Resum

Per a la realització d’aquest projecte ha estat necessari la creació d’un nombre elevat de fitxers i scripts en llenguatge C. En aquest Annex es presenten els fitxers d’utilitats no associats a cap model i necessaris per “centralitzar” un seguit de definicions, opcions de configuració i necessari per mantenir una “base” de models.
Sumari

E. UTILITATS

5
### E. Utilitats

<table>
<thead>
<tr>
<th>Variable</th>
<th>MEAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSRR_1</td>
<td>15, 15, 0</td>
</tr>
<tr>
<td>ESSRR_2</td>
<td>11, 11, 0</td>
</tr>
<tr>
<td>ESSRR_3</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>ESSRR_4</td>
<td>6, 0, 6</td>
</tr>
<tr>
<td>ESSRR_5</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ESSRR_6</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ESSRR_7</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ESSRR_8</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>ESSRR_9</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>ESSRR_10</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>ESSRR_11</td>
<td>10, 6, 4</td>
</tr>
<tr>
<td>ESSRR_12</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ESSRR_13</td>
<td>1, 0, 1</td>
</tr>
<tr>
<td>ETRR_1</td>
<td>23, 23, 0</td>
</tr>
<tr>
<td>ETRR_2</td>
<td>45, 45, 0</td>
</tr>
<tr>
<td>ETRR_3</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ETRR_4</td>
<td>12, 12, 0</td>
</tr>
<tr>
<td>ETRR_5</td>
<td>9, 7, 2</td>
</tr>
<tr>
<td>ETRR_6</td>
<td>8, 8, 0</td>
</tr>
<tr>
<td>ETRR_7</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ETRR_8</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>ETRR_9</td>
<td>2, 2, 0</td>
</tr>
<tr>
<td>ETRR_10</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ETRR_11</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>ETRR_12</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ETRR_13</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ETRR_14</td>
<td>3, 0, 3</td>
</tr>
<tr>
<td>EARR_1</td>
<td>21, 21, 0</td>
</tr>
<tr>
<td>EARR_2</td>
<td>9, 0, 9</td>
</tr>
<tr>
<td>EARR_3</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>EARR_4</td>
<td>12, 12, 0</td>
</tr>
<tr>
<td>EARR_5</td>
<td>12, 12, 0</td>
</tr>
<tr>
<td>EARR_6</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>EARR_7</td>
<td>2, 2, 0</td>
</tr>
<tr>
<td>EARR_8</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>EARR_9</td>
<td>2, 0, 2</td>
</tr>
<tr>
<td>EARR_10</td>
<td>2, 1, 1</td>
</tr>
<tr>
<td>EARR_11</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>EARR_12</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>EARR_13</td>
<td>3, 0, 3</td>
</tr>
<tr>
<td>ECRTE_1</td>
<td>13, 7, 6</td>
</tr>
<tr>
<td>ECRTE_2</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>ECRTE_3</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>ECRTE_4</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>ECRTE_5</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>ECRTE_6</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>CRDTE_1</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>CRDTE_2</td>
<td>9, 9, 0</td>
</tr>
<tr>
<td>CRDTE_3</td>
<td>18, 18, 0</td>
</tr>
<tr>
<td>CRDTE_4</td>
<td>30, 30, 0</td>
</tr>
<tr>
<td>CRDTE_5</td>
<td>30, 30, 0</td>
</tr>
<tr>
<td>CRDTE_6</td>
<td>7, 7, 0</td>
</tr>
<tr>
<td>ECRRS_1</td>
<td>12, 12, 0</td>
</tr>
<tr>
<td>ECRRS_2</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>ECRRS_3</td>
<td>3, 2, 1</td>
</tr>
<tr>
<td>ECRRS_4</td>
<td>36, 36, 0</td>
</tr>
<tr>
<td>CRDDS_1</td>
<td>78, 78, 0</td>
</tr>
<tr>
<td>CRDDS_2</td>
<td>78, 78, 0</td>
</tr>
<tr>
<td>CRDDS_3</td>
<td>48, 48, 0</td>
</tr>
<tr>
<td>CRDDS_4</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>CRDDS_5</td>
<td>3, 3, 0</td>
</tr>
<tr>
<td>CRDDS_6</td>
<td>6, 6, 0</td>
</tr>
<tr>
<td>CRDDS_7</td>
<td>3, 0, 3</td>
</tr>
</tbody>
</table>
CRDDS_8, MEAS, 6, 6, 0
CIAV'D_1, MEAS, 46, 46, 0
CIAV'D_2, MEAS, 6, 6, 0
CIAV'D_3, MEAS, 1, 1, 0
CIAV'D_4, MEAS, 2, 2, 0
CIAV'D_5, MEAS, 1, 1, 0
CIAV'D_6, MEAS, 1, 1, 0
CIAV'D_7, MEAS, 1, 1, 0
CIAV'D_8, MEAS, 1, 1, 0
CIAV'D_9, MEAS, 1, 1, 0
CIAV'D_10, MEAS, 1, 1, 0
CIAV'D_11, MEAS, 1, 1, 0
CIAV'D_12, MEAS, 1, 1, 0
CIAV'D_13, MEAS, 1, 1, 0
CCRD_3, MEAS, 9, 9, 0
CCRD_2, MEAS, 3, 3, 0
CCRD_3, MEAS, 3, 3, 0
CCRD_4, MEAS, 3, 3, 0
CCRD_5, MEAS, 1, 1, 0
mgerror_0, MSG, 15, 15, 0
mgerror_1, MSG, 7, 7, 0
mgerror_2, MSG, 19, 19, 0
mgerror_3, MSG, 3, 3, 0
mgerror_4, MSG, 3, 3, 0
mgerror_5, MSG, 7, 7, 0
mgerror_6, MSG, 2, 2, 0
mgerror_7, MSG, 1, 1, 0
mgerror_8, MSG, 3, 3, 0
mgerror_9, MSG, 2, 2, 0
mgerror_10, MSG, 2, 2, 0
mgerror_11, MSG, 2, 2, 0
mgerror_12, MSG, 2, 2, 0
mgerror_13, MSG, 3, 3, 0
mgerror_14, MSG, 4, 4, 0
mgerror_15, MSG, 3, 3, 0
mgerror_16, MSG, 4, 4, 0
mgerror_17, MSG, 4, 4, 0
mgerror_18, MSG, 2, 2, 0
mgerror_19, MSG, 2, 2, 0
mgerror_20, MSG, 3, 3, 0
mgerror_21, MSG, 2, 2, 0
mgerror_22, MSG, 3, 3, 0
mgerror_23, MSG, 1, 1, 0
mgerror_24, MSG, 1, 1, 0
mgerror_25, MSG, 3, 3, 0
mgerror_26, MSG, 1, 1, 0
mgerror_27, MSG, 2, 2, 0
mgerror_28, MSG, 5, 5, 0
mgerror_29, MSG, 2, 2, 0
mgerror_30, MSG, 2, 2, 0
mgerror_31, MSG, 2, 2, 0
mgerror_32, MSG, 2, 2, 0
mgerror_33, MSG, 10, 10, 0
mgerror_34, MSG, 4, 4, 0
mgerror_35, MSG, 1, 1, 0
mgerror_36, MSG, 2, 2, 0
mgerror_37, MSG, 2, 2, 0
mgerror_38, MSG, 2, 2, 0
mgerror_39, MSG, 2, 2, 0
mgerror_40, MSG, 2, 2, 0
mgerror_41, MSG, 2, 2, 0
mgerror_42, MSG, 3, 3, 0
mgerror_43, MSG, 1, 1, 0
mgerror_44, MSG, 3, 3, 0
mgerror_45, MSG, 3, 3, 0
mgerror_46, MSG, 1, 1, 0
mgerror_47, MSG, 4, 4, 0
msgerror_48, MSG, 2, 2, 0
msgerror_49, MSG, 5, 5, 0
msgerror_50, MSG, 3, 3, 0
msgerror_51, MSG, 4, 4, 0
msgerror_52, MSG, 1, 1, 0
msgerror_53, MSG, 1, 1, 0
msgerror_54, MSG, 1, 1, 0
msgerror_55, MSG, 2, 2, 0
msgerror_56, MSG, 1, 1, 0
msgerror_57, MSG, 1, 1, 0
msgerror_58, MSG, 7, 7, 0
msgerror_59, MSG, 1, 1, 0
msgerror_60, MSG, 4, 4, 0
msgerror_61, MSG, 1, 1, 0
msgerror_62, MSG, 5, 5, 0
msgerror_63, MSG, 2, 2, 0
msgerror_64, MSG, 1, 1, 0
msgerror_65, MSG, 1, 1, 0
modgen_1, MODGEN, 4, 4, 0
modgen_2, MODGEN, 4, 4, 0
modgen_3, MODGEN, 2, 2, 0
/
*     * Header of the driver_lib.c file.
*     */
#if !defined(DRIVER_LIB_H)
define DRIVER_LIB_H
#include <stdarg.h> /* for the ellipsis '...' */
#include <stdio.h> /* for FILE* */
#include "test_conf.h"
*/ set of messages */
typedef struct msg_set_ msg_set_t;
/
* Returns the operating system version, which can be any of: OS_LINUX
* (Linux), OS_SOLARIS (Solaris), and OS_UNKNOWN (not known).
*     * NOTE: the OS type is read from the env. var. "METFAC2_OS".
*     */
extern
int
os_type(void);
/
* Returns the version of the Valgrind program, which can be any of:
* VG_20030716 (version 20030716), VG_1_9_6 (version 1.9.6), VG_2_0_0 (version
* 2.0.0), VG_UNKNOWN (version not known), and VG_NOTFOUND (Valgrind is not
* available).
*     */
extern
int
valgrind_version(void);
/
* Returns the version of the gcov program, which can be any of: GCOV_303p
* (version 303p), GCOV_1_5 (version 1.5), GCOV_2_9_5 (version 2.9.5),
* GCOV_UNKNOWN (version not known), and GCOV_NOTFOUND (gcov is not
* available).
* /
extern
int
gcov_version(void);

/*
 * Builds up in the string comm the command to call the executable file that
 * results from compiling a model. modnam holds the model name, vectinp holds
 * the file the executable will read parameters from, and vectout holds the
 * file where the executable will print its messages, if any. If the
 * environment variable M2TEST_MEMDBG is defined and set to anything different
 * from the empty string, the executable file will be called through a memory
 * debugger: Valgrind if the operating system is Linux and bcheck if the OS is
 * Solaris. The debugger's report will be printed to the file *dbginfo. If
 * M2TEST_MEMDBG is undefined or has been set to the empty string, no memory
 * debugger will be used and the value of *dbginfo is irrelevant.
 */

extern
void
build_met2comm(char *comm, const char *vectnam, const char *vectinp,
        const char *vectout, const char *valgout);

/*
 * Given a model called *modname, builds up in *fname the name of the exec
 * file of the corresponding driver. Such a name consists of the string to
 * which the macro DRIVER_FILE_PREFIX expands followed by the contents of
 * *modname.
 * 
 * NOTE: the macro DRIVER_FILE_PREFIX is defined in test_conf.h.
 */

extern
void
build_driver_filename(const char *modname, char *fname);

/*
 * Stores in par_set[] and prints to file *fp the set of parameters given next
 * to the 'format' argument. Those parameters are given "a la" printf(). Thus,
 * for a model having two parameters named SIZE and LAMBDAS, the first one of
 * type integer and the second of type double, the call to this function might
 * look as follows:
 * 
 * build_parset(filepointer, stringpointer, "%d\n%g\n", valueofSIZE,
 *              valueofLAMBDAS);
 */

extern
void
build_parset(FILE *fp, char parset[], const char *format, ...);

/*
 * Given a model's name *modname and an integer n val>0, opens the file whose
 * name consists of the string to which the macro EXACT_VAL_FILE_SUFFIX
 * expands appended to *modname, and reads from it n val different numerical
 * values. Upon success, val[1], ..., val[n val] hold the read values and 0 is
 * returned; upon failure, the function returns an integer <>0 and the
 * contents of val[] is undefined.
 * 
 * NOTES: the macro EXACT_VAL_FILE_SUFFIX is defined in test_conf.h; in the
 * file, there must be a single value per line (empty lines are allowed,
 * though).
 */

extern
int
read_exact_values_meas(const char *modname, int n_val, double val[]);

/*
 * Given a model called *modname, opens the log file of the corresponding
 * driver and returns a pointer to it. If opening the file fails, returns
 * NULL.
 */
extern
FILE *
open_driver_logfile(const char *modname);

/*
 * Given the name of a measure in *measname, sets *cd to the numerical code
 * the measure has in METFAC2 and returns the integer 0; if *measname is not a
 * valid measure name, returns an integer <>0 and the contents of *cd is
 * undefined.
 */
extern
int
meas_num_code(const char *measname, int *cd);

/*
 * Given a model's name *modname, a measure's name *measname, a string *eqstr,
 * and an integer n_val>0, opens the log file that resulted from running the
 * model and reads from it n_val numerical values of the measure. The function
 * assumes that in the log file the values of the measure are given one per
 * line in the form:
 * *
 * *measname *eqstr <value>
 * *
 * and, therefore, typically the function will be called with *eqstr equal to
 * " = " , " < " , or " >= " . If all the values are read successfully, the
 * function returns 0 and the read values in val[1],
 * ... val[n_val]. Otherwise, the function returns an integer <>0 and the
 * contents of val[] is undefined.
 */
extern
int
read_m2_val_meas(const char *modname, const char *measname,
                 const char *eqstr, int n_val, double val[]);

/*
 * Given a model's name *modname, a measure's name *measname, a string
 * *absname, and an integer n_p>0, opens the log file that resulted from
 * running the model and reads from it n_p pairs of numerical values of bounds
 * for the measure, (lower bound, upper bound). The function assumes that in
 * the log file such pairs appear one per line in the form
 * *
 * *absname " = " <value> <lower_bound> " <= " *measname " <= " <upper_bound>
 * *
 * If all the pairs of bounds are read successfully, the function returns 0,
 * the lower bounds in lb_val[1], ..., lb_val[n_p], and the upper bounds in
 * ub_val[1], ..., ub_val[n_p]. Otherwise, the function returns an integer
 * <>0 and the contents of both lb_val[] and ub_val[] is undefined.
 */
extern
int
read_m2_bounds_meas(const char *modname, const char *measname,
const char *absname, int n_p, double lb_val[],
double ub_val[]);

/*
 * Prints to the file pointed to by *fp the identity of the vector defined by
 * the model's name *modname and the set of parameters *par. Useful for tasks
 * where no numerical method is chosen (i.e. generation of the model).
 */
extern
void
print_ident_vector_zero_meth(const char *modname, const char *par, FILE *fp);

/*
 * Prints to the logfile pointed to by *fp the identity of the vector defined
 * by the model's name *modname and the set of parameters *par. *measname is
 * the name of the measure, meth is the code of the numerical method, and
 * max_err is the maximum error/tolerance. Useful for measures that use one
 * numerical method.
 */
extern
void
print_ident_vector_one_meth(const char *modname, const char *par,
const char *measname, int meth,
double max_err, FILE *fp);

/*
 * Does the same as print_ident_vector_one_meth() for measures that use two
 * numerical methods.
 */
extern
void
print_ident_vector_two_meth(const char *modname, const char *par,
const char *measname, int meth1, int meth2,
double max_err1, double max_err2, FILE *fp);

/*
 * Does the same as print_ident_vector_one_meth() for measures that use three
 * numerical methods.
 */
extern
void
print_ident_vector_three_meth(const char *modname, const char *par,
const char *measname, int meth1, int meth2,
int meth3, double max_err1, double max_err2,
double max_err3, FILE *fp);

/*
 * Given in meth the numerical method with which the measure ETRR has been
 * computed (see below), in exact_val and met2_val the exact and computed
 * values, respectively, in max_err the maximum error/tolerance with which the
 * computation has been performed, and in fp a pointer to the log file of the
 * driver, checks whether the computed value is ok, printing to the file the
 * appropriate message. Returns 0 if meth is a valid method and an integer <>0
 * otherwise.
 * NOTE: use check_result_ETRR_b() for Bounding Regenerative Randomization.
 */
extern
int check_result_ETRR(int meth, double max_err, double exact_val, double met2_val, 
                      FILE *fp);

/*
 * Given in exact_val the exact value of the measure ETRR, in met2_bound the 
 * computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum 
 * error with which the computation has been performed, and in fp a pointer to 
 * the log file of the driver, checks whether the bound is ok, printing to the 
 * file the appropriate message. Returns 0 if lb is either TRUE or FALSE and 
 * an integer <>0 otherwise.
*/
extern
int check_result_ETRR_b(int lb, double max_err, double exact_val, double met2_bound, 
                         FILE *fp);

/*
 * Given in meth1 and meth2 the numerical methods with which the measure ESSRR 
 * has been computed (see below), in exact_val and met2_val the exact and 
 * computed values, respectively, in max_err1 and max_err2 the maximum 
 * error/tolerance for meth1 and meth2, respectively, with which the 
 * computation has been performed, and in fp a pointer to the log file of the 
 * driver, checks whether the computed value is ok, printing to the file the 
 * appropriate message. Returns 0 if meth1 and meth2 are valid methods and an 
 * integer <>0 otherwise.
*/
extern
int check_result_ESSRR(int meth1, int meth2, double max_err1, double max_err2, 
                        double exact_val, double met2_val, FILE *fp);

/*
 * Given in meth the numerical method with which the measure EARR has been 
 * computed, in exact_val and met2_val the exact and computed values, 
 * respectively, in max_err the maximum error/tolerance with which the 
 * computation has been performed, and in fp a pointer to the log file of the 
 * driver, checks whether the computed value is ok, printing to the file the 
 * appropriate message. Returns 0 if meth is a valid method and an integer <>0 
 * otherwise.
*/
extern
int check_result_EARR(int meth, double max_err, double exact_val, double met2_val, 
                       FILE *fp);

/*
 * Given in meth the numerical method with which the measure CCRD has been 
 * computed, in exact_val and met2_val the exact and computed values, 
 * respectively, in max_err the maximum error with which the computation has 
 * been performed, and in fp a pointer to the log file of the driver, checks 
 * whether the computed value is ok, printing to the file the appropriate 
 * message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
*/
extern
int check_result_CCRD(int meth, double max_err, double exact_val, double met2_val, 
                       FILE *fp);
/*
 * Given in meth the numerical method with which the measure CIAVD has been
 * computed, in exact_val and met2_val the exact and computed values,
 * respectively, in max_err the maximum error with which the computation has
 * been performed, and in fp a pointer to the log file of the driver, checks
 * whether the computed value is ok, printing to the file the appropriate
 * message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
 */

extern int
check_result_CIAVD(int meth, double max_err, double exact_val, double met2_val, 
FILE *fp);

/*
 * Given in meth the numerical method with which the measure ECRTE has been
 * computed, in exact_val and met2_val the exact and computed values,
 * respectively, in max_err the maximum error with which the computation has
 * been performed, and in fp a pointer to the log file of the driver, checks
 * whether the computed value is ok, printing to the file the appropriate
 * message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
 * NOTE: use check_result_CRDTE_b() for Bounding Regenerative Randomization.
 */

extern int
check_result_ECRTE(int meth, double max_err, double exact_val, double met2_val, 
FILE *fp);

/*
 * Given in meth the numerical method with which the measure CRDTE has been
 * computed (see below), in exact_val and met2_val the exact and computed
 * values, respectively, in max_err the maximum error with which the
 * computation has been performed, and in fp a pointer to the log file of the
 * driver, checks whether the computed value is ok, printing to the file the
 * appropriate message. Returns 0 if meth is a valid method and an integer <>0
 * otherwise.
 */

extern int
check_result_CRDTE(int meth, double max_err, double exact_val, double met2_val, 
FILE *fp);

/*
 * Given in exact_val the exact value of the measure CRDTE, in met2_bound the
 * computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum
 * error with which the computation has been performed, and in fp a pointer to
 * the log file of the driver, checks whether the bound is ok, printing to the
 * file the appropriate message. Returns 0 if lb is either TRUE or FALSE and
 * an integer <>0 otherwise.
 */

extern int
check_result_CRDTE_b(int lb, double max_err, double exact_val, double met2_bound, 
FILE *fp);

/*
 * Given in meth1 and meth2 the numerical methods with which the measure ECRDS
 * has been computed (see below), in exact_val and met2_val the exact and
* computed values, respectively, in max_err1 and max_err2 the maximum
* error/tolerance for meth1 and meth2, respectively, with which the
* computation has been performed, and in fp a pointer to the log file of the
* driver, checks whether the computed value is ok, printing to the file the
* appropriate message. Returns 0 if meth1 and meth2 are valid methods and an
* integer <>0 otherwise.
*/

extern int
check_result_ECRDS(int meth1, int meth2, double max_err1, double max_err2,
                    double exact_val, double met2_val, FILE *fp);

/*
 * Given in meth1, meth2, and meth3 the numerical methods with which the
 * measure CRDDS has been computed (see below), in exact_val and met2_val the
 * exact and computed values, respectively, in max_err1, max_err2, and
 * max_err3 the maximum error/tolerance for meth1, meth2, and meth3,
 * respectively, with which the computation has been performed, and in fp a
 * pointer to the log file of the driver, checks whether the computed value is
 * ok, printing to the file the appropriate message. Returns 0 if meth1,
 * meth2, and meth3 are valid methods and an integer <>0 otherwise.
*/

extern int
check_result_CRDDS(int meth1, int meth2, int meth3, double max_err1,
                    double max_err2, double max_err3, double exact_val,
                    double met2_val, FILE *fp);

/*
 * Given in exact_val the exact value of the measure CRDDS, in met2_bound the
 * computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum
 * error with which the computation has been performed, and in fp a pointer to
 * the log file of the driver, checks whether the bound is ok, printing to the
 * file the appropriate message. Returns 0 if lb is either TRUE or FALSE and
 * an integer <>0 otherwise.
*/

extern int
check_result_CRDDS_b(int lb, double max_err, double exact_val, double met2_bound,
                      FILE *fp);

/*
 * Given a model name *modname, reads from the appropriate file a set of
 * expected error and warning messages, returning them in the newly allocated
 * data structures **experrp and **expwarp, respectively. The name of the file
 * is built by appending to *modname the string that results from the
 * concatenation of: EXPECTED_MSG_FILE_SUFFIX, an underscore, and the integer
 * vect_idx translated into a string. The function assumes that in the file,
 * the body of each error message is enclosed by EXP_ERR_MSG_BEGIN and
 * EXP_ERR_MSG_END, and the body of each warning message is enclosed by
 * EXP_WARN_MSG_BEGIN and EXP_WARN_MSG_END. The function returns 0 on success
 * and 1 on failure.
 * NOTE: lines that are empty according to empty_line() are discarded.
 * NOTE: EXPECTED_MSG_FILE_SUFFIX, EXP_ERR_MSG_BEGIN, EXP_ERR_MSG_END,
 * EXP_WARN_MSG_BEGIN, and EXP_WARN_MSG_END are macros defined in the
 * test_conf.h file.
*/

extern int
read_expected_msgs(const char *modname, const int vect_idx,
msg_set_t **experrp, msg_set_t **expwarp);

/*
 * Given a model name *modname, reads from the appropriate file the set of
 * error and warning messages generated by the model, returning them in the
 * newly allocated data structures **met2errp and **met2warp,
 * respectively. The name of the file is built by appending the string ".log"
 * to *modname. The function assumes that each message has the following
 * structure:
 *<anything> <header_beg> <anything> <header_end> <first_meaningful_sentence>
 *<zero_or_more_meaningful_sentences_mixed_with_empty_lines>
 *where for error messages, <header_beg> and <header_end> are, respectively,
 *the strings to which the macros ERR_MSG_HEADER_BEGIN and ERR_MSG_HEADER_END
 *expand, and for warning messages, <header_beg> and <header_end> are,
 *respectively, the strings to which the macros WAR_MSG_HEADER_BEGIN and
 *WAR_MSG_HEADER_END expand.
 *The function returns 0 on success and 1 on failure.
 *NOTE: ERR_MSG_HEADER_BEGIN, ERR_MSG_HEADER_END, WAR_MSG_HEADER_BEGIN, and
 *WAR_MSG_HEADER_END are macros defined in the test_conf.h file.
 */

extern int
read_met2_msgs(const char *modname, msg_set_t **met2errp,
msg_set_t **met2warp);

/*
 * Given a set *met2p of read messages and a set *expp of expected ones,
 * compares the first against the second, prints appropriate message(s) to the
 * file *fp according to the results of the comparison, and frees both
 * sets. The function returns 0 on success and 1 on failure.
 *NOTE: if both sets are empty, the function frees them and returns 0 without
 *doing anything else.
 *NOTE: the function works as follows. For each read message in *met2p, it
 *looks for an expected message in *expp such that letting n be its number of
 *sentences, all the n sentences of the expected message match up in a
 *one-by-one fashion with the first n sentences of the read message
 *regardless dots, commas, colons, semicolons, blank spaces, tabs, and new
 *line characters. If such an expected message is found, the read one is
 *regarded as correct; otherwise, the read message is regarded as incorrect.
 */

extern int
check_msgs(msg_set_t *met2p, msg_set_t *expp, FILE *fp);

/*
 * Looks for expmsg[] or nonexpmsg[] in the log file of the model modname[],
 * printing PASS_MSG to the file *outfp if the string found is expmsg[] and
 * FAIL_MSG if the string found is nonexpmsg[]. Returns 0 on success and 1 on
 * failure (e.g. the log file contains neither of those strings).
 *NOTE: PASS_MSG and FAIL_MSG are macros defined in test_conf.h.
 */

extern int
check_msg_logfile(const char modname[], const char expmsg[],
const char nonexpmsg[], FILE *outfp);

/*
 * Looks in the log file of the model modname[] for any of the strings to
 * which the macros CTMCGEN_OK_MSG and CTMCGEN_BAD_MSG expand to. (Currently,
 * "Test of the CTMC: PASS" and "Test of the CTMC: FAIL." If it is found the
 * first string, which signals that the generation of the CTMC has been
 * checked and is okay, prints to the file *outfp the string yielded by the
 * macro DRIVER_LOG_BEG_CTMCGENRES, the string yielded by the macro PASS_MSG,
 * and the string yielded by the macro DRIVER_LOG_END_CTMCGENRES. (Currently,
 * "test of the CTMC: start", "PASS", and "test of the CTMC: end.") If the
 * string found is the second one, which tells us that the CTMC is not okay,
 * prints the same strings but FAIL_MSG (currently, "FAIL") instead of
 * PASS_MSG. And if neither string is found, which signals that the generation
 * of the CTMC has not be checked, prints nothing. The function returns 0 on
 * success and an integer <>0 on failure.
 */

extern
int
check_Markov_chain(const char modname[], FILE *outfp);

/*
 * Frees the set of models.
 */

extern
void
free_modset(void);

/*
 * Reads from file *fname the set of models. The file has to contain a
 * non-empty line per model (empty lines are discarded) with the following
 * structure:
 *<name><sep>"MEAS,""MSG,""MODGEN,"<vect><sep><#ok_vect><sep><#wrong_vect>
 * where:
 *<name>: model name
 *<sep>: one or more of: white space, comma, period, colon, semicolon, tab
 * MEAS: the model is intended for testing the computation of measures
 * MSG: the model is intended for testing warning and error messages
 * MODGEN: the model is intended for testing the generation of the model
 *<#vect>: no. of vectors to be generated based on the model
 *<#ok_vect>: no. of generated vectors that are supposed to pass the
 * corresponding test
 *<#wrong_vect>: no. of generated vectors that are supposed not to pass the
 * corresponding test
 * The function returns 0 if the list has been read successfully and an
 * integer <>0 otherwise.
 */

extern
int
read_mod_set(const char fname[]);
/ * Prints to the standard output a one-line list of models of type tp sorted * in increasing order of the name of the model. */

extern int pr_modset(int tp);

/ * If the environment variable M2TEST_MEMDBG is defined and set to anything * different from the empty string, does the following: opens file 'from; * appends to the file 'tofp the string 'beg_str, the contents of file 'from, * and the string 'end_str; _removes_ file 'from; returns 0 if everything went * okay and an integer <>0 on failure. Otherwise, the function does nothing * and returns 0. */

extern int append_file(FILE *tofp, char *from, char *beg_str, char *end_str);

/ * Given the name of a model in *modname, looks for errors in each run of the * model that has been reported in the log file of the corresponding model * driver. For each such run, the function prints (standard output) the string * to which the macro PASS_MSG expands if no errors are found, and the string * to which the macro FAIL_MSG expands plus some other info otherwise. In * addition, once all runs have been processed, the function prints a brief * summary telling how many runs have been examined, in how many of them one * or more errors have been found, and whether those numbers agree with the * expected ones read from the file 'test_conf.modset'. The function returns 0 * on success and an integer <>0 upon failure. */

extern int lookfor_errors(const char *modname);

#endif /* DRIVER_LIB_H */

/* Small utility to print to standard output the list of names of models. */
/* Usage: */
/*  gen_model_list <mode> */
/*  <mode>=0: prints the list of models for testing the computation */
/*  of measures */
/*  <mode>=1: prints the list of models for testing warning and error messages */
/*  <mode>=2: prints the list of models for testing generation of a model */

#include <stdlib.h>
#include <stdio.h>

#include "driver_lib.h"

int main(int argc, char *argv[]) {
  int mode, kind;


if (argc != 2) goto L1;
(void) sscanf(argv[1], "%d", &mode);
if (read_mod_set(MOD_LIST_FILE) != 0) goto L1;
switch (mode) {
  case 0:
    kind = MOD_MEAS;
    break;
  case 1:
    kind = MOD_ERRWAR;
    break;
  case 2:
    kind = MOD_MODGEN;
    break;
  default:
    goto L1;
    break;
}
if (pr_modset(kind) != 0) goto L1;
free_modset();
return EXIT_SUCCESS;
L1:
return EXIT_FAILURE;

/*
 * If the environment variable MZTEST_CODECOV is defined and is not set to
 * the empty string, this program calls gcov on each of the source files of the
 * 'tool' directory of METFAC2.1 that are being tested and prints the
 * corresponding coverage statistics on the standard output (see below). Otherwise, the program does nothing. Apart from MZTEST_CODECOV, the
 * behavior of the program is also affected by the environment variables
 * MZTEST_GCOVFLAGS and METFAC2. The MZTEST_GCOVFLAGS variable contains the
 * flags for the gcov program. (If the variable is empty or not defined, gcov
 * will be called with no flags.) The METFAC2 variable must hold the full path
 * to the root directory of METFAC2.1.
 * Coverage statistics format:
 * 
 * <file_name>: i:<x> b:<y>({z}) c:<t>
 * 
 * where:
 * 
 * <file_name>: name of the source file.
 * 
 * <x>: percent of lines of <file_name> that have been executed.
 * 
 * <y>: percent of branches of <file_name> that have been executed.
 * 
 * <z>: percent of branches of <file_name> that have been taken at least once.
 * 
 * <t>: percent of function calls in <file_name> that have been executed.
 * 
 * Given the way gcov works, every time this program is called, the resulting
 * statistics take into account all the METFAC2 .exe files that have been
 * executed since either METFAC2 was compiled or the utility init_cov was
 * called. Calling this program also results in creating a .gcov file for each
 * METFAC2 source file on which gcov is called. Such .gcov files are also
 * cumulative.
 * 
 * Available compilation macros:
 * 
 * DBG_GET_COV: For debugging.
 * 
 * NOTE: the program uses the program-scope variable n_source_files defined in
 * driver_lib.c */
/* System headers. */

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/* Custom headers. */
#include "driver_lib.h"

/* Macros. */
#define GCOV_OUT_TMP_FILE "gcov_results_tmp"
#define GCOV_CAT_FILE "gcov_results_cat"

#define GCOV_OUT_LINE_IDENT_1_5 "source lines executed"
#define GCOV_OUT_BRANCH_IDENT_1_5 "branches executed"
#define GCOV_OUT_TAKENBRANCH_IDENT_1_5 "branches taken at least once"
#define GCOV_OUT_CALL_IDENT_1_5 "calls executed"

#define GCOV_OUT_LINE_IDENT_2_9_5 "source lines executed"
#define GCOV_OUT_BRANCH_IDENT_2_9_5 "branches executed"
#define GCOV_OUT_TAKENBRANCH_IDENT_2_9_5 "branches taken at least once"
#define GCOV_OUT_CALL_IDENT_2_9_5 "calls executed"

#define GCOV_OUT_FILE_IDENT_303p "File"
#define GCOV_OUT_LINE_IDENT_303p "Lines executed:"
#define GCOV_OUT_BRANCH_IDENT_303p "Branches executed:"
#define GCOV_OUT_TAKENBRANCH_IDENT_303p "Taken at least once:"
#define GCOV_OUT_CALL_IDENT_303p "Calls executed:"

int
main(void)
{
    char *met_root, *gcov_flg, tmp[BUF_SZ], gcov_prefix[BUF_SZ], gcov_out_dir[BUF_SZ], gcov_out_file[BUF_SZ], *ret, *s, line[LINEWIDTH+2];
    FILE *fp;
    int ii, st, gcov_vers, os,
    have_b_flag;  /* TRUE if gcov is invoked with the '-b'
       * flag and FALSE otherwise */
    float info[MAX_SOURCE_FILES][4]; /* info[i][0]: percent of lines executed
    * in file source_files[i];
    * * info[i][1]: percent of branches
    * executed in file source_files[i];
    * * info[i][2]: percent of branches taken
    * at least once in file source_files[i];
    * * info[i][3]: percent of function calls
    * executed in file source_files[i] */

    /* set-up */
    s = getenv("M2TEST_CODECOV");
    if ( !(s && *s) ) {  /* no code coverage info has been generated: return */
           return EXIT_SUCCESS;
    }
    if ( (os = os_type()) == OS_UNKNOWN ) {
    /* unknown OS */
    (void) fprintf(stderr, "get_cov: unable to find out OS.\n")
    return EXIT_FAILURE;
    }

    /* ... continue with code ... */
gcov_version(); /* gcov version */
if (gcov_version != "GOV_UNKNOWN") {
    (void) fprintf(stderr, "get_cov: unknown gcov version.\n");
    return EXIT_FAILURE;
} else if (gcov_version == "GOV_NOTFOUND") {
    (void) fprintf(stderr, "get_cov: gcov is not available.\n");
    return EXIT_FAILURE;
}
gcovflags = getenv("M2TEST_GCOVFLAGS"); /* gcov flags */
if (!(met_root = getenv("METFAC2"))) { /* METFAC2's root directory */
    (void) fprintf(stderr, "get_cov: unable to find env. var. 'METFAC2'.\n");
    return EXIT_FAILURE;
}

/* build up the system command to invoke gcov */
strncpy(gcov_prefix, "gcov ");
if (gcovflags) strcat(gcov_prefix, gcovflags);
strcat(gcov_prefix, " -o ");
strcat(gcov_prefix, met_root);
strcat(gcov_prefix, "/bin/");
if (os == OS_LINUX) strcat(gcov_prefix, LINUX_BIN_DIR);
else strcat(gcov_prefix, SOLARIS_BIN_DIR);
strcat(gcov_prefix, " ");
strcat(gcov_prefix, met_root);
strcat(gcov_prefix, "/src/tool/");

/* build up the name of the directory where gcov will store the raw
 coverage info */
strncpy(gcov_out_dir, met_root);
strcat(gcov_out_dir, "/bin/");
if (os == OS_LINUX) strcat(gcov_out_dir, LINUX_BIN_DIR);
else strcat(gcov_out_dir, SOLARIS_BIN_DIR);
strcat(gcov_out_dir, "/");
strcat(gcov_out_dir, GCOV_OUT_DIR);
strcat(gcov_out_dir, "/");

/* build up the name of the file to which gcov output will be redirected */
strncpy(gcov_out_file, gcov_out_dir);
strcat(gcov_out_file, GCOV_OUT_FILE);

/* should we expect branch probabilities info? */
if (gcovflags & "-b") have_b_flag = TRUE;
else have_b_flag = FALSE;

/* call gcov on each METFAC2.i's source file, storing gcov's output in the
 * file gcov_out_file */
for (i = 0; i < n_source_files; i++) {
    strcpy(tmp, gcov_prefix);
    strcat(tmp, source_files[i]);
    if (os == OS_LINUX) strcat(tmp, ".c > ");
    else strcat(tmp, ".c l > ");
    strcat(tmp, gcov_out_dir);
    if (i == 0) strcat(tmp, GCOV_OUT_FILE);
    else strcat(tmp, GCOV_OUT_TMP_FILE);
    system(tmp);
    if (i > 0) {
        strcat(tmp, "cat ");
        strcat(tmp, gcov_out_file);
        strcat(tmp, " ");
        strcat(tmp, gcov_out_dir);
        if (os == OS_LINUX) strcat(tmp, " > ");
        else strcat(tmp, " l > ");
        strcat(tmp, gcov_out_dir);
        system(tmp);
        strcat(tmp, "\mv ");
        strcat(tmp, gcov_out_dir);
strcat(tmp, GCOV_CAT_FILE);
strcat(tmp, ",");
strcat(tmp, gcov_out_file);
system(tmp);
}
strcpy(tmp, "mv ");
strcat(tmp, source_files[ii]);
strcat(tmp, ".c.gcov ");
strcat(tmp, gcov_out_dir);
system(tmp);

/* process gcov_out_file so that it will have just one line per each * METFAC2's file */
if (!fp = fopen(gcov_out_file, "r")) {
    (void) fprint(stderr, "get_cov: unable to open file '%%s'.
",
gcov_out_file);
    return EXIT_FAILURE;
}
switch (gcovVers) {
    case GCOV_303p:
        ret = fgets(line, LINEWIDTH+2, fp);
        ii = 0;
        st = 0;
        while (ret && ii < nSourceFiles) {
            switch (st) {
            case 0:
                if (strstr(line, GCOV_OUT_FILE_IDENT_303p)) st = 1;
                break;
            case 1:
                if ((s = strstr(line, GCOV_OUT_LINE_IDENT_303p))) {
                    ifdef(DBG_GET_COV)
                        (void) printf("Line info: %%s\n", line);
                    endif
                    sscanf(s+strlen(GCOV_OUT_LINE_IDENT_303p), "%f",
                        &info[ii][0]);
                    if (haveBFlag) st = 2;
                    else ++ii;
                    break;
            case 2:
                if ((s = strstr(line, GCOV_OUT_BRANCH_IDENT_303p))) {
                    ifdef(DBG_GET_COV)
                        (void) printf("Branch info: %%s\n", line);
                    endif
                    sscanf(s+strlen(GCOV_OUT_BRANCH_IDENT_303p), "%f",
                        &info[ii][1]);
                    st = 3;
                    break;
            case 3:
                if ((s = strstr(line, GCOV_OUT_TAKENBRANCH_IDENT_303p))) {
                    ifdef(DBG_GET_COV)
                        (void) printf("Taken branch info: %%s\n", line);
                    endif
                    sscanf(s+strlen(GCOV_OUT_TAKENBRANCH_IDENT_303p), "%f",
                        &info[ii][2]);
                    st = 4;
                    break;
            case 4:
                if ((s = strstr(line, GCOV_OUT_CALL_IDENT_303p))) {
                    ifdef(DBG_GET_COV)
                        (void) printf("Call info: %%s\n", line);
                    endif
                    sscanf(s+strlen(GCOV_OUT_CALL_IDENT_303p), "%f",
                        &info[ii][3]);
                    }
```
    ++ii;
    st = 0;
  }
  }
  ret = fgets(line, LINEWIDTH+2, fp);
  break;
  case GCOV_1_5:
    ret = fgets(line, LINEWIDTH+2, fp);
    ii = 0;
    st = 1;
    while (ret && ii < n_source_files) {
      switch (st) {
        case 1:
          if ((s = strstr(line, GCOV_OUT_LINE_IDENT_1_5))) {
            #if defined(DBG_GET_COV)
              (void) printf("Line info: %s\n", line);
            #endif
              sscanf(line+1, "%f", &(info[ii][0]));
              if (have_b_flag) st = 2;
              else ++ii;
            }
            break;
        case 2:
          if ((s = strstr(line, GCOV_OUT_BRANCH_IDENT_1_5))) {
            #if defined(DBG_GET_COV)
              (void) printf("Branch info: %s\n", line);
            #endif
              sscanf(line+1, "%f", &(info[ii][1]));
              st = 3;
            }
            break;
        case 3:
          if ((s = strstr(line, GCOV_OUT_TAKENBRANCH_IDENT_1_5))) {
            #if defined(DBG_GET_COV)
              (void) printf("Taken branch info: %s\n", line);
            #endif
              sscanf(line+1, "%f", &(info[ii][2]));
              st = 4;
            }
            break;
        case 4:
          if ((s = strstr(line, GCOV_OUT_CALL_IDENT_1_5))) {
            #if defined(DBG_GET_COV)
              (void) printf("Call info: %s\n", line);
            #endif
              sscanf(line+1, "%f", &(info[ii][3]));
            ++ii;
            st = 1;
          }
        }
        ret = fgets(line, LINEWIDTH+2, fp);
      } break;
    case GCOV_2_9_5:
      ret = fgets(line, LINEWIDTH+2, fp);
      ii = 0;
      st = 1;
      while (ret && ii < n_source_files) {
        switch (st) {
          case 1:
            if ((s = strstr(line, GCOV_OUT_LINE_IDENT_2_9_5))) {
              #if defined(DBG_GET_COV)
                (void) printf("Line info: %s\n", line);
              #endif
                sscanf(line+1, "%f", &(info[ii][0]));
                if (have_b_flag) st = 2;
                else ++ii;
```
} break;
case 2:
  if ((s = strstr(line, GCOV_OUT_BRANCH_IDENT_2_9_5))) {
    #if defined(DBG_GET_COV)
      (void) printf("Branch info: \%s\n", line);
    #endif
    sscanf(line+1, "%f", &info[ii][1]);
    st = 3;
  } break;
case 3:
  if ((s = strstr(line, GCOV_OUT_TAKENBRANCH_IDENT_2_9_5))) {
    #if defined(DBG_GET_COV)
      (void) printf("Taken branch info: \%s\n", line);
    #endif
    sscanf(line+1, "%f", &info[ii][2]);
    st = 4;
  } break;
case 4:
  if ((s = strstr(line, GCOV_OUT_CALL_IDENT_2_9_5))) {
    #if defined(DBG_GET_COV)
      (void) printf("Call info: \%s\n", line);
    #endif
    sscanf(line+1, "%f", &info[ii][3]);
    ++ii;
    st = 1;
  }
  ret = fgets(line, LINEWIDTH+2, fp);
} while (1);
break;
}
(void) fclose(fp);

if (ii < n_source_files) {
  (void) fprintf(stderr, "get_cov: file '%s' is empty or has not enough
                   info.\n",
                   gcov_out_file);
  return EXIT_FAILURE;
}
(void) remove(GCOV_OUT_TMP_FILE);

if (have_b_flag) {
  for (ii = 0; ii < n_source_files; ii++) {
    (void) printf("%s: \tl.: %g; ", source_files[ii], info[ii][0]);
    if (info[ii][0] > 0.0) {
      (void) printf("%tb.: %g (%g); ", info[ii][1], info[ii][2]);
    } else {
      (void) printf("%tb.: %g (%g); ", info[ii][1], info[ii][2]);
    }
    if (info[ii][2] > 0.0) {
      (void) printf("%tc: %g.\n", info[ii][3]);
    } else {
      (void) printf("%tc: %g.\n", info[ii][3]);
    }
  }
}
else {
  for (ii = 0; ii < n_source_files; ii++) {
    (void) printf("%s: \tl.: %g.\n", source_files[ii], info[ii][0]);
  }
}

return EXIT_SUCCESS;

/**
 * Deletes the .da and .gcov files that have resulted (the .da files from
* compiling METFAC2.1 with the -fprofile-arcs and -ftest-coverage flags and
* (the .gcov files) from calling get_cov. The coverage statistics given by
* get_cov take into account all the METFAC2.exe files that have been run
* since either METFAC2.1 itself was compiled or this program was called.
* *
* NOTE: the program reads the full path to the root directory of METFAC2.1
* from the environment variable METFAC2.
* *
* NOTE: the program uses the program-scope variable n_source_files defined in
* driver_lib.c
*/

/** System headers. */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/** Custom headers. */
#include "driver_lib.h"

int
main(void)
{
    char *met_root, tmp[BUF_SZ];
    int i, os;

    if (!met_root = getenv("METFAC2")) { /* METFAC2's root directory */
        (void) fprintf(stderr, "init_cov: unable to find env. var. 'METFAC2'.\n");
        return EXIT_FAILURE;
    }
    if ((os = os_type()) == OS_UNKNOWN) {
        /* unknown OS */
        (void) fprintf(stderr, "init_cov: unable to find out OS.\n");
        return EXIT_FAILURE;
    }

    for (i = 0; i < n_source_files; i++) {
        strcpy(tmp, met_root);
        strcat(tmp, "/bin/");
        * if (os == OS_LINUX) strcat(tmp, LINUX_BIN_DIR);
        else strcat(tmp, SOLARIS_BIN_DIR);
        strcat(tmp, "/");
        strcat(tmp, source_files[i]);
        strcat(tmp, ".da");
        remove(tmp);

        strcpy(tmp, met_root);
        strcat(tmp, "/bin/");
        * if (os == OS_LINUX) strcat(tmp, LINUX_BIN_DIR);
        else strcat(tmp, SOLARIS_BIN_DIR);
        strcat(tmp, "/");
        strcat(tmp, GCOV_OUT_DIR);
        strcat(tmp, "/");
        strcat(tmp, source_files[i]);
        strcat(tmp, ".c.gcov");
        remove(tmp);
    }
    return EXIT_SUCCESS;
}

#!/usr/bin/tcsh
# Script to run a set of examples for checking the correctness
# of METFAC2's: measures, issued messages, and model generation.
#
# NOTE: this script calls all or some of three scripts (which ones depend
# on the input arguments ---see below) named run_tests_meas, run_tests_msg,
# and run_tests_modgen, which deal with, respectively, measures, messages,
# and model generation. In turn, each of such scripts calls a set of driver
# files, and it is from within each driver file that the METFAC2's .exe
# files are actually run with one or more sets of input parameters.
#
# Usage:
# run_tests (or: run_tests followed by 'msg', 'meas', and 'modgen'
# given in any order)
# runs all the available examples
# run_tests meas <list>
# runs the examples given in <list> for checking measures or all the
# examples if <list> is empty
# run_tests msg <list>
# runs the examples given in <list> for checking messages or all the
# examples if <list> is empty
# run_tests modgen <list>
# runs the examples given in <list> for checking the generation of
# the model or all the examples if <list> is empty
# run_tests followed by any combination of: 'meas' and a list of zero
# or more examples; 'msg' and a list of zero or more examples; and
# 'modgen' and a list of zero or more examples.
#
# NOTE: 'run_tests meas meas', run_tests 'msg msg', and so on are not
# valid inputs.
#
# Runs the following examples: for checking measures, those obtained
# by concatenating the corresponding lists (all the examples if
# the concatenation yielded an empty list); for checking messages,
# those obtained by concatenating the corresponding lists (all the
# examples if the concatenation yielded an empty list); for checking
# the generation of the model, those obtained by concatenating the
# corresponding lists (all the examples if the concatenation yielded
# an empty list).
#
# Output:
# file run_tests.log: log file (see variable LOG_FILE below)
# file run_tests.res: results file (see variable RES_FILE below)
# file run_tests.cov (see variable COV_FILE below): coverage data file
#
# Note: the script uses the 'comparestring' utility to compare strings.

set LOG_FILE=(run_tests.log)
set RES_FILE=(run_tests.res)
set COV_FILE=(run_tests.cov)
set MEAS_START=(meas)
set MSG_START=(msg)
set MODGEN_START=(modgen)
set DEFAULT_MEAS_LIST=('gen_model_list 0')
if ($status != 0) then
   echo "Unable to read the list of models"
   exit 1
endif
set DEFAULT_MSG_LIST=('gen_model_list 1')
if ($status != 0) then
   echo "Unable to read the list of models"
   exit 1
endif
set DEFAULT_MODGEN_LIST=('gen_model_list 2')
if ($status != 0) then
   echo "Unable to read the list of models"
   exit 1
endif

# make sure LOG_FILE, RES_FILE, and COV_FILE are all created anew
if (-e $LOG_FILE) \rm $LOG_FILE
if (-e $RES_FILE) \rm $RES_FILE
if (-e $COV_FILE) \rm $COV_FILE

# command-line parsing
set meas_list=('')
set msg_list=('')
set modgen_list=('')
@ it_meas_list = 0
@ it_msg_list = 0
@ it_modgen_list = 0
@ found_meas=0
@ found_msg=0
@ found_modgen=0
@ within_meas=0
@ within_msg=0
@ within_modgen=0
@ i=1

foreach val ($argv[1]*)

set meas_cmp=('comparestring $argv[$i] $MEAS_START')
set msg_cmp=('comparestring $argv[$i] $MSG_START')
set modgen_cmp=('comparestring $argv[$i] $MODGEN_START')

if ($meas_cmp) then
   # found "meas"
   if (!($within_meas)) then
      @ found_meas=1
      @ within_meas=1
      @ within_msg=0
      @ within_modgen=0
   else
      # found "meas meas"
      echo "Bad arguments: found 'meas' 'meas'"
      exit 1
   endif
else if ($msg_cmp) then
   # found "msg"
   if (!($within_msg)) then
      @ found_msg=1
      @ within_meas=0
      @ within_msg=1
      @ within_modgen=0
   else
      # found "msg msg"
      echo "Bad arguments: found 'msg' 'msg'"
      exit 1
   endif
else
else if ($modgen_cmp) then
    # found "modgen"  
    if (!$within_modgen) then
        @ found_modgen=1  
        @ within_meas=0  
        @ within_msg=0  
        @ within_modgen=1  
    else
        # found "modgen modgen"
        echo "Bad arguments: found 'modgen' 'modgen'"
        exit 1  
    endif
    endif
if (!$meas_cmp || $msg_cmp || $modgen_cmp) then
if ($within_meas) then
    # found anything after "meas"
    set meas_list=($meas_list $argv[$i])  
    @ it_meas_list++
else if ($within_msg) then
    # found anything after "msg"
    set msg_list=($msg_list $argv[$i])  
    @ it_msg_list++
else if ($within_modgen) then
    # found anything after "modgen"
    set modgen_list=($modgen_list $argv[$i])  
    @ it_modgen_list++
else
    # never found "meas" nor "msg" nor "modgen"
    echo "Bad arguments: expecting 'meas' or 'msg' or 'modgen'"
    exit 1  
    endif
    @ i++
end

# which script(s) to run and with which arguments
if (!$found_meas || $found_msg || $found_modgen) then
    # run all
    set do_meas=(1)
    set full_meas=(1)
    set meas_list=($DEFAULT_MEAS_LIST)
    set do_msg=(1)
    set full_msg=(1)
    set msg_list=($DEFAULT_MSG_LIST)
    set do_modgen=(1)
    set full_modgen=(1)
    set modgen_list=($DEFAULT_MODGEN_LIST)
else
    if ($found_meas) then
        set do_meas=(1)
        if (!$it_meas_list) then
            set full_meas=(0)
        else
            set full_meas=(1)
            set meas_list=($DEFAULT_MEAS_LIST)
        endif
    else
        set do_meas=(0)
    endif
    if ($found_msg) then
        set do_msg=(1)
        if (!$it_msg_list) then
            set full_msg=(0)
        else
            set full_msg=(1)
            set msg_list=($DEFAULT_MSG_LIST)
        endif
    else
        set do_msg=(0)
    endif
    if ($found_modgen) then
        set do_modgen=(1)
        if (!$it_modgen_list) then
            set full_modgen=(0)
        else
            set full_modgen=(1)
            set modgen_list=($DEFAULT_MODGEN_LIST)
        endif
    else
        set do_modgen=(0)
    endif
end
endif
else
    set do_msg=(0)
endif
if ($found_modgen) then
    set do_modgen=(1)
    if ($it_modgen_list) then
        set full_modgen=(0)
    else
        set full_modgen=(1)
        set modgen_list=($DEFAULT_MODGEN_LIST)
    endif
else
    set do_modgen=(0)
endif

# actual run
if ($do_meas) then
    echo "Invoking run_tests_meas"
    if (!($full meas)) then
        echo " Examples: "$meas_list
    else
        echo " Examples: all"
    endif
    echo "run_tests_meas" >>! $LOG_FILE
    echo "---------------------" >>! $LOG_FILE
    echo "" >>! $LOG_FILE
    run_tests_meas $LOG_FILE $meas_list
    if ($status != 0) then
        echo "run_tests_meas has failed."
        exit 1
    endif
endif
if ($do_msg) then
    echo "Invoking run_tests_msg"
    if (!($full_msg)) then
        echo " Examples: "$msg_list
    else
        echo " Examples: all"
    endif
    echo "run_tests_msg" >>! $LOG_FILE
    echo "---------------------" >>! $LOG_FILE
    echo "" >>! $LOG_FILE
    run_tests_msg $LOG_FILE $msg_list
    if ($status != 0) then
        echo "run_tests_msg has failed."
        exit 1
    endif
endif
if ($do_modgen) then
    echo "Invoking run_tests_modgen"
    if (!($full_modgen)) then
        echo " Examples: "$modgen_list
    else
        echo " Examples: all"
    endif
    echo "run_tests_modgen" >>! $LOG_FILE
    echo "---------------------" >>! $LOG_FILE
    echo "" >>! $LOG_FILE
    run_tests_modgen $LOG_FILE $modgen_list
    if ($status != 0) then
        echo "run_tests_modgen has failed."
        exit 1
    endif
endif
echo "run_tests: done"

grep 'SUMMARY' $LOG_FILE >>! $RES_FILE

get_cov >& $COV_FILE

exit 0

#
met_main: 1: 91.3.
met_io: 1: 92.56.
met_gen: 1: 90.17.
met_mcutil: 1: 98.67.
met_gmres: 1: 95.27.
met_gs: 1: 95.94.
met_opt: 1: 83.27.
met_s: 1: 97.82.
met_r: 1: 95.35.
met_lap: 1: 91.71.
met_rgd: 1: 94.41.
met_transf: 1: 94.45.
met_share: 1: 92.35.

SUMMARY (memory test: yes, CTMC test: yes): CCRD_1: 9/9 PASS, 0/9 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CCRD_2: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CCRD_3: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CCRD_4: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CCRD_5: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_1: 46/46 PASS, 0/46 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_10: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_11: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_12: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_13: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_2: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_3: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_4: 0/2 PASS, 2/2 FAIL => IT DOES NOT WORK AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_5: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_6: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_7: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CIAVD_8: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_1: 78/78 PASS, 0/78 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_2: 78/78 PASS, 0/78 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_3: 48/48 PASS, 0/48 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: no, CTMC test: yes): CRDDS_4: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_5: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_6: 0/6 PASS, 6/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_7: 0/3 PASS, 3/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDDS_8: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_1: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_2: 9/9 PASS, 0/9 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_3: 18/18 PASS, 0/18 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_4: 30/30 PASS, 0/30 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_5: 30/30 PASS, 0/30 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): CRDTE_6: 7/7 PASS, 0/7 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_1: 21/21 PASS, 0/21 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_10: 1/2 PASS, 1/2 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_11: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_12: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_13: 0/3 PASS, 3/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_2: 0/9 PASS, 9/9 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_3: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_4: 12/12 PASS, 0/12 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_5: 12/12 PASS, 0/12 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_6: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_7: 2/2 PASS, 0/2 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_8: 0/1 PASS, 1/1 FAIL => IT DOES NOT WORK AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): EARR_9: 0/2 PASS, 2/2 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRDS_1: 12/12 PASS, 0/12 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRDS_2: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRDS_3: 2/3 PASS, 1/3 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRDS_4: 36/36 PASS, 0/36 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_1: 7/13 PASS, 6/13 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_2: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_3: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_4: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_5: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ECRTE_6: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ESSRR_1: 15/15 PASS, 0/15 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): ESSRR_10: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.


SUMMARY (memory test: yes, CTMC test: yes): ESSRR_12: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_13: 0/1 PASS, 1/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_2: 11/11 PASS, 0/11 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_3: 6/6 PASS, 0/6 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_4: 0/6 PASS, 6/6 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_5: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_6: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_7: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_8: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ESSRR_9: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_1: 23/23 PASS, 0/23 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_10: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_11: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_12: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_13: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_14: 0/3 PASS, 3/3 FAIL => IT WORKS AS EXPECTED.


SUMMARY (memory test: yes, CTMC test: yes): ETRR_3: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_4: 12/12 PASS, 0/12 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_5: 7/9 PASS, 2/9 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_6: 8/8 PASS, 0/8 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_7: 1/1 PASS, 0/1 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_8: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): ETRR_9: 2/2 PASS, 0/2 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: no): msegerror_0: 15/15 PASS, 0/15 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_1: 7/7 PASS, 0/7 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_10: 2/2 PASS, 0/2 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_11: 2/2 PASS, 0/2 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_12: 5/5 PASS, 0/5 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_13: 3/3 PASS, 0/3 FAIL => IT WORKS AS EXPECTED.

SUMMARY (memory test: yes, CTMC test: yes): msegerror_14: 4/4 PASS, 0/4 FAIL => IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_15: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_16: 4/4 PASS, 0/4 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_17: 3/4 PASS, 1/4 FAIL ⇒ IT WORKS NOT AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_18: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_19: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_20: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_21: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_22: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_23: 1/1 PASS, 0/1 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_24: 1/1 PASS, 0/1 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_25: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_26: 1/1 PASS, 0/1 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_27: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_28: 3/5 PASS, 2/5 FAIL ⇒ IT WORKS NOT AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_29: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_3: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_30: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_31: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_32: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_33: 10/10 PASS, 0/10 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_34: 4/4 PASS, 0/4 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_35: 1/1 PASS, 0/1 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_36: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_37: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_38: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_39: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_4: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_40: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_41: 2/2 PASS, 0/2 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_42: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_43: 1/1 PASS, 0/1 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_44: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
SUMMARY (memory test: yes, CTMC test: yes): msgerror_45: 3/3 PASS, 0/3 FAIL ⇒ IT WORKS AS EXPECTED.
### Annex E

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_46: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_47: 4/4 PASS, 0/4 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_48: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_49: 5/5 PASS, 0/5 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_5: 7/7 PASS, 0/7 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_50: 3/3 PASS, 0/3 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_51: 4/4 PASS, 0/4 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_52: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_53: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_54: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_55: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_56: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_57: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_58: 7/7 PASS, 0/7 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_59: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_6: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_60: 4/4 PASS, 0/4 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_61: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_62: 4/5 PASS, 1/5 FAIL => IT

DOES NOT WORK AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_63: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_64: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_65: 0/1 PASS, 1/1 FAIL => IT

DOES NOT WORK AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_7: 1/1 PASS, 0/1 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_8: 3/3 PASS, 0/3 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): msgerror_9: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): modgen_1: 4/4 PASS, 0/4 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): modgen_2: 4/4 PASS, 0/4 FAIL => IT

WORKS AS EXPECTED.

#### SUMMARY (memory test: yes, CTMC test: yes): modgen_3: 2/2 PASS, 0/2 FAIL => IT

WORKS AS EXPECTED.

```
#!/usr/bin/tcsh

# Script to run the set of examples aimed at checking the correctness of
# the measures computed by METPAC.
#
# Usage:
#
#   run_tests_meas <logfile> -> runs all the examples (see variable
# DEFAULT EXAMPLE LIST below)
#
```
# run_tests_meas <logfile> <example_list> -> runs only the given examples
# NOTE: for each example, say mytest, the script expects to find the
# following files:
# mytest.spec -> METFAC specification file (There may or may not exist a
# mytest.c file.)
# mytest_maple -> file containing the Maple commands required to compute
# the exact solution of the example.
# driver_mytest.c -> driver program that actually runs the example.
# Output:
# file <logfile>
# NOTE: the script uses the custom utility 'timestamp' to obtain the
# timestamp of a file (082303: it could --should?-- be replaced by the
# system command 'date -r file'). Moreover, the name of the C compiler and
# the names and values of the flags to pass down to it to compile an
# example are those yielded by the environment variables METFAC2_CC
# and METFAC2_CC_FLAGS.
#
#******************************************************************************

set CC_COMPILER = ($METFAC2_CC)
set CC_FLAGS = ($METFAC2_CC_FLAGS)
set MAPLE_FLAGS = (-q -w0)
# list of examples is got from the file 'test_conf.modset'
set DEFAULT_EXAMPLE_LIST = ('gen_model_list 0')
if ($1 == "") then
  echo "Usage: run_tests_meas <logfile> [<examples_list>]"
  exit 1
else
  set LOG_FILE = ($argv[1])
  if ($2 == "") then
    set example_list = ($argv[2])
  else
    set example_list = ($DEFAULT_EXAMPLE_LIST)
  endif
  foreach example ($example_list)
    set example_spec = ($example".spec")
    set example_c = ($example".c")
    set example_exe = ($example".exe")
    set metfac_lib = ($METFAC2/bin/linux/libslvr.a)
    if (-e $example_spec) then
      set found1 = (1)
    else
      set found1 = (0)
    endif
    if (-e $example_c) then
      set found2 = (1)
    else
      set found2 = (0)
    endif
    if (-e $example_exe) then
      set found3 = (1)
    else
      set found3 = (0)
    endif
    if ($found1) then
      set t1 = ('timestamp $example_spec')
      if ($found2) then
        set t2 = ('timestamp $example_c')
      else
        set t2 = ('timestamp $example_exe')
      endif
      echo $t1 $t2 $example
    endif
  endforeach
endif
set t2 = ($t1)
endif
if ($found3) then
  set t3 = (`timestamp $example_exe`)
else
  set t3 = (0)
endif
set t4 = (`timestamp $metfac_lib`)
if (!$found3 || $t1 > $t3 || $t2 > $t3 || $t4 > $t3) then
  echo "Compiling model "$example" ..."
  echo "$example"
m2build $example
  echo ""
  echo "Done." if ($status != 0) then
    echo "Model: "$example": compilation failed"
    exit 1
endif
else
  echo "File "$example_spec" not found."
  exit 1
endif
set example_maple = ($example".maple")
set example_maple_out = ($example".log_maple")
if (-e $example_maple) then
  set found1 = (1)
else
  set found1 = (0)
endif
if (-e $example_maple_out) then
  set found2 = (1)
else
  set found2 = (0)
endif
if ($found1) then
  set t1 = (`timestamp $example_maple`)
  if ($found2) then
    set t2 = (`timestamp $example_maple_out`)
  else
    set t2 = (0)
  endif
  if (!$found2) || $t1 > $t2 then
    echo "Invoking Maple for example "$example".
    (Command file: "$example_maple") ..."
    maple "$MAPLE_FLAGS $example_maple >! $example_maple_out
    echo "Done."
  endif
else
  echo "File "$example_maple" not found."
  exit 1
endif
set example_driver_c = ("driver "$example".c")
set driver_lib_c = ("driver_lib.c")
set driver_lib_h = ("driver_lib.h")
set example_driver = ("driver "$example")
set driver_lib = ("driver_lib.o")
if (-e $example_driver_c) then
  set found1 = (1)
else
  set found1 = (0)
endif
if (-e $driver_lib_c) then
  set found2 = (1)
else
  echo "..."
set found2 = (0)
endif
if (-e $driver_lib_h) then
    set found3 = (1)
else
    set found3 = (0)
endif
if (-e $driver_lib) then
    set found4 = (1)
else
    set found4 = (0)
endif
if (-e $example_driver) then
    set found5 = (1)
else
    set found5 = (0)
endif
if ($found1 && $found2 && $found3) then
    set t2 = ('timestamp $driver_lib_c')
    set t3 = ('timestamp $driver_lib_h')
    if ($found4) then
        set t4 = ('timestamp $driver_lib')
    else
        set t4 = (0)
    endif
    if ($found5) then
        set t5 = ('timestamp $example_driver')
    else
        set t5 = (0)
    endif
    if (!($found4) || $t2 > $t4 || $t3 > $t4) then
        echo "Invoking the C compiler on the library file: " 
        "$driver_lib_c" ...
        $CC_COMPILER $CC_FLAGS -c $driver_lib_c
    endif
    if ($status != 0) then
        echo "$driver_lib_c": compilation failed"
        exit 1
    endif
    echo "Done."
    echo "Invoking the C compiler on the driver file: " 
    "$example_driver_c" ...
    $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o "$example_driver -lm"
    if ($status != 0) then
        echo "$example_driver_c": compilation failed"
        exit 1
    endif
    echo "Done."
else if (!($found5) || $t4 > $t5 || $t1 > $t5) then
    echo "Invoking the C compiler on the driver file: " 
    "$example_driver_c" ...
    $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o "$example_driver -lm"
    if ($status != 0) then
        echo "$example_driver_c": compilation failed"
        exit 1
    endif
    echo "Done."
endif
else
    if (!($found1)) echo "File "$example_driver_c" not found." 
    if (!($found2)) echo "File "$driver_lib_c" not found." 
    if (!($found3)) echo "File "$driver_lib_h" not found." 
    exit 1
endif
echo "Running "$example_driver" ...");
echo "Running "$example_driver" ..."); $LOG_FILE
echo "" $LOG_FILE
$example_driver $LOG_FILE
if ($status != 0) then
  echo "Failed."
  exit 1
endif
echo "Done."
echo ""
echo "$example_driver" $LOG_FILE
echo "$example_driver" $LOG_FILE
echo "$example_driver" $LOG_FILE
# rm $example_exe #not to exceed disk quota
echo "$example_driver" $LOG_FILE
end
exit 0
#
#!/usr/bin/tcsh

#############################################################################
#
# Script to run the set of examples aimed at checking the generation
# of the model
#
# Usage:
#
# run_tests_modgen <logfile> -> runs all the examples (see variable
# DEFAULT EXAMPLE LIST below)
#
# run_tests_modgen <logfile> <example_list> -> runs only the given examples
#
# NOTE: for each example, say mytest, the script expects to find the
# following files:
#
# mytest.spec -> METFAC specification file. (There may or may not exist a
# mytest.c file.)
#
# Output:
#
# file <logfile>
#
# NOTE: the script uses the custom utility 'timestamp' to obtain the
# timestamp of a file (082303: it could --should?-- be replaced by the
# system command 'date -r file'). Moreover, the name of the C compiler and
# the names and values of the flags to pass down to it to compile an
# example are those yielded by the environment variables METFAC2_CC
# and METFAC2_CC_FLAGS.
#
# ############################################################################

set CC_COMPILER = ($METFAC2_CC)
set CC_FLAGS = ($METFAC2_CC_FLAGS)
# list of examples is got from the file 'test_conf.modset'
set DEFAULT EXAMPLE LIST = (`gen_model_list 2``
if ($1 == "") then
  echo "Usage: run_tests_modgen <logfile> [<examples_list>]"
  exit 1
else
  set LOG_FILE = ($argv[1])
  if ($2 != "") then
    set example_list = ($argv[2]*)
  else
    example_list = ($DEFAULT EXAMPLE LIST)
  endif
  endif

foreach example ($example_list)
    set example_spec = ($example".spec")
    set example_c = ($example".c")
    set example_exe = ($example".exe")
    set metfac_lib = ($METFAC2/bin/linux/libslvr.a)
    if (-e $example_spec) then
        set found1 = (1)
    else
        set found1 = (0)
    endif
    if (-e $example_c) then
        set found2 = (1)
    else
        set found2 = (0)
    endif
    if (-e $example_exe) then
        set found3 = (1)
    else
        set found3 = (0)
    endif
    if ($found1) then
        set t1 = ('timestamp $example_spec')
    if ($found2) then
        set t2 = ('timestamp $example_c')
    else
        set t2 = ($t1)
    endif
    if ($found3) then
        set t3 = ('timestamp $example_exe')
    else
        set t3 = (0)
    endif
    set t4 = ('timestamp $metfac_lib')
    if (!(($found3 || $t1 > $t3 || $t2 > $t3 || $t4 > $t3)) then
        echo "Compiling model "$example" ..."
        echo ""m2build $example
        echo ""Done."
        if ($status != 0) then
            echo "Model: "$example": compilation failed"
            exit 1
        endif
    else
        echo "File "$example_spec" not found."
        exit 1
    endif
    set example_driver_c = ("driver."$example".c")
    set driver_lib_c = ("driver_lib.c")
    set driver_lib_h = ("driver_lib.h")
    set example_driver = ("driver."$example")
    set driver_lib = ("driver_lib.o")
    if (-e $example_driver_c) then
        set found1 = (1)
    else
        set found1 = (0)
    endif
    if (-e $driver_lib_c) then
        set found2 = (1)
    else
        set found2 = (0)
    endif
    if (-e $driver_lib_h) then
        set found3 = (1)
    else
        set found3 = (0)
    endif
set found3 = (0)
endif
if (!-e $driver_lib) then
    set found4 = (1)
else
    set found4 = (0)
endif
if (!-e $example_driver) then
    set found5 = (1)
else
    set found5 = (0)
endif
if ($found1 && $found2 && $found3) then
    set t1 = ('timestamp $example_driver_c')
    set t2 = ('timestamp $driver_lib_c')
    set t3 = ('timestamp $driver_lib_h')
    if ($found4) then
        set t4 = ('timestamp $driver_lib')
    else
        set t4 = (0)
    endif
    if ($found5) then
        set t5 = ('timestamp $example_driver')
    else
        set t5 = (0)
    endif
    if (!($found4 || $t2 > $t4 || $t3 > $t4)) then
        echo "Invoking the C compiler on the library file: "$driver_lib_c" ...
        $CC_COMPILER $CC_FLAGS -c $driver_lib_c
        if ($status != 0) then
            echo driver_lib_c": compilation failed"
            exit 1
        endif
        echo "Done."
        echo "Invoking the C compiler on the driver file: "$example_driver_c" ...
        $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o $example_driver -lm
        if ($status != 0) then
            echo $example_driver_c": compilation failed"
            exit 1
        endif
        echo "Done."
    else if (!($found5 || $t4 > $t5 || $t1 > $t5)) then
        echo "Invoking the C compiler on the driver file: "$example_driver_c" ...
        $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o $example_driver -lm
        if ($status != 0) then
            echo $example_driver_c": compilation failed"
            exit 1
        endif
        echo "Done."
    else if (!($found1)) echo "File $example_driver_c not found."
    if (!($found2)) echo "File $driver_lib_c not found."
    if (!($found3)) echo "File $driver_lib_h not found."
    exit 1
endif
else
    echo "Running "$example_driver" ...
    echo "Running "$example_driver" ...">>$LOG_FILE
    echo "" >> $LOG_FILE
    $example_driver >>k! $LOG_FILE
    if ($status != 0) then
        echo "Failed."
exit 1
endif
echo "Done."
echo ""
echo "" >>! $LOG_FILE
echo "Done." >>! $LOG_FILE
echo "" >>! $LOG_FILE
# \rm $example_exe #not to exceed disk quota
derm
exit 0
#
#!/usr/bin/tcsh

###########################################################################
# Script to run the set of examples aimed at checking the correctness of # the messages issued by METFAC.
# Usage:
#  run_tests_msg <logfile> -> runs all the examples (see variable DEFAULT_EXAMPLE_LIST below)
#  run_tests_msg <logfile> <example_list> -> runs only the given examples
#  NOTE: for each example, say mytest, the script expects to find the # following files:
#  mytest.spec     -> METFAC specification file. (There may or may not exist a # mytest.c file.)
#  driver_mytest.c -> driver program that actually runs the example.
#  mytest_expected_msg_<x> -> file(s) containing the expected messages # (there must be as many such files as times mytest.exe # is called from within driver_mytest.c; <x> is 1 for # the file that contains the messages that are expected # after mytest.exe is ran first, 2 for the file # containing the messages expected after the second # run of mytest.exe, and so on.)
# Output:
#  file <logfile>
#  NOTE: the script uses the custom utility 'timestamp' to obtain the # timestamp of a file (082303: it could --should?-- be replaced by the # system command 'date -r file'). Moreover, the name of the C compiler and # the names and values of the flags to pass down to it to compile an # example are those yielded by the environment variables METFAC2_CC # and METFAC2_CC_FLAGS.
#
###########################################################################

set CC_COMPILER = "$METFAC2_CC"
set CC_FLAGS = "$METFAC2_CC_FLAGS"
# list of examples is got from the file 'test_conf.modset' set DEFAULT_EXAMPLE_LIST = ('gen_model_list 1')
if ($! == "") then
   echo "Usage: run_tests_msg <logfile> [<examples_list>]"
else
   set LOG_FILE = ($argv[1])
   if ($2 !" "") then
      set example_list = ($argv[2]*)
   else
      example_list = ($DEFAULT_EXAMPLE_LIST)
   endif
foreach example ($example_list)
  set example_spec = ($example".spec")
  set example_c = ($example".c")
  set example_exe = ($example".exe")
  set metfac_lib = ($METFAO2/bin/linux/libslvr.a)
  if (-e $example_spec) then
    set found1 = (1)
  else
    set found1 = (0)
  endif
  if (-e $example_c) then
    set found2 = (1)
  else
    set found2 = (0)
  endif
  if (-e $example_exe) then
    set found3 = (1)
  else
    set found3 = (0)
  endif
  if ($found1) then
    set t1 = (`timestamp $example_spec`) 
    if ($found2) then
      set t2 = (`timestamp $example_c`) 
    else
      set t2 = ($t1)
    endif
    if ($found3) then
      set t3 = (`timestamp $example_exe`) 
    else
      set t3 = (0)
    endif
    set t4 = (`timestamp $metfac_lib`) 
    if (!($found3) || $t1 > $t3 || $t2 > $t3 || $t4 > $t3) then
      echo "Compiling model "$example" ..."
      echo "" m2build $example
      echo "" if ($status != 0) then
        echo "Model: "$example": compilation failed"
        exit 1
      endif
    else
      echo "File "$example_spec" not found."
      exit 1
    endif
  endif
set example_driver_c = ("driver "$example".c")
set driver_lib_c = ("driver_lib.c")
set driver_lib_h = ("driver_lib.h")
set example_driver = ("driver "$example")
set driver_lib = ("driver_lib.o")
if (-e $example_driver_c) then
  set found1 = (1)
else
  set found1 = (0)
endif
if (-e $driver_lib_c) then
  set found2 = (1)
else
  set found2 = (0)
endif
if (-e $driver_lib_h) then
  set found3 = (1)
else
  set found3 = (0)
endif
else
   set found3 = (0)
endif
if (-e $driver_lib) then
   set found4 = (1)
else
   set found4 = (0)
endif
if (-e $example_driver) then
   set found5 = (1)
else
   set found5 = (0)
endif
if ($found1 && $found2 && $found3) then
   set t1 = ('timestamp $example_driver_c')
   set t2 = ('timestamp $driver_lib_c')
   set t3 = ('timestamp $driver_lib_h')
   if ($found4) then
      set t4 = ('timestamp $driver_lib')
   else
      set t4 = (0)
   endif
   if ($found5) then
      set t5 = ('timestamp $example_driver')
   else
      set t5 = (0)
   endif
   if (!($found4) || $t2 > $t4 || $t3 > $t4) then
      echo "Invoking the C compiler on the library file: " \\
      "$driver_lib_c" ..."
      $CC_COMPILER $CC_FLAGS -c $driver_lib_c
      if ($status != 0) then
         echo driver_lib_c": compilation failed"
         exit 1
      endif
      echo "Done."
      echo "Invoking the C compiler on the driver file: " \\
      "$example_driver_c" ..."
      $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o \\
      $example_driver -lm
      if ($status != 0) then
         echo $example_driver_c": compilation failed"
         exit 1
      endif
      echo "Done."
   else
      if (!($found5) || $t4 > $t5 || $t1 > $t5) then
         echo "Invoking the C compiler on the driver file: " \\
         "$example_driver_c" ..."
         $CC_COMPILER $CC_FLAGS $example_driver_c $driver_lib -o \\
         $example_driver -lm
         if ($status != 0) then
            echo $example_driver_c": compilation failed"
            exit 1
         endif
         echo "Done."
      endif
   else
      if (!$found1) echo "$example_driver_c" not found."
      if (!$found2) echo "$driver_lib_c" not found."
      if (!$found3) echo "$driver_lib_h" not found."
      exit 1
   endif
   echo "Running "$example_driver" ..."
   echo "Running "$example_driver" ..." >> $LOG_FILE
   echo "" $LOG_FILE
   $example_driver $LOG_FILE
   if ($status != 0) then
```c
echo "Failed."
exit 1
endif
echo "Done."
echo ""
echo "" >>! $LOG_FILE
echo "Done." >>! $LOG_FILE
echo "" >>! $LOG_FILE
# \rm $example_exe # not to exceed disk quota
end
exit 0
#
/*
 * Common definitions for:
 *
 * gen_model_list.c
 *
 * get_cov.c
 *
 * init_cov.c
 *
 * timestamp.c
 *
 * comparestring.c
 *
 * driver_lib.c
 *
 * driver[]
 */
#if !defined(TEST_CONF_H)
define TEST_CONF_H

/* source files (see driver_lib.c for the actual contents of the variable) */
define MAX_SOURCE_FILES 32
extern const char *source_files[];
extern const int n_source_files;

/* constants */
define TRUE 1
#define FALSE 0
#define PASS_MSG "PASS"
#define FAIL_MSG "FAIL"
define AGREEMENT_MSG "IT WORKS AS EXPECTED"
define DISAGREEMENT_MSG "IT DOES NOT WORK AS EXPECTED"

/* OS's */
define OS_LINUX 1
#define OS_SOLARIS 2
#define OS_UNKNOWN 3

/* METFACT2's binaries are in the directory <metroot>/bin/LINUX_BIN_DIR */
* or <metroot>/bin/SOLARIS_BIN_DIR */
define LINUX_BIN_DIR "linux"
define SOLARIS_BIN_DIR "solaris"

/* Valgrind flags */
define VALG_INVOCATION "valgrind --leak-check=yes --show-reachable=yes --
logfile-fd="
define VALG_LOGFILE_FD "9"

/* Valgrind versions */
define VG_20030716 1
#define VG_1_9_6 2
#define VG_2_0_0 3
#define VG_UNKNOWN 4
```
#define VG_NOTFOUND 5

/* temporary file where Valgrind output is printed to */
#define VALGRIND_OUT "tmp.valgrind"

/* message starting with which Valgrind tells us how many errors it * has found */
#define VALGRIND_ERR_REPORT_MSG "ERROR SUMMARY:

/* message starting with which Valgrind tells us how much memory our program * has failed to free before returning */
#define VALGRIND_NONFREE_DMEM_MSG "malloc/free: in use at exit:

/* bcheck flags */
#define BCHECK_INVOCATION "bcheck -all -q -o 

/* bcheck error identifiers */
#define BCHECK_BAF "baf" /* bad free */
#define BCHECK_DUP "dup" /* duplicate free */
#define BCHECK_MAF "maf" /* misaligned free */
#define BCHECK_MAR "mar" /* misaligned read */
#define BCHECK_MAW "maw" /* misaligned write */
#define BCHECK_OOM "oom" /* out of memory */
#define BCHECK_RUA "rua" /* read from unallocated */
#define BCHECK_RUI "rui" /* read from uninitialized */
#define BCHECK_WRO "wro" /* write to read-only */
#define BCHECK_WUA "wua" /* write to unallocated */
#define BCHECK_AIB "aib" /* address in block */
#define BCHECK_AIR "air" /* address in register */
#define BCHECK_MEL "mel" /* memory leak */

/* gcov versions */
#define GCOV_303p 1
#define GCOV_1_5_2
#define GCOV_2_9_5 3
#define GCOV.UNKNOWN 4
#define GCOV_NOTFOUND 5

/* file where gcov output is printed to */
#define GCOV_OUT_FILE "cov_results"

/* the full path of the directory where the above file is kept is formed by * appending the value to which the macro below expands to the path of the * directory where the METAPC2 library (the libslvr.a file) lies */
#define GCOV_OUT_DIR "cov_analysis"

/* length of input/output lines (the variable where the line is to be stored * should have sizeof(LINENWIDTH+2 to make room for the newline and terminating * characters, and, likewise, the variable should be filled by calling the * fgets() function with LINENWIDTH+2 as its second argument) */
#define LINENWIDTH 2048

/* size of temporary char variables */
#define BUF_SZ 2048

/* when using methods without rigorous error control, the result will be * regarded as correct if the actual error is at most this value times * the user-given tolerance. */
#define TOLFACT 2.5

/* signal the beginning and end, respectively, of the section of the log file * of a driver where the vector is identified (name of the measure, used * method(s), etc) */
#define DRIVER_LOG_BEG IDENT "/" /* vector identity: start */
#define DRIVER_LOG_END IDENT "/" /* vector identity: end */

/* signal the beginning and end, respectively, of the section of the log file * of a driver where the result of the comparisons on the numerical results
/* are given */
#define DRIVER_LOG_BEG_NUMRES "/* numerical results test: start */"
#define DRIVER_LOG_END_NUMRES "/* numerical results test: end */"

/* signal the beginning and end, respectively, of the section of the log file
* of a driver where the output of the memory debugger has been inserted */
#define DRIVER_LOG_BEG_MEMRES "/* memory tests: start */"
#define DRIVER_LOG_END_MEMRES "/* memory tests: end */"

/* signal the beginning and end, respectively, of the section of the log file
* of a driver where the result of the comparisons on the error and warning
* error messages are given */
#define DRIVER_LOG_BEG_MSGRES "/* test of error and warning messages: start */"
#define DRIVER_LOG_END_MSGRES "/* test of error and warning messages: end */"

/* signal the beginning and end, respectively, of the section of the log file
* of a driver where the result of the test on the generation of the model is
* given */
#define DRIVER_LOG_BEG_MODGENRES "/* test of the model generation: start */"
#define DRIVER_LOG_END_MODGENRES "/* test of the model generation: end */"

/* signal the beginning and end, respectively, of the section of a driver log
* file where the result of the test on the CTMC is given */
#define DRIVER_LOG_BEG_CTMCGENRES "/* test of the CTMC: start */"
#define DRIVER_LOG_END_CTMCGENRES "/* test of the CTMC: end */"

/* message printed by METFAC2 to a model's log file signalling that the
* generation of the easy-to-parse description of the model has been checked
* and is ok */
#define RAW_MODGEN_OK_MSG "Test of the easy-to-parse model description: PASS"

/* message printed by METFAC2 to a model's log file signalling that the
* generation of the easy-to-parse description of the model has been checked
* and is _not_ ok */
#define RAW_MODGEN_BAD_MSG "Test of the easy-to-parse model description: FAIL"

/* message printed by METFAC2 to a model's log file signalling that the
* generation of the human-readable description of the model has been checked
* and is ok */
#define NICE_MODGEN_OK_MSG "Test of the human-readable model description: PASS"

/* message printed by METFAC2 to a model's log file signalling that the
* generation of the human-readable description of the model has been checked
* and is _not_ ok */
#define NICE_MODGEN_BAD_MSG "Test of the human-readable model description: FAIL"

/* message printed by METFAC2 to a model's log file signalling that the the
* CTMC has been checked and is ok */
#define CTCMGEN_OK_MSG "Test of the CTMC: PASS"

/* message printed by METFAC2 to a model's log file signalling that the the
* CTMC has been checked and is _not_ ok */
#define CTCMGEN_BAD_MSG "Test of the CTMC: FAIL"

/* for a model called THISMODEL, say, the corresponding exact numerical values
* of the measure are kept in a file whose name is formed by appending to
* THISMODEL the string to which the macro below expands */
#define EXACT_VAL_FILE_SUFFIX ".log_maple"

/* for a model called THISMODEL, say, the name of the C file of the corresponding
* driver is made up by prepending THISMODEL with the string to which the macro
* expands, and appending next ".c" to the result */
#define DRIVER_FILE_PREFIX "driver_"

/* for a model called THISMODEL, say, the name of the output file of the
* corresponding driver is made up by prepending THISMODEL with the string to
* which the macro DRIVER_FILE_PREFIX expands, and appending next the contents
* of the macro below to the result */
#define DRIVER_OUT_FILE_SUFFIX ".log"

/* temporary input file for model executable files */
#define MODEL_TMP_INP "tmp.inp"

/* for a model called THISMODEL, say, the expected warning and error messages */
* of the model are kept in a file whose name is formed by appending to
* THISMODEL the string below */
#define EXPECTED_MSG_FILE_SUFFIX ".expected_msg"

/* each expected error message is enclosed by the strings below */
#define EXP_ERR_MSG_BEGIN " /* begin error message */"
#define EXP_ERR_MSG_END "/* end error message */"

/* each expected warning message is enclosed by the strings below */
#define EXP_WARN_MSG_BEGIN " /* begin warning message */"
#define EXP_WARN_MSG_END "/* end warning message */"

/* in a model's log file, the first meaningful sentence of an error message */
* belongs to the line containing the string defined by ERR_MSG_HEADER_BEGIN,
* and the sentence itself begins right after the string defined by
* ERR_MSG_HEADER_END */

#define ERR_MSG_HEADER_BEGIN "[metfac2 error"
#define ERR_MSG_HEADER_END "]

/* in a model's log file, the first meaningful sentence of a warning message */
* belongs to the line containing the string defined by WAR_MSG_HEADER_BEGIN,
* and the sentence itself begins right after the string defined by
* WAR_MSG_HEADER_END */

#define WAR_MSG_HEADER_BEGIN "[metfac2 warning"
#define WAR_MSG_HEADER_END "]

/*/ model set stuff */

/* make sure the number MOD_MEAS expands to is smaller than the one yielded by */
* MOD_ERRWAR */
#define MOD_MEAS 1
#define MOD_MEAS_STR "MEAS"
#define MOD_ERRWAR 2

#define MOD_ERRWAR_STR "MSG"
#define MOD_MODGEN 3
#define MOD_MODGEN_STR "MODGEN"

/* name of the file that contains the set of models (see read_modset() in */
* driver_lib.c for a detailed description of the structure of the file) */
#define MOD_LIST_FILE "test_conf.modset"

#endif /* TEST_CONF_H */

/* Utilities. */

/* System headers. */
#include <float.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>

/* Custom headers. */
#include "driver_lib.h"

/* Data structures. */

/* to store pairs (name, integer) */
struct meas_code {  
  char name[BUF_SZ];  
  int code;  
};
typedef struct meas_code meas_code_t;

/* message */
struct msg_  
{  
  int n_sent;  /* no. of non-empty sentences making up the  
    * message */  
  char **sents;  /* sents[i], 0<i<n_sent: pointer to the i'th  
    * sentence (note: each sentence has room for  
    * MAX_TOKEN_SIZE+2 characters) */  
  int sents_sz,  /* current size of the sents array */  
    idx,  /* message index: l for the message read first, 2  
    * for the message read in the second place, etc */  
    type,  /* EXP_MSG for expected messages and READ_MSG  
    * otherwise */  
  flag;  
};
typedef struct msg_ msg_t;

/* set of messages */
struct msg_set_  
{  
  int n_msg;  /* no. of messages */  
  struct msg_ **msgs;  /* msgs[i], 0<i<n_msg: pointer to the i'th  
    * message */  
  int msgs_sz;  /* current size of the msgs array */  
};

/* model info */
struct modinf_  
{  
  char name[BUF_SZ];  /* model name */  
  int type,  /* either MOD_MEAS (the model is for checking the  
    * computation of measures) or MOD_ERROR (the model  
    * is for checking error and warning messages  
    * issued) */  
  int n_exp_vect,  /* no. of expected vectors, i.e. no. of vectors that  
    * the corresponding driver should generate */  
  n_exp_vect_pass,  /* no. of expected vectors that are supposed to pass  
    * the corresponding test */  
  n_exp_vect_fail;  /* no. of expected vectors that are supposed _not_ to  
    * pass the test (because of, e.g., the inability to  
    * accurately control the error incurred in computing  
    * some measures) */  
};
typedef struct modinf_ modinf_t;

/* model set */
struct modset_  
{  
  int n_mod;  /* no. of models */  
  struct modinf_ **mods,  /* mods[i], 0<i<n_mod: i'th model info */  
    **mods_by_nm,  /* pointers to cells of mods sorted in increasing  
    * order of the field 'name' */  
  **mods_by_tp_nm;  /* pointers to cells of mods sorted in increasing
* order of first the field 'type' and next the field 'name' */
int mods_sz; /* current size of the mods array */
}

typedef struct modset_ modset_t;

/* Macro definition. */
#define VALG_VERS_TMP_FILE "valgrind_version.tmp"
#define GCOV_VERS_TMP_FILE "gcov_version.tmp"
#define DEF_MSG_SZ 8
#define DEF_SENT_SZ 8
#define READ_MSG 1
#define EXP_MSG 2
#define DEF_N_MOD 64

/* File-scope variables. */

/* pointer to the set of models */
static
modset_t
*msetp_g = NULL;

/* Program-scope variables. */

/* list and no. of METFAC source files */
const
char *
source_files[] = {
    "met_main",
    "met_meas",
    "met_ic",
    "met_gen",
    "met_mcutil",
    "met_gmres",
    "met_gsl",
    "met_optor",
    "met_sr",
    "met_ii",
    "met_lap",
    "met_rqd",
    "met_transf",
    "met_share"};

const
int
n_source_files = sizeof(source_files)/sizeof(char*);

/* Functions. */

/*
 * Returns the operating system version, which can be any of: OS_LINUX
 * (Linux), OS_SOLARIS (Solaris), and OS_UNKNOWN (not known).
 * NOTE: the OS type is read from the env. var. "METFAC2_OS".
 */

int os_type(void)
{
    char *os;
    os = getenv("METFAC2_OS");
if (os && strstr(os, "linux")) return OS_LINUX;
else if (os && strstr(os, "solaris")) return OS_SOLARIS;
else return OS_UNKNOWN;
}

/*
* Returns the version of the Valgrind program, which can be any of:
* VG_20030716 (version 20030716), VG_1_9_6 (version 1.9.6), VG_2_0_0 (version
* 2.0.0), VG UNKNOWN (version not known), and VG_NOTFOUND (Valgrind is not
* available).
*/

int
valgrind_version(void)
{
char command[BUFSZ], vers[LINENWWTH+2];
int ret;
FILE *fp;

switch (os_type()) {
    case OS_UNKNOWN:
        case OS_SOLARIS: /* Valgrind doesn't (and won't) run on Solaris */
            return VG_NOTFOUND;
        case OS_LINUX:
            strcpy(command, "valgrind --version > ");
            break;
}
strcat(command, VALG_VERS_TMP_FILE);
system(command);
if (!fp = fopen(VALG_VERS_TMP_FILE, "r")) {
    (void) fprintf(stderr, "valgrind_version(): unable to open 
    "file '%s' file.
    ");
    exit(EXIT_FAILURE);
}
fgets(vers, LINENWWTH+2, fp);
if (strstr(vers, "not found")) ret = VG_NOTFOUND;
else if (strstr(vers, "20030716")) ret = VG_20030716;
else if (strstr(vers, "1.9.6")) ret = VG_1_9_6;
else if (strstr(vers, "2.0.0")) ret = VG_2_0_0;
else ret = VG_UNKNOWN;
if (fclose(fp) != 0) {
    (void) fprintf(stderr, "valgrind_version(): unable to close "
    "file '%s'.n",
    VALG_VERS_TMP_FILE);
    exit(EXIT_FAILURE);
}
remove(VALG_VERS_TMP_FILE);
return ret;
}

/*
* Returns the version of the gcov program, which can be any of: GCOV_303p
* (version 303p), GCOV_1_5 (version 1.5), GCOV_2_9_5 (version 2.9.5),
* GCOV UNKNOWN (version not known), and GCOV_NOTFOUND (gcov is not
* available).
*/

int
gcov_version(void)
{
char command[BUFSZ], vers[LINENWWTH+2];
int ret;
FILE *fp;

switch(os_type()) {
    case OS_UNKNOWN:
        return GCOV_NOTFOUND;
case OS_LINUX:
    strncpy(command, "gcov -v ");
    break;
case OS_SOLARIS:
    /* strncpy(command, "/usr/bin/tcsh gcov_version_solaris "); */
    strncpy(command, "gcov -v 2> ");
    break;
}
strcat(command, GCOV_VERSION_TMP_FILE);
system(command);
if (!fp = fopen(GCOV_VERSION_TMP_FILE, "r")) {
    (void) fprintf(stderr, "gcov_version(): unable to open 
    "file 's' file:\n", GCOV_VERSION_TMP_FILE);
    exit(EXIT_FAILURE);
}
fgets(vers, LINEWIDTH+2, fp);
if (strstr(vers, "not found")) ret = GCOV_NOTFOUND;
else if (strstr(vers, "303p")) ret = GCOV_303p;
else if (strstr(vers, "1.5")) ret = GCOV_1_5;
else if (strstr(vers, "2.9.5")) ret = GCOV_2_9_5;
else ret = GCOV_UNKNOWN;
if (fclose(fp) != 0) {
    (void) fprintf(stderr, "gcov_version(): unable to close 
    "file 's' file:\n", GCOV_VERSION_TMP_FILE);
    exit(EXIT_FAILURE);
}
remove(GCOV_VERSION_TMP_FILE);
return ret;
}

/*
 * Builds up in the string comm the command to call the executable file that
 * results from compiling a model. modnam holds the model name, vectinp holds
 * the file the executable will read parameters from, and vectout holds the
 * file where the executable will print its messages, if any. If the
 * environment variable M2TEST_MEMDBG is defined and set to anything different
 * from the empty string, the executable file will be called through a memory
 * debugger: Valgrind if the operating system is Linux and bcheck if the OS is
 * Solaris. The debugger's report will be printed to the file * dbginfo. If
 * M2TEST_MEMDBG is undefined or has been set to the empty string, no memory
 * debugger will be used and the value of *dbginfo is irrelevant.
 */
void
build_met2comm(char *comm, const char *modnam, const char *vectinp,
    const char *vectout, const char *dbginfo)
{
    static int os, memdbg, firstcall = TRUE;
    static char command[BUF_SZ];

    if (firstcall) {
        char *s;
        int vg Vers;

        firstcall = FALSE;
        os = os_type();
        s = getenv("M2TEST_MEMDBG");
        if (s && *s) memdbg = TRUE;
        else memdbg = FALSE;
        switch (os) {
        case OS_LINUX:
            switch (memdbg) {
                case TRUE:
                    vg Vers = valgrind_version();
                    if (vg Vers == VG_NOTFOUND) {
                        /* force error */
                        strcpy(command, "valgrind_not_available ");
                    }  
                    break;
                case FALSE:
                case OS_SOLARIS:
                    break;
            }
        }
        break;
        case OS_SOLARIS:
            break;
        }
    }
    printf("%s\n", command);
    if (firstcall) {
        return;
    }
    (void) execv(command, (char **) &comm);
    return;
}
} else if (vg_ver == VG_UNKNOWN) {
    /* force error */
    strcpy(command, "valgrind_ver_not_known");
} else {
    strcpy(command, VALG_INVOCATION);
    strcat(command, VALG_LOGFILE_FD);
    strcat(command, ":");
    strcat(command, modnam);
    strcat(command, ".exe");
    strcat(command, vectinp);
    strcat(command, ">");
    strcat(command, vectout);
    strcat(command, "->");
    strcat(command, VALG_LOGFILE_FD);
    strcat(command, ">");
    strcat(command, dbginfo);
}
break;
case FALSE:
    strcpy(command, modnam);
    strcat(command, ".exe");
    strcat(command, vectinp);
    strcat(command, ">");
    strcat(command, vectout);
    break;
}
break;
case OS_SOLARIS:
    switch (memdbg) {
    case TRUE:
        strcpy(command, CHECK_INVOCATION);
        strcat(command, dbginfo);
        strcat(command, ":");
        strcat(command, modnam);
        strcat(command, ".exe");
        strcat(command, vectinp);
        if (!strstr(vectout, "/dev/null")) {
            strcat(command, "->");
        } else strcat(command, ">");
        strcat(command, vectout);
        break;
    case FALSE:
        strcpy(command, modnam);
        strcat(command, ".exe");
        strcat(command, vectinp);
        if (!strstr(vectout, "/dev/null")) {
            strcat(command, "->");
        } else strcat(command, ">");
        strcat(command, vectout);
        break;
    }
break;
case OS_UNKNOWN:
    /* force error */
    strcpy(command, "do_not_know_which_OS_you_work_with");
    break;
}
strcpy(comm, command);
return;

/*
 * Given a model called *modelname, builds up in *name the name of the exec
 * file of the corresponding driver. Such a name consists of the string to
 * which the macro DRIVER_FILE_PREFIX expands followed by the contents of
/* modname.

NOTE: the macro DRIVER_FILE_PREFIX is defined in test_conf.h.
*/

void build_driver_filename(const char *modname, char *fname)
{
    strncpy(fname, DRIVER_FILE_PREFIX);
    strcat(fname, modname);
    return;
}

/*
* Given a model called *modname, builds up in *fname the name of the log file
* of the corresponding driver. Such a name consists of the string to which
* the macro DRIVER_FILE_PREFIX expands followed by the contents of *modname
* followed by the string to which the macro DRIVER_OUT_FILE_SUFFIX expands.
* NOTE: the macros DRIVER_FILE_PREFIX and DRIVER_OUT_FILE_SUFFIX are defined
* in test_conf.h.
*/

static void build_driver_logfilename(const char *modname, char *fname)
{
    strncpy(fname, DRIVER_FILE_PREFIX);
    strcat(fname, modname);
    strcat(fname, DRIVER_OUT_FILE_SUFFIX);
    return;
}

/*
* Stores in par_set[] and prints to file *fp the set of parameters given next
* to the 'format' argument. Those parameters are given "a la" printf(). Thus,
* for a model having two parameters named SIZE and LAMBDA, the first one of
* type integer and the second of type double, the call to this function might
* look as follows:
*    * build_parset(filepointer, stringpointer, "%d\n%g\n", valueofSIZE,
*    *     valueofLAMBDA);
*/

void build_parset(FILE *fp, char parset[], const char *format, ...)
{
    va_list argp;

    va_start(argp, format);
    (void) vsprintf(parset, format, argp);
    va_end(argp);
    (void) fprintf(fp, "%s", parset);
    return;
}

/*
* Given a model's name *modname and an integer n_val>0, opens the file whose
* name consists of the string to which the macro EXACT_VAL_FILE_SUFFIX
* expands appended to *modname, and reads from it n_val different numerical
* values. Upon success, val[1], ..., val[n_val] hold the read values and 0 is
* returned; upon failure, the function returns an integer <>0 and the
* contents of val[] is undefined.
* NOTES: the macro EXACT_VAL_FILE_SUFFIX is defined in test_conf.h; in the
* file, there must be a single value per line (empty lines are allowed,
* though).
*/

int
read_exact_values_meas(const char *modname, int n_val, double val[])
{
    char tmp[BUF_SZ], line[LINEWIDTH+2];
    FILE *fp;
    int i, notend, noteof, ret;

    strcpy(tmp, modname);
    strcat(tmp, EXACT_VAL_FILE_SUFFIX);
    if (!(fp = fopen(tmp, "r"))) return 1;
    i = 0;
    notend = TRUE;
    noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    while (noteof && notend) {
        ret = sscanf(line, "%lf", &val[i++]);
        if (ret != EOF && ret > 0 && ++i == n_val) notend = FALSE;
        else noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    }
    if (fclose(fp) != 0 || notend) return 1;
    else return 0;
}

FILE *
open_driver_logfile(const char *modname)
{
    char fname[BUF_SZ];

    build_driver_logfilename(modname, fname);
    return (fopen(fname, "w"));
}

/*
 * Comparison function to sort an array of type meas_code_t in ascending order
 * of the 'name' field.
 */

static
int
cmp_meas_code(const void *a, const void *b)
{
    return (strcmp(((meas_code_t *)a)->name, ((meas_code_t *)b)->name));
}

/*
 * Given the name of a measure in *measname, sets *cd to the numerical code
 * the measure has in METFAC2 and returns the integer 0; if *measname is not a
 * valid measure name, returns an integer <>0 and the contents of *cd is
 * undefined.
 */

int
meas_num_code(const char *measname, int *cd)
{
    static meas_code_t
namecode[] = {
    {"ETRR", 1},
    {"ESSRR", 2},
    {"EARR", 3},
    {"CRCED", 4},
    {"IACED", 5},
    {"ECRTE", 6},
    {"CRDTE", 7},
    {"ECRDS", 8},
    {"CRDDS", 9}
};
static int n_namecode = sizeof(namecode)/sizeof(meas_code_t),
    sorted = 0;
meas_code_t key, *elem;
if (!sorted) {
    qsort(namecode, n_namecode, sizeof(meas_code_t), cmp_meas_code);
    sorted = 1;
}
(void) strcpy(key.name, measname);
if (!elem =
    bsearch(&key, namecode, n_namecode, sizeof(meas_code_t),
            cmp_meas_code)) return 1;
*cd = elem->code;
return 0;

/*
   * Given a model's name *modname, a measure's name *measname, a string *eqstr,
   * and an integer n_val>0, opens the log file that resulted from running the
   * model and reads from it n_val numerical values of the measure. The function
   * assumes that in the log file the values of the measure are given one per
   * line in the form:
   *   *measname *eqstr <value>
   * and, therefore, typically the function will be called with *eqstr equal to
   * " = ", " <= ", or " >= ". If all the values are read successfully, the
   * function returns 0 and the read values in val[1],
   * ... val[n_val]. Otherwise, the function returns an integer <>0 and the
   * contents of val[] is undefined.
   */
int
read_met2_val meas(const char *modname, const char *measname,
                   const char *eqstr, int n_val, double val[])
{
    char tmp[BUF_SZ], line[LINENOTH+2], *ptr;
    FILE *fp;
    int i, notend, noteof, len;
    strcpy(tmp, modname);
    strcat(tmp, ".log");
    if (!((fp = fopen(tmp, "r"))) return 1;
    len = strlen(measname)+strlen(eqstr);
    i = 0;
    notend = TRUE;
    noteof = (fgets(line, LINENOTH+2, fp) == NULL? FALSE: TRUE);
    while (noteof && notend) {
        if ((ptr = strstr(line, measname))) {
            sscanf(ptr+len, ">%f", &val[+i]);
        }
        if (i == n_val) notend = FALSE;
        else noteof = (fgets(line, LINENOTH+2, fp) == NULL? FALSE: TRUE);
    }
    if (fclose(fp) ! 0 | notend) return 1;
    else return 0;
}
int read_met2_bounds_meas(const char *modname, const char *measname, const char *absname, int n_p, double lb_val[], double ub_val[])
{
    char str[BUF_S2], line[LINEWIDTH+2], *ptr;
    FILE *fp;
    int i, notend, noteof;
    double tmp;

    strcpy(str, modname);
    strcat(str, "\_log");
    if (! (fp = fopen(str, "r"))) return 1;

    strcpy(str, absname);
    strcat(str, " = \lf \lf <= ");
    strcat(str, measname);
    strcat(str, " <= \lf");
    i = 0;
    notend = TRUE;
    noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    while (noteof && notend) {
        if ((ptr = strstr(line, measname))) {
            sscanf(line, str, &tmp, &lb_val[i+1], &ub_val[i+1]);
            ++i;
        } else if (i == n_p) notend = FALSE;
        else noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    }
    if (fclose(fp) != 0 || notend) return 1;
    else return 0;
}

/*  
* Prints to the file pointed to by *fp the identity of the vector defined by  
* the model's name *modname and the set of parameters *par. Useful for tasks  
* where no numerical method is chosen (i.e. generation of the model).  
*/
void print_ident_vector_zero_meth(const char *modname, const char *par, FILE *fp)
{
    (void) fprintf(fp, "%s\\n", DRIVER_LOG_BEG_IDENT);
    (void) fprintf(fp, "Model: %s\\n", modname);
    (void) fprintf(fp, "Parameters:\n\n", par);
    (void) fprintf(fp, "%s\\n", DRIVER_LOG_END_IDENT);
    return;
}
/ * Prints to the file pointed to by *fp the identity of the vector defined by * the model's name *modname and the set of parameters *par. *measname is the * name of the measure, meth is the code of the numerical method, and max_err * is the maximum error/tolerance. Useful for measures that use one numerical * method. */

void print_ident_vector_one_meth(const char *modname, const char *par,
const char *measname, int meth,
double max_err, FILE *fp)
{
(void) fprintf(fp, "%s\n", DRIVER_LOG_BEG_IDENT);
(void) fprintf(fp, "Model: %s.\n", modname);
(void) fprintf(fp, "Measure: %s.\n", measname);
(void) fprintf(fp, "Parameters:\n\n", par);
(void) fprintf(fp, "Method: %d (err/tol: %g).\n", meth, max_err);
(void) fprintf(fp, "%s\n\n", DRIVER_LOG_END_IDENT);
return;
}

/*
* Does the same as print_ident_vector_one_meth() for measures that use two
* numerical methods.
*/

void print_ident_vector_two_meth(const char *modname, const char *par,
const char *measname, int meth1, int meth2,
double max_err1, double max_err2, FILE *fp)
{
(void) fprintf(fp, "%s\n", DRIVER_LOG_BEG_IDENT);
(void) fprintf(fp, "Model: %s.\n", modname);
(void) fprintf(fp, "Measure: %s.\n", measname);
(void) fprintf(fp, "Parameters:\n\n", par);
(void) fprintf(fp, "Methods: %d (err/tol: %g), %d (err/tol: %g).\n", meth1, max_err1, meth2, max_err2);
(void) fprintf(fp, "%s\n\n", DRIVER_LOG_END_IDENT);
return;
}

/*
* Does the same as print_ident_vector_one_meth() for measures that use three
* numerical methods.
*/

void print_ident_vector_three_meth(const char *modname, const char *par,
const char *measname, int meth1, int meth2,
int meth3, double max_err1, double max_err2,
double max_err3, FILE *fp)
{
(void) fprintf(fp, "%s\n", DRIVER_LOG_BEG_IDENT);
(void) fprintf(fp, "Model: %s.\n", modname);
(void) fprintf(fp, "Measure: %s.\n", measname);
(void) fprintf(fp, "Parameters:\n\n", par);
(void) fprintf(fp, "Methods: %d (err/tol: %g), %d (err/tol: %g), %d (err/tol: %g).\n", meth1, max_err1, meth2, max_err2, meth3, max_err3);
(void) fprintf(fp, "%s\n\n", DRIVER_LOG_END_IDENT);
return;
/*
 * Determines whether x1 and x2 are approximately equal to a relative accuracy
 * dbl epsilon.
 * 
 * SOURCE: fcmp() function of the Gnu Scientific Library (GSL).
 */

static
int


dblcmp(const double x1, const double x2, const double epsilon)
{
    int exponent;
    double max, delta, difference;

    /* Find exponent of largest absolute value */
    max = (fabs(x1) > fabs(x2)) ? x1 : x2;
    frexp(max, &exponent);

    /* Form a neighborhood of size 2*delta */
    delta = ldexp(epsilon, exponent);
    difference = x1-x2;

    if (difference > delta) { /* x1 > x2 */
        return 1;
    } else if (difference < -delta) { /* x1 < x2 */
        return -1;
    } else { /* -delta <= difference <= delta */
        return 0;
    }
}

/*
 * Yields 1 if x1 <= x2 and 0 otherwise assuming the last 5 bits of both x1
 * and x2 are contaminated by round-off error.
 *
 * NOTE: if e denotes the number of bits of error, the factor multiplying
 * DBL_EPSILON should be 2^e-1. (See doc of the Theodore C. Belding's fcmp
 * package.)
 */

#define NOTLARGER(x1,x2) (dblcmp((x1),(x2),31.0*DBL_EPSILON)==1?0:1)

/*
 * Yields 1 if x1 >= x2 and 0 otherwise assuming the last 5 bits of both x1
 * and x2 are contaminated by round-off error.
 *
 * NOTE: if e denotes the number of bits of error, the factor multiplying
 * DBL_EPSILON should be 2^e-1. (See doc of Theodore C. Belding fcmp package.)
 */

#define NOTSMALLER(x1,x2) (dblcmp((x1),(x2),31.0*DBL_EPSILON)==-1?0:1)

/*
 * Yields 1 if x1 = x2 and 0 otherwise assuming the last 5 bits of both x1
 * and x2 are contaminated by round-off error.
 *
 * NOTE: if e denotes the number of bits of error, the factor multiplying
 * DBL_EPSILON should be 2^e-1. (See doc of the Theodore C. Belding fcmp
 * package.)
 */

#define EQUAL(x1,x2) (dblcmp((x1),(x2),31.0*DBL_EPSILON)==0?1:0)

/*
 * Given in meth the numerical method with which the measure ETRR has been
* computed (see below), in exact_val and met2_val the exact and computed
* values, respectively, in max_err the maximum error/tolerance with which the
* computation has been performed, and in fp a pointer to the log file of the
* driver, checks whether the computed value is ok, printing to the file the
* appropriate message. Returns 0 if meth is a valid method and an integer <>0
* otherwise.
* 
* NOTE: use check_result_ETRR_b() for Bounding Regenerative Randomization.
*/

int
check_result_ETRR(int meth, double max_err, double exact_val, double met2_val,
                   FILE *fp)
{
    int ret;
    double allow_err, actual_err;

    ret = 0;
    switch (meth) {
        case 1: /* SR_ABS */
        case 4: /* RR */
            /* we control exact_val-met2_val rigorously; moreover, met2_val
             * should be not larger than exact_val */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                           met2_val);
            allow_err = max_err;
            actual_err = exact_val-met2_val;
            (void) fprintf(fp, "abs. err. = %g, ", actual_err);
            if (NOTLARGER(met2_val, exact_val) & NOTLARGER(actual_err,
                         allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
        case 2: /* SSD */
        case 7: /* RQD */
            /* we control |exact_val-met2_val| rigorously */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                           met2_val);
            allow_err = max_err;
            actual_err = fabs(exact_val-met2_val);
            (void) fprintf(fp, "|abs. err.| = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
        case 3: /* SR_REL */
            /* we control |(exact_val-met2_val)|/met2_val rigorously */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                           met2_val);
            allow_err = max_err;
            if (met2_val != 0.0) {
                actual_err = fabs((exact_val-met2_val)/met2_val);
            } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
            else actual_err = DBL_MAX;
            (void) fprintf(fp, "|rel. err.| = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
        case 5: /* RR_LT */
            /* we control exact_val-met2_val approximately */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
met2_val;
allow_err = TOLFACT*max_err;
actual_err = fabs(exact_val-met2_val);
(void) fprintf(fp, "|abs. err.| = %g , actual_err);
if (NOTLARGER(actual_err, allow_err)) {
(void) fprintf(fp, "%s\n", PASS_MSG);
} else {
(void) fprintf(fp, "%s\n", FAIL_MSG);
}
break;
default: /* either RR_B or unknown method */
ret = 1;
break;
} /* switch (meth) */
return ret;

/*
* Given in exact_val the exact value of the measure ETRR, in met2_bound the
* computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum
* error with which the computation has been performed, and in fp a pointer to
* the log file of the driver, checks whether the bound is ok, printing to the
* file the appropriate message. Returns 0 if lb is either TRUE or FALSE and
* an integer <>0 otherwise.
*/

int
check_result_ETRR_b(int lb, double max_err, double exact_val,
                   double met2_bound, FILE *fp)
{
int ret;
ret = 0;
switch (lb) {
  case TRUE:
    (void) fprintf(fp, "exact = %g, lower bound = %g ,",
                   exact_val, met2_bound);
    /* neglecting round-off errors, the actual bound should not be
     * smaller than met2_bound */
    if (NOTLARGER(met2_bound, exact_val)) {
      (void) fprintf(fp, "%s\n", PASS_MSG);
    } else (void) fprintf(fp, "%s\n", FAIL_MSG);
    break;
  case FALSE:
    (void) fprintf(fp, "exact = %g, upper bound = %g ",
                   exact_val, met2_bound);
    /* neglecting round-off errors, the actual bound should not be
     * larger than met2_bound+max_err */
    if (NOTSMALLER(met2_bound+max_err, exact_val)) {
      (void) fprintf(fp, "%s\n", PASS_MSG);
    } else (void) fprintf(fp, "%s\n", FAIL_MSG);
    break;
  default:
    ret = 1;
    break;
  }
return ret;
}

/*
* Given in meth1 and meth2 the numerical methods with which the measure ESSRR
* has been computed (see below), in exact_val and met2_val the exact and
* computed values, respectively, in max_err1 and max_err2 the maximum
* error/tolerance for meth1 and meth2, respectively, with which the
* computation has been performed, and in fp a pointer to the log file of the
* driver, checks whether the computed value is ok, printing to the file the
* appropriate message. Returns 0 if meth1 and meth2 are valid methods and an
* integer <>0 otherwise.
*/

int check_result_ESSRR(int meth1, int meth2, double max_err1, double max_err2,
                      double exact_val, double met2_val, FILE *fp)
{
    int ret;
    double allow_err, actual_err;

    if (!(meth2 == 1 || meth2 == 2 || meth2 == 3)) return 1;
    ret = 0;
    switch (meth1) {
        case 1: /* GS */
        case 2: /* BGS */
        case 3: /* SOR */
        case 4: /* GMRES */
            /* we control |exact_val-met2_val|/met2_val approximately */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                          met2_val);
            if (max_err1 > max_err2) allow_err = TOLFACT*max_err1;
            else allow_err = TOLFACT*max_err2;
            if (met2_val != 0.0) {
                actual_err = fabs((exact_val-met2_val)/met2_val);
            } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
            else actual_err = DBL_MAX;
            (void) fprintf(fp, "|rel. err.| = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
        case 5: /* SRD */
            /* we control exact_val-met2_val rigorously */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                          met2_val);
            allow_err = max_err1;
            actual_err = exact_val-met2_val;
            (void) fprintf(fp, "abs. err. = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
        default: /* unknown */
            ret = 1;
            break;
    } /* switch (meth1) */
    return ret;
}

/*
* Given in meth the numerical method with which the measure EARR has been
* computed, in exact_val and met2_val the exact and computed values,
* respectively, in max_err the maximum error/tolerance with which the
* computation has been performed, and in fp a pointer to the log file of the
* driver, checks whether the computed value is ok, printing to the file the
* appropriate message. Returns 0 if meth is a valid method and an integer <>0
* otherwise.
*/

int
check_result_ERR(int meth, double max_err, double exact_val, double met2_val, 
    FILE *fp)
{
    int ret;
    double allow_err, actual_err;

    ret = 0;
    switch (meth) {
    case 1: /* SR_ABS */
        /* we control exact_val-met2_val rigorously; moreover, met2_val 
         * should be not larger than exact_val */
        (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, 
                  met2_val);
        allow_err = max_err;
        actual_err = exact_val-met2_val;
        if (NOTLARGER(met2_val, exact_val) && NOTLARGER(actual_err, 
                  allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    case 2: /* SSD */
    case 6: /* RQQ */
        /* we control |exact_val-met2_val| rigorously */
        (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, 
                  met2_val);
        allow_err = max_err;
        actual_err = fabs(exact_val-met2_val);
        if (NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    case 3: /* SR_REL */
        /* we control |(exact_val-met2_val)|/met2_val rigorously */
        (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, 
                  met2_val);
        if (met2_val != 0.0) {
            actual_err = fabs((exact_val-met2_val)/met2_val);
        } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
        else actual_err = DBL_MAX;
        if (NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    case 5: /* RR_LT */
        /* we control |exact_val-met2_val| approximately */
        (void) fprintf(fp, "Exact = %g, computed = %g, ", exact_val, 
                  met2_val);
        allow_err = TOLFACT*max_err;
        actual_err = fabs(exact_val-met2_val);
        if (NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    default: /* unknown method */
    }
ret = 1;
break;
} /* switch (meth) */
return ret;
}

/*
* Given in meth the numerical method with which the measure CCRD has been
* computed, in exact_val and met2_val the exact and computed values,
* respectively, in max_err the maximum error with which the computation has
* been performed, and in fp a pointer to the log file of the driver, checks
* whether the computed value is ok, printing to the file the appropriate
* message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
*/

int check_result_CCRD(int meth, double max_err, double exact_val, double met2_val,
                      FILE *fp)
{
  int ret;
  double allow_err, actual_err;
  ret = 0;
  switch (meth) {
    case 1: /* the only available method */
      /* we control exact_val-met2_val rigorously; moreover, met2_val
       * should be not larger than exact_val */
      (void) fprintf(fp, "exact = %g, computed = %g, \", exact_val,
                     met2_val);
      allow_err = max_err;
      actual_err = exact_val-met2_val;
      (void) fprintf(fp, "abs. err. = %g, \", actual_err);
      if (NOTLARGER(met2_val, exact_val) && NOTLARGER(actual_err,
                        allow_err)) {
        (void) fprintf(fp, "%s.n", PASS_MSG);
      } else {
        (void) fprintf(fp, "%s.n", FAIL_MSG);
      }
      break;
    default: /* unknown method */
      ret = 1;
      break;
  } /* switch (meth) */
  return ret;
}

/*
* Given in meth the numerical method with which the measure CIAVD has been
* computed, in exact_val and met2_val the exact and computed values,
* respectively, in max_err the maximum error with which the computation has
* been performed, and in fp a pointer to the log file of the driver, checks
* whether the computed value is ok, printing to the file the appropriate
* message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
*/

int check_result_CIAVD(int meth, double max_err, double exact_val, double met2_val,
                       FILE *fp)
{
  int ret;
  double allow_err, actual_err;
  ret = 0;
  switch (meth) {
    case 1: /* A */
    case 2: /* A_TRF */

/* we control exact_val-met2_val rigorously; moreover, met2_val should be not larger than exact_val */
(void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, met2_val);
allow_err = max_err;
actual_err = exact_val-met2_val;
(void) fprintf(fp, "abs. err. = %g, ", actual_err);
if (NOTLARGER(met2_val, exact_val) && NOTLARGER(actual_err, allow_err)) {
    (void) fprintf(fp, "%s.\n", PASS_MSG);
} else {
    (void) fprintf(fp, "%s.\n", FAIL_MSG);
}
break;
default: /* unknown method */
    ret = 1;
break;
} /* switch (meth) */
return ret;
}

/*
 * Given in meth the numerical method with which the measure ECRTE has been
 * computed, in exact_val and met2_val the exact and computed values,
 * respectively, in max_err the maximum error with which the computation has
 * been performed, and in fp a pointer to the log file of the driver, checks
 * whether the computed value is ok, printing to the file the appropriate
 * message. Returns 0 if meth is a valid method and an integer <>0 otherwise.
 */

int check_result_ECRTE(int meth, double max_err, double exact_val, double met2_val,
              FILE *fp)
{
    int ret;
    double allow_err, actual_err;
    ret = 0;
    switch (meth) {
        case 1: /* GS */
        case 2: /* SOR */
        case 3: /* GMRES */
        case 4: /* AGS */
        case 5: /* ASOR */
            /* we control |(exact_val-met2_val)|/met2_val approximately */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, met2_val);
            allow_err = TOLFACT*max_err;
            if (met2_val != 0.0) {
                actual_err = fabs((exact_val-met2_val)/met2_val);
            } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
            else actual_err = DBL_MAX;
            (void) fprintf(fp, "|rel. err.| = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s.\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s.\n", FAIL_MSG);
            }
            break;
default: /* unknown method */
                ret = 1;
            break;
    } /* switch (meth) */
    return ret;
}
int check_result_CRDTE(int meth, double max_err, double exact_val, double met2_val, FILE *fp)
{
    int ret;
    double allow_err, actual_err;

    ret = 0;
    switch (meth) {
    case 1: /* SR_ABS */
        /* we have rigorous control of exact_val=met2_val; moreover,
         * met2_val should be not larger than exact_val */
        (void) fprintf(fp, "exact = %g, computed = %g, \", exact_val, met2_val);
        allow_err = max_err;
        actual_err = exact_val - met2_val;
        if (NOTLARGER(met2_val, exact_val) && NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    case 2: /* SR_REL */
        /* we have rigorous control or |(exact_val-met2_val)|/met2_val */
        (void) fprintf(fp, "exact = %g, computed = %g, \", exact_val, met2_val);
        allow_err = max_err;
        if (met2_val != 0.0) {
            actual_err = fabs((exact_val-met2_val)/met2_val);
        } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
        else actual_err = DBL_MAX;
        (void) fprintf(fp, "|rel. err.| = %g, \", actual_err);
        if (NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
    case 4: /* RR_LT */
        /* we don't have rigorous control of exact_val=met2_val */
        (void) fprintf(fp, "exact = %g, computed = %g, \", exact_val, met2_val);
        allow_err = TOLFACT*max_err;
        actual_err = fabs(exact_val-met2_val);
        (void) fprintf(fp, "|abs. err.| = %g, \", actual_err);
        if (NOTLARGER(actual_err, allow_err)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else {
            (void) fprintf(fp, "%s\n", FAIL_MSG);
        }
        break;
default: /* either RR_B or unknown method */
    ret = 1;
    break;

} /* switch {meth} */

return ret;


/*
* Given in exact_val the exact value of the measure CRDTE, in met2_bound the
* computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum
* error with which the computation has been performed, and in fp a pointer to
* the log file of the driver, checks whether the bound is ok, printing to the
* file the appropriate message. Returns 0 if lb is either TRUE or FALSE and
* an integer <>0 otherwise.
*/

int
check_result_CRDTE_b(int lb, double max_err, double exact_val,
                      double met2_bound, FILE *fp)
{
    int ret;

    ret = 0;
    switch (lb) {
      /* the error in the bounds is bounded from above by max_err */
      case TRUE:
        (void) fprintf(fp, "exact = %g, lower bound = %g, ", exact_val,
                       met2_bound);
        /* neglecting round-off errors, the actual bound should not be
         * smaller than met2_bound */
        if (NOTLARGER(met2_bound, exact_val)) {
            (void) fprintf(fp, "%.%s\n", PASS_MSG);
        } else (void) fprintf(fp, "%.%s\n", FAIL_MSG);
        break;
      case FALSE:
        (void) fprintf(fp, "exact = %g, upper bound = %g, ", exact_val,
                       met2_bound);
        /* neglecting round-off errors, the actual bound should not be
         * larger than met2_bound+max_err */
        if (NOTSMALLER(met2_bound+max_err, exact_val)) {
            (void) fprintf(fp, "%.%s\n", PASS_MSG);
        } else (void) fprintf(fp, "%.%s\n", FAIL_MSG);
        break;
      default:
        ret = 1;
        break;
    }
    return ret;

    */

/*
* Given in meth1 and meth2 the numerical methods with which the measure ECRDS
* has been computed (see below), in exact_val and met2_val the exact and
* computed values, respectively, in max_err1 and max_err2 the maximum
* error/tolerance for meth1 and meth2, respectively, with which the
* computation has been performed, and in fp a pointer to the log file of the
* driver, checks whether the computed value is ok, printing to the file the
* appropriate message. Returns 0 if meth1 and meth2 are valid methods and an
* integer <>0 otherwise.
*/

int
check_result_ECRDS(int meth1, int meth2, double max_err1, double max_err2,
                    double exact_val, double met2_val, FILE *fp)
{
double allow_err, actual_err;
if (! (meth1 == 1 || meth1 == 2 || meth1 == 3 || meth1 == 4)) return 1;
if (! (meth2 == 1 || meth2 == 2 || meth2 == 3)) return 1;
if (max_err1 > max_err2) allow_err = TOLFACT*max_err1;
else allow_err = TOLFACT*max_err2;
/* we don't have rigorous control of |(exact_val-met2_val)/met2_val */
(void) fprintf(fp, "exact = %g, computed = %g, ", exact_val, met2_val);
if (met2_val != 0.0) {
    actual_err = fabs((exact_val-met2_val)/met2_val);
} else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
else actual_err = DBL_MAX;
(void) fprintf(fp, "|rel. err.| = %g, ", actual_err);
if (NOTLARGER(actual_err, allow_err)) {
    (void) fprintf(fp, "%s\n", PASS_MSG);
} else {
    (void) fprintf(fp, "%s\n", FAIL_MSG);
}
return 0;
*/

int check_result_CRDDS(int meth1, int meth2, int meth3, double max_err1,
    double max_err2, double max_err3, double exact_val,
    double met2_val, FILE *fp)
{
    int ret;
    double allow_err, actual_err;
    if (! (meth1 == 1 || meth1 == 2 || meth1 == 3 || meth1 == 4)) return 1;
    if (! (meth2 == 1 || meth2 == 2 || meth2 == 3)) return 1;
    switch (meth3) {
    case 1: /* SR_ABS */
        case 3: /* RR */
            /* we control exact_val-met2_val approximately */
            (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
                met2_val);
            allow_err = TOLFACT*max_err3;
            actual_err = exact_val-met2_val;
            (void) fprintf(fp, "abs. err. = %g, ", actual_err);
            if (NOTLARGER(actual_err, allow_err)) {
                (void) fprintf(fp, "%s\n", PASS_MSG);
            } else {
                (void) fprintf(fp, "%s\n", FAIL_MSG);
            }
            break;
    case 2: /* SR_REL */
        /* we control |(exact_val-met2_val)/met2_val approximately */
        (void) fprintf(fp, "exact = %g, computed = %g, ", exact_val,
            met2_val);
        allow_err = TOLFACT*max_err3;
        if (met2_val != 0.0) {
            actual_err = fabs((exact_val-met2_val)/met2_val);
        } else if (EQUAL(exact_val, met2_val)) actual_err = 0.0;
        else actual_err = DBL_MAX;
(void) fprintf(fp, "rel. err. = %g, actual err. = %g", actual_err);
if (NOTLARGER(actual_err, allow_err)) {
    (void) fprintf(fp, "abs. err. = %g", actual_err);
    allow_err = TOLFACT*max_err3;
    actual_err = fabs(exact_val-met2_val);
    (void) fprintf(fp, "abs. err. = %g", actual_err);
    if (NOTLARGER(actual_err, allow_err)) {
        (void) fprintf(fp, "%s\n", PASS_MSG);
    } else {
        (void) fprintf(fp, "%s\n", FAIL_MSG);
    }
} else {
    (void) fprintf(fp, "%s\n", FAIL_MSG);
}
break;
case 4: /* RR_LT */
    /* we control |exact_val-met2_val| approximately */
    (void) fprintf(fp, "exact = %g, computed = %g", exact_val, met2_val);
    allow_err = TOLFACT*max_err3;
    actual_err = fabs(exact_val-met2_val);
    (void) fprintf(fp, "abs. err. = %g", actual_err);
    if (NOTLARGER(actual_err, allow_err)) {
        (void) fprintf(fp, "%s\n", PASS_MSG);
    } else {
        (void) fprintf(fp, "%s\n", FAIL_MSG);
    }
    break;
default: /* either RR_B or unknown method */
    ret = 1;
    break;
} /* switch (meth3) */
return ret;

/*
 * Given in exact_val the exact value of the measure CRDDS, in met2_bound the
 * computed lower (lb=TRUE) or upper (lb=FALSE) bound, in max_err the maximum
 * error with which the computation has been performed, and in fp a pointer to
 * the log file of the driver, checks whether the bound is ok, printing to the
 * file the appropriate message. Returns 0 if lb is either TRUE or FALSE and
 * an integer <>0 otherwise.
 */

int check_result_CRDDS_b(int lb, double max_err, double exact_val,
    double met2_bound, FILE *fp)
{
    int ret;
    ret = 0;
    switch (lb) {
    /* the error in the bounds is bounded from above by max_err */
    case TRUE:
        (void) fprintf(fp, "exact = %g, lower bound = %g", exact_val, met2_bound);
        /* neglecting round-off errors, the actual bound should not be
         * smaller than met2_bound */
        if (NOTLARGER(met2_bound, exact_val)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else (void) fprintf(fp, "%s\n", FAIL_MSG);
        break;
    case FALSE:
        (void) fprintf(fp, "exact = %g, upper bound = %g", exact_val, met2_bound);
        /* neglecting round-off errors, the actual bound should not be
         * larger than met2_bound+max_err */
        if (NOTSMALLER(met2_bound+max_err, exact_val)) {
            (void) fprintf(fp, "%s\n", PASS_MSG);
        } else (void) fprintf(fp, "%s\n", FAIL_MSG);
        break;
    default:
        ret = 1;
        break;
}
return ret;
}

/*
 * Returns a pointer to newly allocated room appropriate for holding a message
 * consisting of ns sentences if ns>0 and DEF_SENT_SZ sentences
 * otherwise. Returns NULL if allocation fails.
 * NOTE: DEF_SENT_SZ is a macro defined in driver_lib.c.
 */

static
msg_t *
new_msg(int ns)
{
    msg_t *p;
    int i;

    if (!({p = calloc(1, sizeof(msg_t)))) return NULL;
    p->n_sent = 0;
    if (ns > 0) p->sents_sz = ns;
    else p->sents_sz = DEF_SENT_SZ;
    if (!({p->sents = realloc(p->sents, (p->sents_sz)*sizeof(char *))})) return NULL;
    for (i = 0; i < p->sents_sz; i++) {
        p->sents[i] = NULL;
    }
    return p;
}

/*
 * Adds sentence sent to message *p. Returns 0 on success and 1 on failure.
 */

static
int
add_sentence_to_msg(char *sent, msg_t *p)
{
    if (++(p->n_sent) > p->sents_sz) {
        int i, old_sz = p->sents_sz;

        p->sents_sz += DEF_SENT_SZ;
        if (!({p->sents = realloc(p->sents, (p->sents_sz)*sizeof(char *))})) {
            return 1;
        }
        for (i = old_sz; i < p->sents_sz; i++) {
            p->sents[i] = NULL;
        }
    }
    if (!({p->sents[p->n_sent-1] = calloc(LINEWIDTH+2, sizeof(char)))
            || strlen(sent) > LINEWIDTH+2}) {
            return 1;
        }
    (void) strncpy(p->sents[p->n_sent-1], sent);
    return 0;
}

/*
 * Frees message *p.
 */

static
void
free_msg(msg_t *p)
{
    int i;
for (i = 0; i < p->sents_sz; i++) {
    if (p->sents[i]) free(p->sents[i]);
} free(p->sents); free(p);
return;
}

/**
 * Returns a pointer to newly allocated room appropriate for holding a set of
 * nm messages if nm>0 and DEF_MSG_SZ messages otherwise. Returns NULL if
 * allocation fails.
 */
static
msg_set_t *
new_msg_set(int nm)
{
    msg_set_t *p;
    int i;
    if (!((p = calloc(1, sizeof(msg_set_t)))) return NULL;
    p->n_msg = 0;
    if (nm > 0) p->msgs_sz = nm;
    else p->msgs_sz = DEF_MSG_SZ;
    if (!((p->msgs = calloc(p->msgs_sz, sizeof(msg_t *))))) return NULL;
    for (i = 0; i < p->msgs_sz; i++) {
        p->msgs[i] = NULL;
    }
    return p;
}

/**
 * Creates an empty message in the set *setp of type kind with room for ns
 * sentences if ns>0 and DEF_SENT_SZ otherwise, returning 0 on success and 1
 * on failure.
 * 
 * NOTE: DEF_SENT_SZ is a macro defined in driver_lib.c.
 * 
 * NOTE: Use add_to_msg() below to add sentences to the message.
 */
static
int
create_empty_msg(msg_set_t *setp, int kind, int ns)
{
    if (++(setp->n_msg) > setp->msgs_sz) {
        int i, old_sz = setp->msgs_sz;
        setp->msgs_sz += DEF_SENT_SZ;
        if (!((setp->msgs =
                realloc(setp->msgs, (setp->msgs_sz)*sizeof(msg_t *))))) {
            return 1;
        }
        for (i = old_sz; i < setp->msgs_sz; i++) {
            setp->msgs[i] = NULL;
        }
    }
    if (!((setp->msgs[setp->n_msg-1] = new_msg(ns)))) return 1;
    (setp->msgs[setp->n_msg-1])->idx = setp->n_msg;
    (setp->msgs[setp->n_msg-1])->type = kind;
    (setp->msgs[setp->n_msg-1])->flag = FALSE;
    return 0;
}
/*
 * Adds sentence *sent to the current message of the set *setp. (The current
 * message is the one internally set up by the last call to
 * create_empty_msg().) Returns 0 on success and 1 on failure.
 */
static
int
add_to_msg(char *sent, msg_set_t *setp)
{
    if (setp->n_msg == 0) return 1;
    else return add_sentence_to_msg(sent, setp->msgs[setp->n_msg-1]);
}

/*
 * Frees set of messages *p.
 */
static
void
free_msg_set(msg_set_t *p)
{
    int i;

    for (i = 0; i < p->n_msg; i++) {
        if (p->msgs[i]) free_msg(p->msgs[i]);
    }
    free(p->msgs);
    free(p);
    return;
}

/*
 * Given an array of chars line of size sz, scans it until a null character is
 * found or its end is reached, and returns: 0 if line is null terminated and
 * not empty (1); 1 if line is null terminated and empty (1); and 2 otherwise.
 * (1) A null-terminated empty line is one that consists of only blank, tab
 * and new-line characters.
 */
static
int
easy_line(char *line, int sz)
{
    int empty, i;
    char *s;

    if (sz == 0) return 2; /* not null-terminated line */
    empty = 1;
    for (s = line, i = 1; i <= sz && *s; s++) {
        if (empty) {
            empty = (s == ' ' || s == '	' || s == '
')? 0: 1;
        }
    }
    if (i > sz) return 2; /* not null-terminated line */
    else if (empty) return 1; /* null-terminated, empty line */
    else return 0; /* null-terminated, non-empty line */
}

/*
 * Given a model name *modname, reads from the appropriate file a set of
 * expected error and warning messages, returning them in the newly allocated
* data structures **experrp and **expwarp, respectively. The name of the file
* is built by appending to *modname the string that results from the
* concatenation of: EXPECTED_MSG_FILE_SUFFIX, an underscore, and the integer
* vect_idx translated into a string. The funcion assumes that in the file,
* the body of each error message is enclosed by EXP_ERR_MSG_BEGIN and
* EXP_ERR_MSG_END, and the body of each warning message is enclosed by
* EXP_WARN_MSG_BEGIN and EXP_WARN_MSG_END. The function returns 0 on success
* and 1 on failure.
* 
* NOTE: lines that are empty according to empty_line() are discarded.
* 
* NOTE: EXPECTED_MSG_FILE_SUFFIX, EXP_ERR_MSG_BEGIN, EXP_ERR_MSG_END,
* EXP_WARN_MSG_BEGIN, and EXP_WARN_MSG_END are macros defined in the
* test_conf.h file.
*/

int
read_expected_msgs(const char *modname, const int vect_idx,
msg_set_t **experrp, msg_set_t **expwarp)
{
char tmp[BUF_SZ+2], line[LINEN_WIDTH+2];
FILE *fp;
int noteof, retval,
state;  /* 0 if expecting a new message to begin, 1 if reading an
* error message, and 2 if reading a warning message */

if (!(*experrp = new_msg_set(0))) return 1;
if (!(*expwarp = new_msg_set(0))) return 1;
strcpy(tmp, modname);
strcat(tmp, EXPECTED_MSG_FILE_SUFFIX);
strcat(tmp, "\n");
if (fp = fopen(tmp, "%d", vect_idx)) return 1;

state = 0;
noteof = (fgets(line, LINEN_WIDTH+2, fp) == NULL? FALSE: TRUE);
while (noteof) |
switch (state) {
    case 0:
        if (strstr(line, EXP_ERR_MSG_BEGIN)) {
            state = 1;
            if (create_empty_msg(*experrp, EXP_MSG, 0) != 0) return 1;
        } else if (strstr(line, EXP_WARN_MSG_BEGIN)) {
            if (create_empty_msg(*expwarp, EXP_MSG, 0) != 0) return 1;
            state = 2;
        } else return 1;
        break;
    case 1:
        if (strstr(line, EXP_ERR_MSG_BEGIN)
            || strstr(line, EXP_WARN_MSG_BEGIN)
            || strstr(line, EXP_WARN_MSG_END)) {
            /* non-properly closed message */
            return 1;
        } else {
            if (strstr(line, EXP_WARN_MSG_END)) state = 0;
            else if (!(*retval = empty_line(line, LINEN_WIDTH+2))) {
                /* null-terminated, non-empty line */
                if (add_to_msg(line, *experrp) != 0) return 1;
            } else if (retval == 2) { /* not null-terminated line */
                return 1;
            } else return 1;
        } else return 1;
        break;
    case 2:
        if (strstr(line, EXP_WARN_MSG_BEGIN)
```c
|| strstr(line, EXP_ERR_MSG_BEGIN)
|| strstr(line, EXP_ERR_MSG_END))
    /* non-properly closed message */
    return 1;
} else {
    if (strstr(line, EXP_WARN_MSG_END)) state = 0;
    else {
        if (!empty_line(line, LINEWIDTH+2)) {
            /* null-terminated, non-empty line */
            if (add_to_msg(line, *expwarn) != 0) return 1;
        } else if (retval == 2) {
            /* not null-terminated line */
            return 1;
        }
    }
}
break;
default:
    (void) fprintf(fp, "read_expected_msgs(): unknown case.\n");
    exit(EXIT_FAILURE);
    break;
}
    noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
}
if ((*experr)->n_msg > 0) {
    /* check there aren't empty error messages */
    int i;
    msg_t *p;
    for (i = 0; i < (*experr)->n_msg; i++) {
        p = (*experr)->msgs[i];
        if (p->n_sent == 0) return 1;
    }
}
if ((*expwarn)->n_msg > 0) {
    /* check there aren't empty warning messages */
    int i;
    msg_t *p;
    for (i = 0; i < (*expwarn)->n_msg; i++) {
        p = (*expwarn)->msgs[i];
        if (p->n_sent == 0) return 1;
    }
}
if (fclose(fp) != 0 || state != 0) {
    /* state!=0: some message not properly closed */
    return 1;
} else return 0;
}

/*
 * Given a model name *modname, reads from the appropriate file the set of
 * error and warning messages generated by the model, returning them in the
 * newly allocated data structures **met2errp and **met2warnp, 
 * respectively. The name of the file is built by appending the string ".log"
 * to *modname. The function assumes that each message has the following
 * structure:
 * 
 * <anything> <header_beg> <anything> <header_end> <first_meaningful_sentence>
 * <zero_or_more_meaningful_sentences_mixed_with_empty_lines>
 * 
 * where for error messages, <header_beg> and <header_end> are, respectively,
 * the strings to which the macros ERR_MSG_HEADER_BEGIN and ERR_MSG_HEADER_END
 * expand, and for warning messages, <header_beg> and <header_end> are,
 * respectively, the strings to which the macros WAR_MSG_HEADER_BEGIN and
 * WAR_MSG_HEADER_END expand.
 */
int read_met2_msgs(const char *modname, msg_set_t **met2errp,
    msg_set_t **met2warp)
{
    char tmp[BUF_SZ+2], line[LINWIDTH+2], *s;
    FILE *fp;
    int noteof, retval,
    state;  /* 0 if expecting a new message to begin, 1 if reading the
    * body of an error message, and 2 if reading the body of
    * a warning message */
    if (!(*met2errp = new_msg_set(0))) return 1;
    if (!(*met2warp = new_msg_set(0))) return 1;
    strcpy(tmp, modname);
    strcat(tmp, ".log");
    if (!(*(fp = fopen(tmp, "r"))) return 1;

    state = 0;
    noteof = (fgets(line, LINWIDTH+2, fp) == NULL? FALSE: TRUE);
    while (noteof) {
        switch (state) {
        case 0:  
            if (strstr(line, ERR_MSG_HEADER_BEGIN)
                && (s = strstr(line, ERR_MSG_HEADER_END))) {
                if (create_empty_msg(*met2errp, READ_MSG, 0) != 0
                    || add_to_msg(s+strlen(ERR_MSG_HEADER_END),
                    *met2errp) != 0) return 1;
                state = 1;
            } else if (strstr(line, WAR_MSG_HEADER_BEGIN)
                && (s = strstr(line, WAR_MSG_HEADER_END))) {
                if (create_empty_msg(*met2warp, READ_MSG, 0) != 0
                    || add_to_msg(s+strlen(WAR_MSG_HEADER_END),
                    *met2warp) != 0) return 1;
                state = 2;
            }
            line[0] = '\0'; /* eat up line */
            break;
        case 1:  
            if (strstr(line, ERR_MSG_HEADER_BEGIN)
                || strstr(line, WAR_MSG_HEADER_BEGIN)) {
                state = 0;
            } else {
                if (!((retval = empty_line(line, LINWIDTH+2)))
                    /* null-terminated, non-empty line */
                    if (add_to_msg(line, *met2errp) != 0) return 1;
                } else if (retval == 2) {  
                    /* not null-terminated line */
                    return 1;
                }
                line[0] = '\0'; /* eat up line */
            }
            break;
        case 2:  
            if (strstr(line, ERR_MSG_HEADER_BEGIN)
                || strstr(line, WAR_MSG_HEADER_BEGIN)) {
                state = 0;
            } else {
                if (!((retval = empty_line(line, LINWIDTH+2)))
                    /* null-terminated, non-empty line */
                    if (add_to_msg(line, *met2warp) != 0) return 1;
                } else if (retval == 2) {  
                    /* null-terminated, non-empty line */
                } else if (retval == 1) {  
                    /* null-terminated, empty line */
                }
            }
            break;
        default:  
            ;
    }  
}
/* not null-terminated line */
return 1;
}
}
break;
default:
(void) fprintf(fp, "read_message(): unknown case.\n");
exit (EXIT_FAILURE);
break;
}
if (line[0] == '\0')
/* eaten-up line */
noteof = (fgets(line, LINEWIDTH+2, fp) == NULL) ? FALSE : TRUE
if (fclose(fp) != 0) return 1;
else return 0;
}

/*
 * Deletes dots, commas, colons, semicolons, blank spaces, tabs, and new line
 * characters from the string of chars line. Returns 0 on success and 1 on
 * failure.
 */
static
int
prune_line(char *line)
{
char *lp, *bp;
for (bp = lp = line; *lp; ++lp)
    if (!(*lp == ' ')
        || *lp == ','
        || *lp == '!
        || *lp == ';'
        || *lp == '
        || *lp == '"
        || *lp == '"
        || *lp == '"
    )
        *bp++ = *lp;
if (bp == line) return 1; /* null line */
    *bp = '\0';
    return 0;
}

/*
 * Deletes dots, commas, colons, semicolons, blank spaces, tabs, and new line
 * characters from the set of messages *setp. Returns 0 on success and 1 on
 * failure.
 */
static
int
prune_msg_set(msg_set_t *setp)
{
    int i, j;
    msg_t *msgp;
    for (i = 1; i <= setp->n_msg; i++)
        msgp = setp->msgs[i-1];
    for (j = 1; j <= msgp->n_sent; j++)
        if (prune_line(msgp->sents[j-1]) != 0) return 1;
    return 0;
}
/*
* Allocates a new set of messages, copies the set *setp into it and returns a
* pointer to the newly allocated set, or NULL if something fails.
*/
static
msg_set_t *
cpy_msg_set(msg_set_t *setp)
{
  msg_set_t *newsetp;
  msg_t *oldmsgp;
  int i, j;

  if (!(newsetp = new_msg_set(setp->n_msg))) return NULL;
  for (i = 1; i <= setp->n_msg; i++) {
    oldmsgp = setp->msgs[i-1];
    if (create_empty_msg(newsetp, 
        oldmsgp->type, oldmsgp->n_sent) != 0) return NULL;
    for (j = 1; j <= oldmsgp->n_sent; j++) {
      if (add_to_msg(oldmsgp->sents[j-1], newsetp) != 0) return NULL;
    }
  }
  return newsetp;
}

/*
* Comparison function to sort messages lexicographically.
*/
static
int
msg_sort_cmp(const void *a, const void *b)
{
  const msg_t **ma = (const msg_t **)a, **mb = (const msg_t **)b;
  int n_sent_both, i, res;

  if ((*ma)->n_sent < (*mb)->n_sent) n_sent_both = (*ma)->n_sent;
  else n_sent_both = (*mb)->n_sent;
  res = 0;
  for (i = 1; i <= n_sent_both && res == 0; i++) {
    res = strcmp((*ma)->sents[i-1], (*mb)->sents[i-1]);
  }
  if (res == 0) {
    if ((*ma)->n_sent < (*mb)->n_sent) res = -1;
    else if ((*ma)->n_sent == (*mb)->n_sent) {
      if ((*ma)->idx < (*mb)->idx) res = -1;
      else res = 1; /* indices are always different */
    } else res = 1;
  }
  return res;
}

/*
* Compares the read message *readmsgp against the expected one *expmsgp:
* letting n be the number of sentences of the expected message, the function
* returns 0 if both messages match up to the first n sentences and a value
* <>0 otherwise.
*/
static
int
msg_lookup_comp(msg_t *readmsgp, msg_t *expmsgp)
{
int i, res;

for (res = 0, i = 1; i <= expmsgp->n_msg && res == 0; i++) {
    res = strcmp(readmsgp->sents[i-1], expmsgp->sents[i-1]);
} return res;

/*
 * Looks up the read message *readp in the set *expp of expected messages,
 * returning a pointer to the matching expected message if found and NULL
 * otherwise.
 */

static
msg_t *
lookup_msg(msg_t *readp, msg_set_t *expp)
{
    int i, res;
    msg_t *msgp;

    res = 1;
    for (i = 1; i <= expp->n_msg && res != 0; i++) {
        msgp = expp->msgs[i-1];
        if (msgp->flag == FALSE
            && (res = msgp_lookup_comp(readp, msgp)) == 0) msgp->flag = TRUE;
    }
    if (res == 0) return msgp;
    else return NULL;
}

/*
 * Given a set *met2p of read messages and a set *expp of expected ones,
 * compares the first against the second, prints appropriate message(s) to the
 * file *fp according to the results of the comparison, and frees both
 * sets. The function returns 0 on success and 1 on failure.
 *
 * NOTE: if both sets are empty, the function frees them and returns 0 without
 * doing anything else.
 *
 * NOTE: the function works as follows. For each read message in *met2p, it
 * looks for an expected message in *expp such that letting n be its number of
 * sentences, all the n sentences of the expected message match up in a
 * one-by-one fashion with the first n sentences of the read message
 * regardless dots, commas, colons, semicolons, blank spaces, tabs, and new
 * line characters. If such an expected message is found, the read one is
 * regarded as correct; otherwise, the read message is regarded as incorrect.
 */

int
check_msgs(msg_set_t *met2p, msg_set_t *expp, FILE *fp)
{
    int i, j;
    msg_set_t *newmet2p, *newexpp;
    msg_t *tmpmsgp, *met2msg, *exppmsg;

    if (met2p->n_msg == 0 && expp->n_msg == 0) return 0;

    newmet2p = newexpp = NULL;
    /* make a copy of the sets of read and expected messages, prune the
     * messages in the first set and prune and sort lexicographically the
     * messages in the second set */
    if (!newexpp = cpy_msg_set(expp))
        goto on_error;
    if (prune_msg_set(newexpp) != 0)
prune_msg_set(newmet2p) != 0) goto on_error;

qsort(newexpxp->msgs, newexpxp->n_msg, sizeof(msg_t *), msg_sort_cmp);
for (i = 1; i <= met2p->n_msg; i++) {
  met2msg = met2p->msgs[i-1];
  (void) fprintf(fp, "READ MESSAGE:\n");
  for (j = 1; j <= met2msg->n_sent; j++) {
    (void) fprintf(fp, "%s", met2msg->sents[j-1]);
  }
  if ((tmppmsg = lookup_msg(newmet2p->msgs[i-1], newexpxp))) {
    expmsg = expxp->msgs[tmppmsg->idx-1];
    (void) fprintf(fp, "MATCHES UP TO THE FIRST %d LINE(S) "
      "WITH THE EXPECTED MESSAGE:\n",
    expmsg->n_sent);
    for (j = 1; j <= expmsg->n_sent; j++) {
      (void) fprintf(fp, "%s", expmsg->sents[j-1]);
    }
  }
  else {
    (void) fprintf(fp, "\%s.\n\n", PASS_MSG);
  }
  (void) fprintf(fp, "DOES NOT MATCH UP WITH ANY EXPECTED "
    "MESSAGE.\n\%s\n\n", FAIL_MSG);
}
}
for (i = 1; i <= newexpxp->n_msg; i++) {
  tmppmsg = newexpxp->msgs[i-1];
  if (tmppmsg->flag == FALSE) {
    expmsg = expxp->msgs[tmppmsg->idx-1];
    (void) fprintf(fp, "EXPECTED MESSAGE:\n");
    for (j = 1; j <= expmsg->n_sent; j++) {
      (void) fprintf(fp, "%s", expmsg->sents[j-1]);
    }
    (void) fprintf(fp, "DOES NOT MATCH "
      "WITH ANY READ MESSAGE.\n\%s\n\n", FAIL_MSG);
  }
  tmppmsg->flage = TRUE; /* Do not reuse tmppmsg. */
  free_msg_set(newexpxp);
  free_msg_set(newmet2p);
  return 0;
}

on_error:
  free_msg_set(newexpxp);
  free_msg_set(newmet2p);
  if (newexpxp) free_msg_set(newexpxp);
  return 1;

/* Looks for expmsg[] or nonexpmsg[] in the log file of the model modname[],
 * printing PASS_MSG to the file *outfp if the string found is expmsg[] and
 * FAIL_MSG if the string found is nonexpmsg[]. Returns 0 on success and 1 on
 * failure (e.g. the log file contains neither of those strings).
 * NOTE: PASS_MSG and FAIL_MSG are macros defined in test_conf.h. */

int
check_msg_logfile(const char modname[], const char expmsg[],
                   const char nonexpmsg[], FILE *outfp)
{
  char tmp[BUF_SZ+2], line[LINWIDTH+2], *e, *ne;
  FILE *fp;
  strcpy(tmp, modname);
  strcat(tmp, ".log");
  if (! (fp = fopen(tmp, "r"))) return 1;

int check_Markov_chain(const char modname[], FILE *outfp) {
    char tmp[BUF_SZ+2], line[LINENUMTH+2], *e, *ne,
    expmsg[] = CTMCGEN_OK_MSG, nonexpmsg[] = CTMCGEN_BAD_MSG;
    FILE *fp;

    strcpy(tmp, modname);
    strcat(tmp, "log");
    if (!(fp = fopen(tmp, "r"))) return 1;
    e = ne = NULL;
    do {
        e = strstr(line, expmsg);
        ne = strstr(line, nonexpmsg);
    } while (!feof(fp) || e || ne);
    if (fclose(fp) || (e && e)) return 1;
    if (e) {void) fprintf(outfp, "\n\n\n\n", DRIVER_LOG_BEG_CTMCGENRES);
        if (e) {void) fprintf(outfp, "\n\n\n\n", PASS_MSG);
            else {void) fprintf(outfp, "\n\n\n\n", FAIL_MSG);
                return 0;
            }
    }

    /*
     * Allocates room for the set of models, returning 0 on success and an integer
     * <>0 if allocation fails.
     */
    static int allocator();
    if (!(msetp_g = calloc(1, sizeof(modset_t)))) return 1;
msetp_g->n_mod = 0;
if (!({msetp_g->mods = calloc(DEF_N_MOD, sizeof(modinf_t)))}) return 1;
        msetp_g->mods_by_nm = NULL;
        msetp_g->mods_by_tp_nm = NULL;
        msetp_g->mods_sz = DEF_N_MOD;
        return 0;
    }

    /*
    * Frees the set of models.
    */
    void
    free_modset(void)
    {
        if (msetp_g) {
            if (msetp_g->mods) free(msetp_g->mods);
            if (msetp_g->mods_by_nm) free(msetp_g->mods_by_nm);
            if (msetp_g->mods_by_tp_nm) free(msetp_g->mods_by_tp_nm);
            msetp_g->n_mod = 0;
            msetp_g->mods = NULL;
            msetp_g->mods_by_nm = NULL;
            msetp_g->mods_by_tp_nm = NULL;
            msetp_g->mods_sz = 0;
            free(msetp_g);
            msetp_g = NULL;
        }
        return;
    }

    /*
    * Creates a new entry in the set of models with name nm, type t, no. of
    * expected vectors n, no. of expected vectors that are supposed to pass the
    * corresponding test np, and no. of expected vectors that are supposed not to
    * pass the corresponding test nf. Returns 0 on success and an integer <>0 on
    * failure.
    */
    static
    int
    new_entry_modset(const char nm[], const int t, const int n, const int np,
                     const int nf)
    {
        if (++(msetp_g->n_mod) > msetp_g->mods_sz) {
            msetp_g->mods_sz += DEF_N_MOD;
            if (!({msetp_g->mods = realloc(msetp_g->mods,
                        msetp_g->mods_sz* sizeof(modinf_t))))}) return 1;
        }
        (void) strcpy(msetp_g->mods[msetp_g->n_mod-1].name, nm);
        msetp_g->mods[msetp_g->n_mod-1].type = t;
        msetp_g->mods[msetp_g->n_mod-1].n_exp_vect = n;
        msetp_g->mods[msetp_g->n_mod-1].n_exp_vect_pass = np;
        msetp_g->mods[msetp_g->n_mod-1].n_exp_vect_fail = nf;
        return 0;
    }

    /*
    * Reads from file *fname the set of models. The file has to contain a
    * non-empty line per model (empty lines are discarded) with the following
    * structure:
    *  *<name><sep>"MEAS," |"MSG," |"MODGEN," <$vect><sep><$ok_vect><sep><$wrong_vect>
    *
* where:
* 
* <name>: model name
* 
* <sep>: one or more of: white space, comma, period, colon, semicolon, tab
* 
* MEAS: the model is intended for testing the computation of measures
* 
* MSG: the model is intended for testing warning and error messages
* 
* MODGEN: the model is intended for testing the generation of the model
* 
* <#vect>: no. of vectors to be generated based on the model
* 
* <#ok_vect>: no. of generated vectors that are supposed to pass the 
* corresponding test
* 
* <#wrong_vect>: no. of generated vectors that are supposed not to pass the 
* corresponding test
* 
* The function returns 0 if the list has been read successfully and an 
* integer != 0 otherwise.
* */

int
read_mod_set(const char fname[])
{
    FILE *fp;
    int noteof, type, n_exp_vect, n_exp_vect_pass, n_exp_vect_fail;
    char *name, *typestr, line[LINEWIDTH+2], *s;
    const char sep[] = " ,;:\t";

    if (ALLOC_MODSET() != 0) return 1;
    if (! (fp = fopen(MOD_LIST_FILE, "r"))) {
        (void) fprintf(stderr, "read_mod_list(): unable to open 
" "file '%s.'\n", fname);
    }
    noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    while (noteof) {
        if ((name = strtok(line, sep))) { /* non-empty line */
            /* type */
            if (! (typestr = strtok(NULL, sep))) return 1;
            else if (strcmp(typestr, MOD_MEAS_STR) == 0) type = MOD_MEAS;
            else if (strcmp(typestr, MOD_WARNING_STR) == 0) type = MOD_WARNING;
            else if (strcmp(typestr, MOD_MODGEN_STR) == 0) type = MOD_MODGEN;
            else return 1;
            /* vectors */
            if (! (s = strtok(NULL, sep))
                | | sscanf(s, "%d", &n_exp_vect) != 1) return 1;
            if (! (s = strtok(NULL, sep))
                | | sscanf(s, "%d", &n_exp_vect_pass) != 1) return 1;
            if (! (s = strtok(NULL, sep))
                | | sscanf(s, "%d", &n_exp_vect_fail) != 1) return 1;
            if (new_entry_modset(name, type, n_exp_vect, n_exp_vect_pass, 
                n_exp_vect_fail) != 0) return 1;
        }
        noteof = (fgets(line, LINEWIDTH+2, fp) == NULL? FALSE: TRUE);
    }
    if (fclose(fp) != 0) {
        (void) fprintf(stderr, "read_mod_list(): unable to close 
" "file '%s.'\n", fname);
        exit(EXIT_FAILURE);
    }
    return 0;
}
* Comparison function to sort the set of models in ascending order of the
* field 'name'.
*/

static
int
modp_cmp_nm(const void *a, const void *b)
{
    const modinf_t **ma = (const modinf_t **)a,
    **mb = (const modinf_t **)b;
    return (strcmp(('ma)->name, ('mb)->name));
}

/**
* Comparison function to sort the set of models in ascending order of, first,
* the field 'type' and, next, the field 'name'.
*/

static
int
modp_cmp_tp_nm(const void *a, const void *b)
{
    const modinf_t **ma = (const modinf_t **)a,
    **mb = (const modinf_t **)b;
    if (($(ma)->type < $(mb)->type) return -1;
    else if ($(ma)->type == $(mb)->type) return (strcmp((ma)->name,
                                 (mb)->name));
    else return 1;
}

/**
* Sorts the set of models in increasing order of the field 'name' if
* onlyname<>0, and in increasing order of first the field 'type' and next the
* field 'name' otherwise. Returns 0 on success and an integer <>0 on failure.
*/

static
int
sort_modset(int onlyname)
{
    int i, dup;
    modinf_t *ml, *m2;

    dup = 0; /* to check there aren't two models with the same name */
    if (onlyname) {
        if (!((msetp_g)->mods_by_nm =
            malloc((msetp_g)->n_mod, sizeof(modinf_t *))))) return 1;
        for (i = 1; i < msetp_g->n_mod; i++) {
            msetp_g->mods_by_nm[i-1] = &((msetp_g)->mods[i-1]);
        }
        qsort(msetp_g->mods_by_nm, msetp_g->n_mod, sizeof(modinf_t *),
             modp_cmp_nm);
        /* check there aren't duplicates */
        for (i = 1; i < msetp_g->n_mod && !dup; i++) {
            ml = msetp_g->mods_by_nm[i-1];
            m2 = msetp_g->mods_by_nm[i];
            dup = (strcmp(ml->name, m2->name) == 0? 1: 0);
        }
        else {
            if (!((msetp_g)->mods_by_tp_nm =
                malloc((msetp_g)->n_mod, sizeof(modinf_t *))))) return 1;
            for (i = 1; i < msetp_g->n_mod; i++) {
                msetp_g->mods_by_tp_nm[i-1] = &((msetp_g)->mods[i-1]);
            }
            qsort(msetp_g->mods_by_tp_nm, msetp_g->n_mod, sizeof(modinf_t *)},
```c
modp_cmp_tp_nm);

if (dup) {
    (void) fprintf(stderr, "sort_modset(): there are two models with "
    "the same name ("\"%s\").\n", ml->name);
    return 1;
} else return 0;

/*
 * Looks up a model with name *modname in the set of models and, if it finds
 * the model, returns 0 and sets *n_exp_vect, *n_exp_vect_pass, and
 * *n_exp_vect_fail to, respectively, the number of vectors expected for the
 * model, the number of such vectors that are supposed to pass the appropriate
 * test, and the number of these vectors that are supposed not to pass the
 * test. Otherwise, the function returns an integer <>0.
 */

static
int
mod_info(const char *modname, int *n_exp_vect, int *n_exp_vect_pass,
    int *n_exp_vect_fail)
{
    static int sorted = 0;
    modinf_t key, *keyp, **cell;
    if (!sorted) {
        /* sort in ascending order of the field 'name' */
        if (sort_modset(1) != 0) return 1;
        sorted = 1;
    }
    (void) strcpy(key.name, modname);
    keyp = &key;
    if (!{cell = bsearch(&keyp, msetp_g->mods_by_nm, msetp_g->n_mod,
            sizeof(modinf_t *)),
        modp_cmp_nm)) return 1;
    *n_exp_vect = (*cell)->n_exp_vect;
    *n_exp_vect_pass = (*cell)->n_exp_vect_pass;
    *n_exp_vect_fail = (*cell)->n_exp_vect_fail;
    return 0;
}

/*
 * Prints to the standard output a one-line list of models of type tp sorted
 * in increasing order of the name of the model.
 */

int
pr_modset(int tp)
{
    static int sorted = 0;
    int i;
    modinf_t *p;
    if (!sorted) {
        /* sort in ascending order of first the field 'type' and next the
         * field 'name' */
        if (sort_modset(0) != 0) return 1;
        sorted = 1;
    }
    for (i = 1; i <= msetp_g->n_mod; i++) {
        p = msetp_g->mods_by_tp_nm[i-1];
        if (p->type == tp) {
            (void) printf("%s ", p->name);
        }
    }
```
return 0;
}

/*
 * If the environment variable M2TEST_MEMDBG is defined and set to anything
 * different from the empty string, opens file *from, writes its contents
 * wrapped up with *beg_str and end_str to the already opened file pointed to
 * by tofp, and _removes_file *from, returning 0 on success and an integer
 * <>0 on failure. Otherwise, the function does nothing and returns 0.
 */

int
append_file(FILE *tofp, char *from, char *beg_str, char *end_str)
{
    static int firstcall = TRUE, memdbg;
    if (firstcall) {
        char *s;
        s = getenv("M2TEST_MEMDBG");
        if (s && *s) memdbg = TRUE;
        else memdbg = FALSE;
        firstcall = FALSE;
    }
    if (memdbg) {
        /* actually scan the file */
        FILE *fromp;
        char line[LINENLENGTH+2];
        if (! (fromp = fopen(from, "r"))) return 1;
        (void) fprintf(tofp, "%s\n", DRIVER_LOG_BEG_MEMRES);
        while (fgets(line, LINENLENGTH+2, fromp)) {
            (void) fprintf(tofp, "%s", line);
        }
        (void) fprintf(tofp, "%s\n\n", DRIVER_LOG_END_MEMRES);
        if (fclose(fromp) != 0 || remove(from) != 0) return 1;
        else return 0;
    }
    return 0;
}

/*
 * Given the name of a model in *modname, looks for errors in each run of the
 * model that has been reported in the log file of the corresponding model
 * driver. For each such run, the function prints (standard output) the string
 * to which the macro PASS_MSG expands if no errors are found, and the string
 * to which the macro FAIL_MSG expands plus some other info otherwise. In
 * addition, once all runs have been processed, the function prints a brief
 * summary telling how many runs have been examined, in how many of them one
 * or more errors have been found, and whether those numbers agree with the
 * expected ones read from the file 'test_conf.modset'. The function returns 0
 * on success and an integer <>0 upon failure.
 */

int
lookfor_errors(const char *modname)
{
    FILE *fp;
    char tmp[BUF_SZ], line[LINENLENGTH+2], vectident[10*LINENLENGTH+2], *s;
    int st,
        /* state of the automaton in charge of scanning the
        * file */
        nummeth_err, /* no. of errors regarding numerical results in the
        * current vector */
        mem_leak_err, /* no. of memory leaks */
        mem_access_err, /* no. of memory access errors */
        memtest_done, /* 1 if memory management has been checked and 0
msg_err, /* no. of errors in error and warning messages */
modgen_err, /* no. of errors in the generation of the model */
ctmcgen_err, /* no. of errors in the generation of the CTMC */
ctmcgentest_done, /* 1 if test of the generation of the CTMC has been 
done and 0 otherwise */
n_vect, /* no. of vectors in the file */
n_f_vect, /* no. of vectors in the file for which any error has 
been found */
os, /* TRUE if we're working under Linux; FALSE */
linuxbox, /* otherwise */
n_exp_vect, /* expected no. of vectors */
n_exp_vect_pass,/* expected no. of vectors that are supposed to pass 
* the appropriate test */
n_exp_vect_fail; /* expected no. of vectors that are supposed not to 
* pass the appropriate test */

long nonfreed;

os = os_type();
if (os == OS_LINUX) linuxbox = TRUE;
else if (os == OS_SOLARIS) linuxbox = FALSE;
else return 1; /* unknown OS */
build_driver_logfilename(modname, tmp);
strcpy(tmp, DRIVER_FILE_PREFIX);
strcat(tmp, modname);
strcat(tmp, DRIVER_OUT_FILE_SUFFIX);
if (!(fp = fopen(tmp, "r"))) return 1;

nummeth_err =
mem_leak_err =
mem_access_err =
memtest_done =
msg_err =
modgen_err =
ctmcgen_err =
ctmcgentest_done = 0;
nonfreed = 0;
n_vect = n_f_vect = 0;
st = 0;
vectident[0] = '\0';

while (fgets(line, LINEWIDTH+2, fp)) {
  switch (st) {
    /*
    * st = 0: start
    * st = 1: getting the model identity (name, measure, method(s),
    * etc)
    * st = 2: checking whether the numerical results are ok
    * st = 3: checking whether memory management is ok
    * st = 4: checking whether error and warning messages are ok
    * st = 5: checking whether the generation of the model is ok
    * st = 6: checking whether the generation of the CTMC is ok
    */
    case 0:
      if (strstr(line, DRIVER_LOG_BEG_IDENT)) {
        if (vectident[0]) {
          /* output previous vector results */
          ++n_vect;
          (void) fprintf(stderr, "%s\n", vectident);
        if (nummeth_err || mem_leak_err || mem_access_err

|| msg_err || modgen_err || ctmcgen_err) { 
/* some error has been found */
++n_f_vect;
(void) fprintf(stdout, FAIL_MSG".\n\n");
if (nummeth_err) {
(void) fprintf(stdout, "Numerical results: 
"found %d errors\n",
nummeth_err);
nummeth_err = 0;
}
if (mem_access_err) {
(void) fprintf(stdout, "Memory management: 
"found %d access errors\n",
mem_access_err);
mem_access_err = 0;
}
if (mem_leak_err) {
(void) fprintf(stdout, "Memory management: 
"found %d leaks\n",
mem_leak_err);
mem_leak_err = 0;
nonfreed = 0;
}
if (msg_err) {
(void) fprintf(stdout, "Error and warning 
"messages: found %d errors\n",
msg_err);
msg_err = 0;
}
if (modgen_err) {
(void) fprintf(stdout, "Model generation: 
"found %d errors\n",
modgen_err);
modgen_err = 0;
}
if (ctmcgen_err) {
(void) fprintf(stdout, "CTMC generation: 
"found %d errors\n",
ctmcgen_err);
ctmcgen_err = 0;
}
else {
/* no error has been found */
(void) fprintf(stdout, PASS_MSG".\n\n");
}
vecident[0] = ' \0';
}
else if (strstr(line, DRIVER_LOG_BEG_NUMRES)) {
st = 2;
}
else if (strstr(line, DRIVER_LOG_BEG_MEMRES)) {
st = 4;
}
else if (strstr(line, DRIVER_LOG_BEG_MSRES)) {
st = 5;
}
else if (strstr(line, DRIVER_LOG_BEG_CTMCGENRES)) {
st = 6;
}
break;
case 1:
if (strstr(line, DRIVER_LOG_END_IDENT)) {
st = 0;
}
else if (strstr(line, DRIVER_LOG_BEG_IDENT) 
|| strstr(line, DRIVER_LOG_BEG_NUMRES) 
|| strstr(line, DRIVER_LOG_END_NUMRES) 
|| strstr(line, DRIVER_LOG_BEG_MEMRES) 
|| strstr(line, DRIVER_LOG_END_MEMRES)
|| strstr(line, DRIVER_LOG_BEG_MSGRES) || strstr(line, DRIVER_LOG_END_MSGRES) || strstr(line, DRIVER_LOG_BEG_MKDIAGRES) || strstr(line, DRIVER_LOG_END_MKDIAGRES) || strstr(line, DRIVER_LOG_BEG_CTMCGENRES) || strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
    return 1;
} else if (line[0] && line[0] != '\n') {
    if (vectident[0]) strcat(vectident, line);
    else strcpy(vectident, line);
} break;
} case 2:
    if (strstr(line, DRIVER_LOG_END_NUMRES)) {
        st = 0;
    } else if (strstr(line, DRIVER_LOG_BEG_IDENT) || strstr(line, DRIVER_LOG_END_IDENT) || strstr(line, DRIVER_LOG_BEG_NUMRES) || strstr(line, DRIVER_LOG_END_NUMRES) || strstr(line, DRIVER_LOG_BEG_MEMRES) || strstr(line, DRIVER_LOG_END_MEMRES) || strstr(line, DRIVER_LOG_BEG_MSGRES) || strstr(line, DRIVER_LOG_END_MSGRES) || strstr(line, DRIVER_LOG_BEG_MKDIAGRES) || strstr(line, DRIVER_LOG_END_MKDIAGRES) || strstr(line, DRIVER_LOG_BEG_CTMCGENRES) || strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
        return 1;
    } else if (strstr(line, FAIL_MSG)) {
        +nummeth_err;
    } break;
} case 3:
    if (strstr(line, DRIVER_LOG_END_MEMRES)) {
        st = 0;
    } else if (strstr(line, DRIVER_LOG_BEG_IDENT) || strstr(line, DRIVER_LOG_END_IDENT) || strstr(line, DRIVER_LOG_BEG_NUMRES) || strstr(line, DRIVER_LOG_END_NUMRES) || strstr(line, DRIVER_LOG_BEG_MEMRES) || strstr(line, DRIVER_LOG_END_MEMRES) || strstr(line, DRIVER_LOG_BEG_MSGRES) || strstr(line, DRIVER_LOG_END_MSGRES) || strstr(line, DRIVER_LOG_BEG_MKDIAGRES) || strstr(line, DRIVER_LOG_END_MKDIAGRES) || strstr(line, DRIVER_LOG_BEG_CTMCGENRES) || strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
        return 1;
    } else {
        memtest_done = 1;
        if (!linuxbox) {
            if ((s = strstr(line, VALGRIND_ERR_REPORT_MSG))) {
                sscanf(s+strlen(VALGRIND_ERR_REPORT_MSG), "%d", &mem_access_err);
            } else if ((s = strstr(line, VALGRIND_NONFREEDMEM_MSG))) {
                sscanf(s+strlen(VALGRIND_NONFREEDMEM_MSG), "%d", &nonfree);
                if (nonfree > 0) +num_leak_err;
            } else {
                if (strstr(line, BCHECK_BAF)) +mem_access_err;
                else if (strstr(line, BCHECK_DUF)) +mem_access_err;
                else if (strstr(line, BCHECK_MAF)) +mem_access_err;
                else if (strstr(line, BCHECK_MAR)) +mem_access_err;
                else if (strstr(line, BCHECK_MAW)) +mem_access_err;
                else if (strstr(line, BCHECK_OOM)) +mem_access_err;
                else if (strstr(line, BCHECK_RUA)) +mem_access_err;
                else if (strstr(line, BCHECK_RUI1)) +mem_access_err;
                else if (strstr(line, BCHECK_WRO)) +mem_access_err;
            }
        }
    }
}
else if (strstr(line, BCHECK_WUA)) ++mem_access_err;
else if (strstr(line, BCHECK_AIB)) ++mem_leak_err;
else if (strstr(line, BCHECK_AIR)) ++mem_leak_err;
else if (strstr(line, BCHECK_MEL)) ++mem_leak_err;
}
break;
case 4:
if (strstr(line, DRIVER_LOG_END_MSGRES)) {
    st = 0;
} else if (strstr(line, DRIVER_LOG_BEG_IDENT)
    || strstr(line, DRIVER_LOG_END_IDENT)
    || strstr(line, DRIVER_LOG_BEG_NUMRES)
    || strstr(line, DRIVER_LOG_END_NUMRES)
    || strstr(line, DRIVER_LOG_BEG_MEMRES)
    || strstr(line, DRIVER_LOG_END_MEMRES)
    || strstr(line, DRIVER_LOG_BEG_MSGRES)
    || strstr(line, DRIVER_LOG_END_MSGRES)
    || strstr(line, DRIVER_LOG_BEG_MODGENRES)
    || strstr(line, DRIVER_LOG_END_MODGENRES)
    || strstr(line, DRIVER_LOG_BEG_CTMCGENRES)
    || strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
    return 1;
} else if (strstr(line, FAIL_MSG)) {
    ++msg_err;
}
break;
case 5:
if (strstr(line, DRIVER_LOG_END_MODGENRES)) {
    st = 0;
} else if (strstr(line, DRIVER_LOG_BEG_IDENT)
    || strstr(line, DRIVER_LOG_END_IDENT)
    || strstr(line, DRIVER_LOG_BEG_NUMRES)
    || strstr(line, DRIVER_LOG_END_NUMRES)
    || strstr(line, DRIVER_LOG_BEG_MEMRES)
    || strstr(line, DRIVER_LOG_END_MEMRES)
    || strstr(line, DRIVER_LOG_BEG_MSGRES)
    || strstr(line, DRIVER_LOG_END_MSGRES)
    || strstr(line, DRIVER_LOG_BEG_MODGENRES)
    || strstr(line, DRIVER_LOG_END_MODGENRES)
    || strstr(line, DRIVER_LOG_BEG_CTMCGENRES)
    || strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
    return 1;
} else if (strstr(line, FAIL_MSG)) {
    ++modgen_err;
}
break;
case 6:
if (strstr(line, DRIVER_LOG_END_CTMCGENRES)) {
    st = 0;
} else if (strstr(line, DRIVER_LOG_BEG_IDENT)
    || strstr(line, DRIVER_LOG_END_IDENT)
    || strstr(line, DRIVER_LOG_BEG_NUMRES)
    || strstr(line, DRIVER_LOG_END_NUMRES)
    || strstr(line, DRIVER_LOG_BEG_MEMRES)
    || strstr(line, DRIVER_LOG_END_MEMRES)
    || strstr(line, DRIVER_LOG_BEG_MSGRES)
    || strstr(line, DRIVER_LOG_END_MSGRES)
    || strstr(line, DRIVER_LOG_BEG_MODGENRES)
    || strstr(line, DRIVER_LOG_END_MODGENRES)
    || strstr(line, DRIVER_LOG_BEG_CTMCGENRES)) {
    return 1;
} else {
    ctmcgentest_done = 1;
    if (strstr(line, FAIL_MSG)) ++ctmcgen_err;
}
break;
} /* switch (st) */
}
if (vectident[0]) {
    /* output last vector results */

++n_vect;
(void) fprintf(stdout, "%s\n", vectident);
if (nummeth_err
 || mem_leak_err
 || mem_access_err
 || msg_err
 || modgen_err
 || ctmcgen_err) {
    /* some error has been found */
++n_fVect;
(void) fprintf(stdout, "FAIL_MSG".\n\n";)
if (nummeth_err) {
    (void) fprintf(stdout, "Numerical results: found "
"%d errors\n", nummeth_err);
    nummeth_err = 0;
}
if (mem_access_err) {
    (void) fprintf(stdout, "Memory management: found "
"%d access errors\n", mem_access_err);
    mem_access_err = 0;
}
if (mem_leak_err) {
    (void) fprintf(stdout, "Memory management: found "
"%d leaks\n", mem_leak_err);
    mem_leak_err = 0;
}
if (msg_err) {
    (void) fprintf(stdout, "Error and warning messages: found "
"%d errors\n", msg_err);
    msg_err = 0;
}
if (modgen_err) {
    (void) fprintf(stdout, "Model generation: found "
"%d errors\n", modgen_err);
    modgen_err = 0;
}
if (ctmcgen_err) {
    (void) fprintf(stdout, "CTMC generation: found "
"%d errors\n", ctmcgen_err);
    ctmcgen_err = 0;
}
} else {
    /* no error has been found */
    (void) fprintf(stdout, "PASS_MSG".\n\n";)
}
vectident[0] = '\0';
}
if (n_vect == 0) return 1;

/* get model info */
if (read_mod_set(MOD_LIST_FILE) != 0) return 1;
if (mod_info(modname, &n_exp_vect, &n_exp_vect_pass,
    &n_exp_vect_fail) != 0) return 1;
free_modset();
if (n_vect != n_exp_vect) {
    /* shouldn't happen */
    (void) fprintf(stderr, "lookfor_errors(): the number of "
        "vectors of model '\%s' does not match up.\n",
    modname);
    exit(EXIT_FAILURE);
}
/* output summary */
(void) fprintf(stdout, "\nSUMMARY (memory test: %s, CTMC test: %s): "
"%s: %d/%d " PASS_MSG", %d/%d "FAIL_MSG" => ",
(memtest_done? "yes": "no"),
(ctmcgtest_done? "yes": "no"),
modname,
n_vect-n_fVect,
n_vect,
  n_f_vect,
  n_vect);
if (n_vect-n_f_vect == n_exp_vect_pass
   && n_f_vect == n_exp_vect_fail) {
  (void) fprintf(stdout, AGREEMENT_MSG"\n");
} else {
  (void) fprintf(stdout, DISAGREEMENT_MSG"\n");
} if (fclose(fp) != 0) return 1;
else return 0;