NAME

bcg_read – a simple interface to read a BCG graph

DESCRIPTION

This interface reads a BCG graph from an application program written in C or C++. To keep things simple, this interface views states as unsigned integer numbers and labels as character strings. Note: this subset of BCG is equivalent to the **.aut** format described in the **aldebaran**(LOCAL) manual page, although it is much more compact.

USAGE

The application program should start with the following directive:

```
#include "bcg_user.h"
```

Then the BCG library should be initialized by invoking the following function:

```
BCG_INIT ();
```

Not invoking this function might cause a run-time error, e.g., a segmentation fault. Invoking **BCG_INIT()** more than once is harmless, although not recommended.

DATA TYPES

In order to read a BCG graph, declare a data structure having the type **BCG_TYPE_OBJECT_TRANSITION** (this type is provided by the "bcg_user.h" file), which contains information related to the graph. Assuming that this data structure will be named *bcg_graph*, this declaration will have the following form:

```
BCG_TYPE_OBJECT_TRANSITION bcg_graph;
```

Note: this interface is reentrant, in the sense that several BCG graphs can be read concurrently, provided that each BCG graph is associated with a distinct data structure of type **BCG_TYPE_OBJECT_TRANSITION**.

The functions of this interface also use the followings types, whose definitions are provided by the "bcg_user.h" file:

BCG_TYPE_BOOLEAN

BCG_TYPE_NATURAL

BCG_TYPE_C_STRING

BCG_TYPE_FILE_NAME

BCG_TYPE_STATE_NUMBER

BCG_TYPE_LABEL_NUMBER

BCG_TYPE_EDGE_NUMBER

FEATURES

....

BCG_OT_READ_BCG_BEGIN

```
BCG_OT_READ_BCG_BEGIN (filename, bcg_graph, access_mode)

 BCG_TYPE_FILE_NAME          filename;
 BCG_TYPE_OBJECT_TRANSITION *bcg_graph;
 BCG_TYPE_NATURAL            access_mode;
 { ... }
```

This function opens a BCG file.

filename
: is a character string containing the path name of the BCG file to be read. It should contain the "**.bcg**" suffix (if the "**.bcg**" suffix is missing, it will be added automatically);

bcg_graph
: is a pointer to the data structure which will become associated with the BCG graph and into which information about the BCG graph will be stored;

access_mode
: is an integer indicating which kind of primitives are expected for exploring the transition relation of the BCG graph.

There are several possible values for *access_mode*:

0: should be used when it is only necessary to enumerate the edges in the BCG graph.

1: should be used when one has to enumerate the successors of a given state.

2: should be used when one has to enumerate the predecessors of a given state.

3: should be used when one has to enumerate both the successors and predecessors of a given state.

4: is similar to value 3, but uses a different data structure.

By default, if *filename* cannot be opened for reading, **BCG_OT_READ_BCG_BEGIN()** will emit an error message and exit the program. However, if the following function call:

```
BCG_OT_READ_BCG_SURVIVE (BCG_TRUE);
```

has occured before the call to **BCG_OT_READ_BCG_BEGIN()**, then **BCG_OT_READ_BCG_BEGIN()** will neither emit an error message nor exit the program, but return normally with its *bcg_graph* parameter set to **NULL** if and only if *filename* cannot be opened.

Below, we assume that the **bcg_graph* data structure has been properly assigned after a successful invocation of the **BCG_OT_READ_BCG_BEGIN()** function.

....

BCG_OT_READ_BCG_SURVIVE

```
BCG_OT_READ_BCG_SURVIVE (mode)

  BCG_TYPE_BOOLEAN mode;
  { ... }
```

This function controls how the **BCG_OT_READ_BCG_BEGIN()** function defined above will behave if the BCG file cannot be opened for writing:

- If *mode* equals **BCG_FALSE**, then **BCG_OT_READ_BCG_BEGIN()** will emit an error message and exit the program. This is the default behaviour.
- If *mode* equals **BCG_TRUE**, then **BCG_OT_READ_BCG_BEGIN()** will neither emit an error message nor exit the program, but return a boolean result. The default behaviour can be restored by calling:

```
BCG_OT_READ_BCG_SURVIVE (BCG_FALSE);
```

....

BCG_OT_INITIAL_STATE

```
BCG_TYPE_STATE_NUMBER BCG_OT_INITIAL_STATE (bcg_graph)

  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  { ... }
```

This function returns the number of the initial state of the BCG graph.

....

BCG_OT_NB_STATES

```
BCG_STATE_NUMBER BCG_OT_NB_STATES (bcg_graph)

 BCG_TYPE_OBJECT_TRANSITION bcg_graph;
 { ... }
```

This function returns the number of states of the BCG graph. States are given unique numbers in the range `0` to `BCG_OT_NB_STATES (`*bcg_graph*`) - 1`.

....

BCG_OT_NB_EDGES

```
BCG_TYPE_EDGE_NUMBER BCG_OT_NB_EDGES (bcg_graph)

 BCG_TYPE_OBJECT_TRANSITION bcg_graph;
 { ... }
```

This function returns the number of edges (i.e., transitions) of the BCG graph. Edges are given unique numbers in the range `0` to `BCG_OT_NB_EDGES (`*bcg_graph*`) - 1`.

....

BCG_OT_NB_LABELS

```
BCG_TYPE_LABEL_NUMBER BCG_OT_NB_LABELS (bcg_graph)

 BCG_TYPE_OBJECT_TRANSITION bcg_graph;
 { ... }
```

This function returns the number of labels of the BCG graph. Labels are given unique numbers in the range `0` to `BCG_OT_NB_LABELS (`*bcg_graph*`) - 1`.

....

BCG_OT_LABEL_STRING

```
BCG_TYPE_C_STRING BCG_OT_LABEL_STRING (bcg_graph,
                   bcg_label_number)

 BCG_TYPE_OBJECT_TRANSITION bcg_graph;
 BCG_TYPE_LABEL_NUMBER      bcg_label_number;
 { ... }
```

This function returns a character string denoting the label whose index is *bcg_label_number*. It assumes *bcg_label_number* belongs to the range `0` to `BCG_OT_NB_LABELS (`*bcg_graph*`) - 1`.

....

BCG_OT_LABEL_VISIBLE

```
BCG_TYPE_BOOLEAN BCG_OT_LABEL_VISIBLE (bcg_graph,
                   bcg_label_number)
```

```
  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  BCG_TYPE_LABEL_NUMBER      bcg_label_number;
  { ... }
```

This function returns a boolean that is equal to **BCG_FALSE** if the label whose index is *bcg_label_number* is not visible, or **BCG_TRUE** if this label is visible. It assumes *bcg_label_number* belongs to the range **0** to **BCG_OT_NB_LABELS (***bcg_graph***) - 1**.

....

BCG_OT_LABEL_GATE

```
 BCG_TYPE_C_STRING BCG_OT_LABEL_GATE (bcg_graph,
                   bcg_label_number)

  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  BCG_TYPE_LABEL_NUMBER      bcg_label_number;
  { ... }
```

This function returns a character string that is equal to **"i"** if the label whose index is *bcg_label_number* is not visible, or to the gate (i.e., first element) of this label if this label is visible. It assumes *bcg_label_number* belongs to the range **0** to **BCG_OT_NB_LABELS (***bcg_graph***) - 1**.

....

BCG_OT_LABEL_HIDDEN_GATE

```
 BCG_TYPE_C_STRING BCG_OT_LABEL_HIDDEN_GATE (bcg_graph,
                   bcg_label_number)

  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  BCG_TYPE_LABEL_NUMBER      bcg_label_number;
  { ... }
```

This function returns a character string that is equal to **""** if the label whose index is *bcg_label_number* is visible, or to the gate (i.e., first element) of this label (if any) before it was hidden. It assumes *bcg_label_number* belongs to the range **0** to **BCG_OT_NB_LABELS (***bcg_graph***) - 1**.

....

BCG_OT_LABEL_CARDINAL

```
 BCG_TYPE_NATURAL BCG_OT_LABEL_CARDINAL (bcg_graph,
                  bcg_label_number)

  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  BCG_TYPE_LABEL_NUMBER      bcg_label_number;
  { ... }
```

This function returns a natural number that is equal to the number of elements in the label whose index is *bcg_label_number* (this number is always equal to 1 if the label is not visible). It assumes *bcg_label_number* belongs to the range **0** to **BCG_OT_NB_LABELS (***bcg_graph***) - 1**.

....

BCG_OT_ITERATE_PLN

```
BCG_TYPE_OBJECT_TRANSITION bcg_graph;
BCG_TYPE_STATE_NUMBER      bcg_state_1;
BCG_TYPE_LABEL_NUMBER      bcg_label_number;
BCG_TYPE_STATE_NUMBER      bcg_state_2;

BCG_OT_ITERATE_PLN (bcg_graph, bcg_state_1,
            bcg_label_number, bcg_state_2)
{
/* loop body */
} BCG_OT_END_ITERATE;
```

This iterator enumerates all the edges in the BCG graph associated to *bcg_graph*.

bcg_state_1 is the number of the origin state of the edge, *bcg_state_2* is the number of the destination state of the edge, and *bcg_label_number* is the number of the label of the edge. These three variables are assigned by the iterator. They are visible in the loop body, where they can be read but not assigned.

For each edge *(bcg_state_1, bcg_label_number, bcg_state_2)*, the loop body is executed. The loop body can be any statement block of the C language. In particular, it may contain "**break**" and "**continue**" statements with their usual semantics. The order in which the edges are enumerated by this iterator is left unspecified.

This iterator is available whatever value was given to *access_mode* when the BCG file associated to *bcg_graph* was opened using **BCG_OT_READ_BCG_BEGIN()**.

There is also a whole family of iterators that allow the enumeration of all the edges for which *bcg_state_1* and/or *bcg_label_number* and/or *bcg_state_2* are given fixed values. The availability of these iterators depends upon the value given to *access_mode* when the BCG file associated to *bcg_graph* was opened. We describe below a representative iterator (named **BCG_OT_ITERATE_P_LN**) in this family.

....

BCG_OT_ITERATE_P_LN

```
BCG_TYPE_OBJECT_TRANSITION bcg_graph;
BCG_TYPE_STATE_NUMBER      bcg_state_1;
BCG_TYPE_LABEL_NUMBER      bcg_label_number;
BCG_TYPE_STATE_NUMBER      bcg_state_2;

BCG_OT_ITERATE_P_LN (bcg_graph, bcg_state_1,
            bcg_label_number, bcg_state_2)
{
/* loop body */
} BCG_OT_END_ITERATE;
```

This iterator enumerates all the edges in the BCG graph with origin state equal to *bcg_state_1*. *bcg_state_1* is a variable or an expression whose value is evaluated when the iterator is started. *bcg_state_1* is not assigned by the iterator.

bcg_state_2 is the number of the destination state of the edge, and *bcg_label_number* is the number of the label of the edge. These two variables are assigned by the iterator. They are visible in the loop body, where they can be read but not assigned.

For each edge *(bcg_state_1, bcg_label_number, bcg_state_2)*, the loop body is executed. The loop body can be any statement block of the C language. In particular, it may contain "**break**" and "**continue**" statements with their usual semantics. The order in which the edges going out of a given state *bcg_state_1* are enumerated by this iterator is left unspecified.

This iterator is available if *access_mode* was given the value 1, 3, or 4 when the BCG file associated to *bcg_graph* was opened using **BCG_OT_READ_BCG_BEGIN()**.

There are several others iterators available. They are defined in file "**$CADP/incl/bcg_iterator.h**".

....

BCG_OT_READ_BCG_END

```
BCG_TYPE_BOOLEAN BCG_OT_READ_BCG_END (bcg_graph);
BCG_TYPE_OBJECT_TRANSITION *bcg_graph;
 { ... }
```

This function closes a BCG graph and erases the contents of the **bcg_graph* data structure.

....

EXAMPLE

```
#include "bcg_user.h"

/* The following function prints information about a BCG graph */

void bcg_print_info (bcg_graph)
BCG_TYPE_OBJECT_TRANSITION bcg_graph;
{
  printf ("initial state = %lu\n", BCG_OT_INITIAL_STATE (bcg_graph));
  printf ("nb states = %lu\n", BCG_OT_NB_STATES (bcg_graph));
  printf ("nb edges = %lu\n", BCG_OT_NB_EDGES (bcg_graph));
  printf ("nb labels = %u\n", BCG_OT_NB_LABELS (bcg_graph));
}
```

```
/* The following function displays an edge */

void bcg_print_edge (bcg_graph, bcg_state_1, bcg_label_number, bcg_state_2)
BCG_TYPE_OBJECT_TRANSITION bcg_graph;
BCG_TYPE_STATE_NUMBER bcg_state_1;
BCG_TYPE_LABEL_NUMBER bcg_label_number;
BCG_TYPE_STATE_NUMBER bcg_state_2;
{
  BCG_TYPE_C_STRING bcg_label_string;
  BCG_TYPE_BOOLEAN bcg_visible;
  BCG_TYPE_NATURAL bcg_cardinal;
  BCG_TYPE_C_STRING bcg_gate;

  bcg_label_string = BCG_OT_LABEL_STRING (bcg_graph, bcg_label_number);
  bcg_visible = BCG_OT_LABEL_VISIBLE (bcg_graph, bcg_label_number);
  bcg_cardinal = BCG_OT_LABEL_CARDINAL (bcg_graph, bcg_label_number);

  printf ("transition from state %lu to state %lu\n",
        bcg_state_1, bcg_state_2);
  printf ("label unique number = %u\n", bcg_label_number);
  printf ("label string = %s\n", bcg_label_string);
  printf ("label cardinal = %u\n", bcg_cardinal);
  if (bcg_visible) {
    bcg_gate = BCG_OT_LABEL_GATE (bcg_graph, bcg_label_number);
    printf ("visible label (gate = %s)\n", bcg_gate);
  } else {
    bcg_gate = BCG_OT_LABEL_HIDDEN_GATE (bcg_graph, bcg_label_number);
    printf ("hidden label (hidden gate = %s)\n", bcg_gate);
  }
}

int main ()
{
  BCG_TYPE_OBJECT_TRANSITION bcg_graph;
  BCG_TYPE_STATE_NUMBER bcg_s1;
  BCG_TYPE_LABEL_NUMBER bcg_label_number;
  BCG_TYPE_STATE_NUMBER bcg_s2;
  BCG_TYPE_STATE_NUMBER bcg_nb_states;

  BCG_INIT ();

  /* The following fragment of code reads and prints all the edges
    of a BCG graph, in an undefined order */

  BCG_OT_READ_BCG_BEGIN ("test.bcg", &bcg_graph, 0);
  bcg_print_info (bcg_graph);
  BCG_OT_ITERATE_PLN (bcg_graph, bcg_s1, bcg_label_number, bcg_s2) {
    bcg_print_edge (bcg_graph, bcg_s1, bcg_label_number, bcg_s2);
  } BCG_OT_END_ITERATE;
  BCG_OT_READ_BCG_END (&bcg_graph);
  /* The following fragment of code reads and prints all the edges
     of a BCG graph, sorted by origin states in increasing order */

  BCG_OT_READ_BCG_BEGIN ("test.bcg", &bcg_graph, 1);
  bcg_print_info (bcg_graph);
  bcg_nb_states = BCG_OT_NB_STATES (bcg_graph);
  for (bcg_s1 = 0; bcg_s1 < bcg_nb_states; bcg_s1++) {
     printf ("successors of state %lu:\n", bcg_s1);
     BCG_OT_ITERATE_P_LN (bcg_graph, bcg_s1, bcg_label_number, bcg_s2) {
       bcg_print_edge (bcg_graph, bcg_s1, bcg_label_number, bcg_s2);
     } BCG_OT_END_ITERATE;
  }
  BCG_OT_READ_BCG_END (&bcg_graph);
  exit (0);
}
```

COMPILING AND LINK EDITING

To compile the application tool, the following options must be passed to the C or C++ compiler:

-I$CADP/incl -L$CADP/bin.‘$CADP/com/arch‘ -lBCG_IO -lBCG -lm

as in, e.g.,

```
$CADP/src/com/cadp_cc tool.c -o tool -I$CADP/incl \
-L$CADP/bin.‘$CADP/com/arch‘ -lBCG_IO -lBCG -lm
```

EXIT STATUS

Application tools share common conventions with respect to diagnostics. Exit status is 0 if everything is alright, 1 otherwise.

AUTHORS

Hubert Garavel (definition of the BCG format) and Renaud Ruffiot (implementation of the BCG environment).

FILES

See the **bcg**(LOCAL) manual page for a description of the files.

SEE ALSO

bcg(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

Please report bugs to Hubert.Garavel@inria.fr