ edge field of the last item is set to **NULL**. The list can be empty if **CAESAR_S1** is a sink state. The address of the first item of the list (or **NULL** if the list is empty) is assigned to ∗**CAESAR_E1_En**. Obviously, the previous value of ∗**CAESAR_E1_En** is lost.

The fields of each item are assigned as follows:

- the ''previous'' state field (if any) will contain the value of **CAESAR_S1**.
- the label field (if any) will contain various values for **CAESAR_L**.
- the ''next'' state field (if any) of each item will contain values for **CAESAR_S2**.
- the mark field (if any) is initialized to a bit string of 0's. The value of the formal parameter **CAESAR_ORDER** determines the order of the items of the linked list. Several cases are currently implemented:
- if **CAESAR_ORDER** is equal to 0, the list order is undefined.
- if **CAESAR_ORDER** is equal to 1, the edge list is sorted in the same order as transitions are enumerated by the **CAESAR_ITERATE_STATE()** procedure.
- if **CAESAR_ORDER** is equal to 2, the edge list is sorted in the reverse order of the order in which transitions are enumerated by the **CAESAR_ITERATE_STATE()** procedure.
- if **CAESAR_ORDER** is equal to 3 or 5, the list is sorted in such a way that the character string values returned by **CAESAR_STRING_LABEL()** are increasing, according to the lexicographical order used in the function **strcmp(3)**.
- if **CAESAR_ORDER** is equal to 4 or 6, the list is sorted in such a way that the character string values returned by **CAESAR_STRING_LABEL()** are decreasing, according to the lexicographical order used in the function **strcmp(3)**.

Additionally, this procedure sets two global variables **caesar_creation** and **caesar_truncation** of type **CAESAR_TYPE_NATURAL**. After any call to **CAESAR_CREATE_EDGE_LIST()**, these variables can be inspected using the two functions **CAESAR_CREATION_EDGE_LIST()** and **CAESAR_TRUNCATION_EDGE_LIST()** defined below. The values of these variables are set as follows:

- if the computation normally succeeds, then **caesar_creation** is set to the number of items in the linked list and **caesar_truncation** is set to zero.
- if allocation fails when building the list (due to a lack of memory), a truncated list is built (the ''successor'' edge field of the last item is still set to **NULL**). Then **caesar_creation** is set to the number of items in the truncated list and **caesar_truncation** is set to the number of items that have not been inserted in the list (this number is greater than zero).

.....

CAESAR_MAX_ORDER_EDGE_LIST

```
CAESAR_TYPE_NATURAL CAESAR_MAX_ORDER_EDGE_LIST ()
   { ... }
```

This function returns the highest order available for edge list creation, i.e., the highest acceptable value for the parameter **CAESAR_ORDER** of function **CAESAR_CREATE_EDGE_LIST()**.

.....

CAESAR_DELETE_EDGE_LIST

```
void CAESAR_DELETE_EDGE_LIST (CAESAR_E1_En)
CAESAR_TYPE_EDGE *CAESAR_E1_En;
{ ... }
```

This procedure frees each item of the linked list pointed to by ***CAESAR_E1_En**. Afterwards, the **NULL** value is assigned to ***CAESAR_E1_En**.

.....

CAESAR_COPY_EDGE_LIST

```
void CAESAR_COPY_EDGE_LIST (CAESAR_E1_Em, CAESAR_E1_En)
CAESAR_TYPE_EDGE *CAESAR_E1_Em;
CAESAR_TYPE_EDGE CAESAR_E1_En;
{ ... }
```

This procedure builds a duplicate list, which is a copy of the linked list pointed to by **CAESAR_E1_En**. A pointer to this duplicate list (or NULL if the list is empty) is assigned to ***CAESAR_E1_Em**. For each item of the linked list pointed to by **CAESAR_E1_En**, a duplicated item is allocated. Said differently, both lists do not have shared items in common.

Additionally, this procedure sets two global variables **caesar_creation** and **caesar_truncation** of type **CAESAR_TYPE_NATURAL**. After any call to **CAESAR_COPY_EDGE_LIST()**, these variables can be inspected using the two functions **CAESAR_CREATION_EDGE_LIST()** and **CAESAR_TRUNCATION_EDGE_LIST()** defined below. The values of these variables are set as in the **CAESAR_CREATE_EDGE_LIST()** function.

The previous value of ***CAESAR_E1_Em** is lost: if it points to a non-empty list, this list should be freed using **CAESAR_DELETE_EDGE_LIST()** before **CAESAR_COPY_EDGE_LIST()** is called.

.....

CAESAR_CREATION_EDGE_LIST

```
CAESAR_TYPE_NATURAL CAESAR_CREATION_EDGE_LIST ()
   { ... }
```

This function returns the value of the global variable **caesar_creation** computed during the last call to **CAESAR_CREATE_EDGE_LIST()** or **CAESAR_COPY_EDGE_LIST()**. This variable can only be accessed using this function.

.....

CAESAR_TRUNCATION_EDGE_LIST

```
CAESAR_TYPE_NATURAL CAESAR_TRUNCATION_EDGE_LIST ()
   { ... }
```

This function returns the value of the global variable **caesar_truncation** computed during the last call to **CAESAR_CREATE_EDGE_LIST()** or **CAESAR_COPY_EDGE_LIST**. This variable can only be accessed using this function.

.....

CAESAR_FORMAT_EDGE_LIST

```
CAESAR_TYPE_FORMAT CAESAR_FORMAT_EDGE_LIST (CAESAR_FORMAT)
   CAESAR_TYPE_FORMAT CAESAR_FORMAT;
   { ... }
```

This function allows to control the format under which edge lists are printed by the procedure **CAESAR_PRINT_EDGE_LIST()** (see below). Currently, the following formats are available:

- With format 0, the edge list is printed as a portion of text. This is mainly intended for debugging

purpose.

- (no other format available yet)

A call to this function sets the current edge list format to **CAESAR_FORMAT**.

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

When called with another value of **CAESAR_FORMAT**, this function does not modify the current edge list format 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 edge list format. If **CAESAR_FORMAT** is equal to the constant **CAESAR_MAXIMAL_FORMAT**, the result is the maximal format value (i.e., 0). In all other cases, the effect of this function is undefined.

.....

CAESAR_MAX_FORMAT_EDGE_LIST

```
CAESAR_TYPE_FORMAT CAESAR_MAX_FORMAT_EDGE_LIST ()
   { ... }
```

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_EDGE_LIST()** instead, called with argument **CAESAR_MAXIMAL_FORMAT**.

This function returns the maximal format value available for printing edge lists.

.....

CAESAR_PRINT_EDGE_LIST

```
void CAESAR_PRINT_EDGE_LIST (CAESAR_FILE, CAESAR_E1_En)
   CAESAR_TYPE_FILE CAESAR_FILE;
   CAESAR_TYPE_EDGE CAESAR_E1_En;
   { ... }
```

This procedure prints to file **CAESAR_FILE** information about the contents of the linked list of edges pointed to by **CAESAR_E1_En**. The nature of the information is determined by the current edge format and the current edge list format (see procedures **CAESAR_FORMAT_EDGE()** and **CAESAR_FORMAT_EDGE_LIST()** above).

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

.....

CAESAR_ITERATE_PLNM_EDGE_LIST

```
#define CAESAR_ITERATE_PLNM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,\
                    CAESAR_S1,CAESAR_L,CAESAR_S2,CAESAR_M) ...
```

with parameters typed as follows:

```
CAESAR_TYPE_EDGE CAESAR_E1_En;
CAESAR_TYPE_EDGE CAESAR_E;
CAESAR_TYPE_STATE CAESAR_S1;
CAESAR_TYPE_LABEL CAESAR_L;
```

```
CAESAR_TYPE_STATE CAESAR_S2;
CAESAR_TYPE_POINTER CAESAR_M;
```

This macro-definition is an iterator which can be used in the same way as a **while (...)** or **for (...; ...; ...)** instruction. It is therefore possible to write an instruction such as:

```
CAESAR_ITERATE_PLNM_EDGE_LIST(caesar_e1_en,caesar_e,
            caesar_s1,caesar_l,caesar_s2,caesar_m) {
...
body of the loop, containing occurrences of variables
caesar_e, caesar_s1, caesar_l, caesar_s2, and caesar_m
...
}
```

CAESAR_E1_En is an expression (r-value) containing a pointer to the first item of a linked list of edges.

CAESAR_E is a variable (l-value) which will be used as the induction variable in the body of the loop. At the n-th iteration step, it points to the n-th item of the linked list.

CAESAR_S1 is a variable (l-value) which can also be used as an induction variable. At the n-th iteration step, it points to the ''previous'' state field of the n-th item of the linked list. If this field does not exist, the result is undefined. At each step, one has:

CAESAR_S1 == **CAESAR_PREVIOUS_STATE_EDGE (CAESAR_E)**

CAESAR_L is a variable (l-value) which can also be used as an induction variable. At the n-th iteration step, it points to the label field of the n-th item of the linked list. If this field does not exist, the result is undefined. At each step, one has:

CAESAR_L == **CAESAR_LABEL_EDGE (CAESAR_E)**

CAESAR_S2 is a variable (l-value) which can also be used as an induction variable. At the n-th iteration step, it points to the ''next'' state field of the n-th item of the linked list. If this field does not exist, the result is undefined. At each step, one has:

CAESAR_S2 == **CAESAR_NEXT_STATE_EDGE (CAESAR_E)**

CAESAR_M is a variable (l-value) which can also be used as an induction variable. At the n-th iteration step, it points to the mark field of the n-th item of the linked list. If this field does not exist, the result is undefined. At each step, one has:

CAESAR_M == **CAESAR_MARK_EDGE (CAESAR_E)**

The body of the loop can be any statement of the C language. In particular, it may contain **break** and **continue** statements with their usual semantics.

This is the most general iterator on linked lists of edges. There are also 15 other iterators derived from the general one. These iterators are simpler than the general one, since they deal with the cases where one or several of the following parameters:

CAESAR_S1, **CAESAR_L**, **CAESAR_S2**, **CAESAR_M**

are omitted. These operators are used according to the needs (for example, the four aforementioned parameters can be omitted if one only wants to compute the length of an edge list) and also depending on the initialization values given to **CAESAR_INIT_EDGE()** (since one or several fields may not actually exist).

The 15 derived iterators are listed below.

```
#define CAESAR_ITERATE_EDGE_LIST(CAESAR_E1_En,CAESAR_E) ...

#define CAESAR_ITERATE_P_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1) ...

#define CAESAR_ITERATE_L_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_L) ...

#define CAESAR_ITERATE_N_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S2) ...

#define CAESAR_ITERATE_M_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_M) ...

#define CAESAR_ITERATE_PL_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_L) ...

#define CAESAR_ITERATE_PN_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_S2) ...

#define CAESAR_ITERATE_PM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_M) ...

#define CAESAR_ITERATE_LN_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_L,CAESAR_S2) ...

#define CAESAR_ITERATE_LM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_L,CAESAR_M) ...

#define CAESAR_ITERATE_NM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S2,CAESAR_M) ...

#define CAESAR_ITERATE_PLN_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_L,CAESAR

#define CAESAR_ITERATE_PLM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_L,CAESAR

#define CAESAR_ITERATE_PNM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_S1,CAESAR_S2,CAESA

#define CAESAR_ITERATE_LNM_EDGE_LIST(CAESAR_E1_En,CAESAR_E,CAESAR_L,CAESAR_S2,CAESAR
```

.....

CAESAR_LENGTH_EDGE_LIST

```
CAESAR_TYPE_NATURAL CAESAR_LENGTH_EDGE_LIST (CAESAR_E1_En)
   CAESAR_TYPE_EDGE CAESAR_E1_En;
   { ... }
```

This function returns the number of items in the linked list pointed to by **CAESAR_E1_En**.

.....

CAESAR_ITEM_EDGE_LIST

```
CAESAR_TYPE_EDGE CAESAR_ITEM_EDGE_LIST (CAESAR_E1_En, CAESAR_N)
```

```
CAESAR_TYPE_EDGE CAESAR_E1_En;
CAESAR_TYPE_NATURAL CAESAR_N;
{ ... }
```

This function returns the **CAESAR_N**-th item in the linked list pointed to by **CAESAR_E1_En** (the first item is numbered 1). If **CAESAR_N** is equal to 0, or is greater than the actual length of the linked list, the result is undefined.

.....

CAESAR_REVERSE_EDGE_LIST

```
void CAESAR_REVERSE_EDGE_LIST (CAESAR_E1_En)
CAESAR_TYPE_EDGE *CAESAR_E1_En;
{ ... }
```

This procedure reverses the linked list of edges pointed to by ***CAESAR_E1_En**.

.....

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