METFAC-2.1 User's Guide

Víctor Suñé and Juan A. Carrasco

Last Revision: July 7, 2021

Contents

Introduction

METFAC-2.1 is a tool for the specification and solution of rewarded, finite continuous-time Markov chains (CTMCs) with a reward structure including reward rates associated with states. (Appendix F provides a brief tutorial on rewarded finite CTMCs.) To install and use the tool, the programming environment must provide:

1. A C compiler compliant with the ISO/IEC 9899:1999 standard (C99) with enabled support for the IEC 60559:1989 floating-point standard (also designated as ANSI/IEEE 754-1985).

2. A C-shell (`csh`).

3. The utilities `ar` to create and modify archives, `ranlib` to generate indices to archives, `nm` to list symbols from object files, and `grep` to search for a pattern in a file.

4. The functions `getrusage` to measure CPU times, `sigemptyset` and `sigaction` to handle signals, and `timer_create`, `timer_settime`, and `timer_delete` to handle timers, all with the syntax and semantics described in the POSIX.1-2001 standard (IEEE Std 1003.1-2001).

Main features of the tool are:

1. Flexible and easy-to-use parametric model specification through a language based on production rules that allows the use of quite arbitrary C-like expressions and external C functions.

2. Fast generation of very large rewarded CTMC models.

3. Well tested (almost 100% coverage of reachable code).

4. CPU time limits controlled by operating system signals to avoid the tool to hung up.

5. Computation using numerical methods and estimation using robust simulation based on estimators which are guaranteed to reduce the variance of the estimators of naive simulation, which is always finite, of seven reward measures in almost their full generality.

6. Incorporation of numerical methods targeted at the computation of some reward measures or bounds for them for classes of finite CTMC models including both exact and bounding failure/repair models of fault-tolerant systems with exponential failure and repair time distributions and repair in every state with failed components and, perhaps, that the structure function of the modeled system be increasing, with component failure rates much smaller than component repair rates.

This document describes and explains how to use METFAC-2.1. Section 1 is a short tutorial introduction to the usage of the tool. Section 2 gives a detailed description of the model specification language based on production rules supported by the tool. Section 3 describes the mechanisms provided by the tool for model debugging. Section 4 defines the seven reward measures implemented in METFAC-2.1, which can be computed using numerical methods and estimated using simulation. For each measure, the numerical and simulation methods offered by the tool are listed, describing in detail the input the user has to provide. Some methods require the rewarded finite CTMC model to fulfill some conditions and these conditions are clearly identified. Appendix A describes how to install the tool. Appendix B includes a detailed description of the preprocessor of the tool that translates model specifications into model-specific C code. Appendix C describes the format of the textual CTMC description files optionally generated by the tool. Those files can be used for model debugging and to interface the tool with other tools. Appendix D describes the error and warning messages issued by the tool. Appendix E provides some mathematical justifications. Appendix F is a brief tutorial on rewarded CTMCs. Finally, Appendix G illustrates the capabilities of the tool using three sizable modeling examples of increasing complexity.

Regarding acknowledgement of the research that has led to the development of the tool, you can cite this User's Guide if you want to acknowledge the tool itself. If you want to acknowledge some particular numerical method incorporated into the tool and developed by us, we would appreciate your citing the reference for the method that is cited in this User's Guide.

Section 1

A Tutorial

In METFAC-2.1, rewarded finite CTMCs are defined by the user in textual form using a model specification language supported by the tool. In this section we illustrate through a small example how to: 1) define a rewarded CTMC using that model specification language, 2) compute using a numerical method one of the reward measures offered by the tool, 3) estimate using simulation one of the reward measures offered by the tool, and 4) generate a compact description of the rewarded CTMC that could possibly be used as input for other tools. We will also take advantage of the example to make some general comments on features of the tool.

The example is a CTMC reliability model of the fault-tolerant system whose block diagram is shown in Figure 1.1. The fault-tolerant system includes five processing modules (PM) and a restoring subsystem implementing fault detection, fault identification, and system reconfiguration. When no processing module is failed, three of the five processing modules are active working in a triple modular redundancy (TMR) configuration and the remaining two processing modules are acting as spares. The three active modules working in a TMR configuration execute the same task and vote their results. Fault detection and fault identification is based on the results issued by the active modules. If an active processing module fails and some spare is available, the restoring subsystem isolates the faulty module and switches in an unfailed spare. When no unfailed spare is available, a fault in an active module is managed by degrading the operation of the system to a duplex mode in which the two active processing modules compare their results to achieve fault detection. Any fault in a processing module when the system is working in duplex mode causes a system failure. A fault in the restoring subsystem also causes a system failure. Active processing modules fail with rate λ_{M}; spare modules fail with rate $\theta\lambda_{\mathrm{M}}$, where θ, $0 \leq \theta < 1$ is a dormancy factor accounting for the fact that spare processing modules fail less often than active processing modules. The restoring subsystem fails with rate λ_{R}. Components do not fail when the system has failed. The system is initially in the state with no processing module failed with probability 0.75 and in the state with one processing module failed with probability 0.25.

1.1 Model Specification

A model specification includes a model specification file named *name*`.spec`, where *name* is a string giving the model name and `.spec` is a mandatory suffix, and an optional C file named *name*`.c`. The file *name*`.spec` includes a description of the model following the syntax of a model

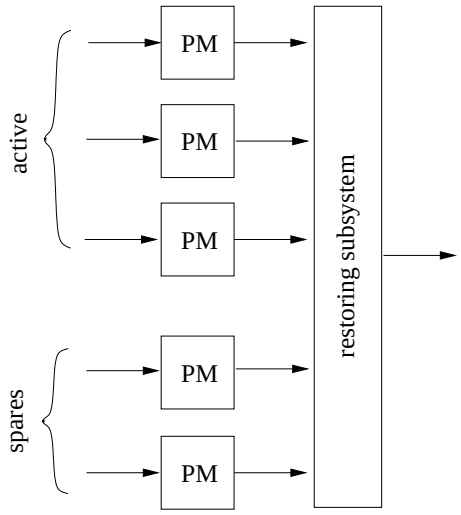

Figure 1.1: Block diagram of the example fault-tolerant system.

specification language based on production rules. The file *name*.`c` has to include definitions of all the external model-specific C functions used by the syntactic constructs of the model description included in the file *name*.`spec` and may include definitions of some C functions with predefined names and prototypes providing information required by some numerical methods or specifying the model checking to be performed. Those C functions with predefined names are summarized in Section 1.1.2. Besides the external model-specific C functions defined in the file *name*.`c`, the syntactic constructs of the model description contained in the file *name*.`spec` may use a subset of standard mathematical C functions. Supported standard mathematical C functions are listed in Table 2.1 on page 29.

1.1.1 The model specification file

The model specification file consists of a declarative part followed by an executive part.

The declarative part comprises, in no particular order, a declaration of state variables, an optional declaration of parameters, and an optional declaration of external model-specific functions.

The declaration of state variables consists of the keyword `state_variables` followed by a comma-separated list of state variable identifiers.[1]

The declaration of parameters consists of the keyword `parameters` followed by comma-separated lists of parameters, each list including the common type of the parameters in the list, which has to be `int` or `double`, followed by the parameter identifiers separated by commas.[2]

The declaration of external model-specific C functions consists of the keyword `external` followed by comma-separated lists of function prototypes. External model-specific C functions may have zero or more parameters and in the latter case all parameters have to be input parameters of type `int` or `double`. Each list of function prototypes includes the common return type of all functions of the list, which has to be `int` or `double`, followed by a comma separated list of

[1] In general, identifiers consist of sequences of alphabetic characters, digits and underscores, starting with an alphabetic character or an underscore, different from the reserved words of the language listed in Section 2.1, page 25.

[2] For instance, a legal declaration could be: `parameters int a, b, double c, int d, e`.

succinct function prototypes. Each succinct prototype includes the identifier of the function and a comma separated list enclosed between parentheses of the parameter types.[3]

The example model will be called TMR_hybrid and its description will include two files: the file `TMR_hybrid.spec` and the file `TMR_hybrid.c`. The state of the model can be described using three state variables: the number of active processing modules, `NA`, the number of unfailed spare processing modules, `NUS`, and the state of the restoring subsystem, `R` (1 if up, 0 if down). Parameters of the model will include the failure rates, λ_{M} and λ_{R}, of, respectively, active processing modules and the restoring subsystem and the dormancy factor θ. Parameters will all be of type `double` and will be called `LD_M`, `LD_R`, and `THETA`, respectively. The model description will make use of two external model-specific C functions. The first of them is called `sys_up` and has two `int` parameters, which are expected to be the values of the state variables `NA` and `R`. The function returns value 1 if the system is up and returns value 0 otherwise. The second external model-specific C function is called `init_prob` and has three `int` parameters, which are expected to be the values of the state variables `NA`, `NUS`, and `R`. The function returns the value of the initial probability of the state. Accordingly, the declarative part of `TMR_hybrid.spec` may be as follows:[4]

```
state_variables
NA,   /* number of active modules */
NUS,  /* number of unfailed spare modules */
R     /* yes (1) if the restoring subsystem is up; no (0) otherwise */

parameters
double
LD_M, /* failure rate of active modules */
LD_R, /* failure rate of the restoring subsystem */
THETA /* dormancy factor for spare modules */

external
int sys_up(int, int),           /* yes (1) if system up; no (0) otherwise */
double init_prob(int, int, int) /* initial probability */
```

Notice that, as in the C programming language, comments are enclosed by `/*` and `*/` and cannot be nested. The reserved words `yes` and `no` stand for, respectively, the `int` constants 1 and 0. They can be used to improve readability.

The executive part of the model specification file comprises, in no particular order, the specification of the state starting at which the CTMC has to be generated (we will refer to that state as the "start state"), an optional specification of the initial probability distribution, the specification of the reward rate structure, and the specification of the production rules.

The specification of the start state consists of the keyword `start_state` followed by a comma-separated list of assignments to state variables. There must be one and only one assignment for each state variable of the model. The right-hand side of each assignment must be an expression not involving state variables. The type of the result of such an expression must be `int`.[5]

The optional specification of the initial probability distribution consists of the keyword `initial_probability` followed by an expression. The type of the result of such an expres-

[3]For instance, a legal declaration could be: `external int f1(int), f2(int,double), double f3(int), f4(double)`.

[4]Again, state variables, parameters, and external model-specific functions can be declared in any order.

[5]The type of the result of an expression is determined as in C.

sion must be `double`. When computing a measure using numerical methods, states of the CTMC are assigned the initial probability that results from the evaluation of that expression. If the specification is absent, the start state is assigned an initial probability equal to 1 and the remaining states are assigned an initial probability equal to 0. When a measure is estimated using simulation, though, the specification of the initial probability distribution has *no effect* and the initial probability distribution *always* used is: Initial probability equal to 1 for the start state and equal to 0 for the remaining states.

The specification of the reward rate structure consists of the keyword `reward_rate` followed by an expression. The type of the result of such an expression must be `double`. States of the CTMC are assigned the reward rate that results from the evaluation of that expression.

In the example, we choose as start state the state with no processing module failed, set to 0.75 the initial probability of that state and to 0.25 the initial probability of the state with one processing module failed, and set to 1 and 0 the reward rate of the states in which the system is, respectively, up and down. Accordingly, the executive part of `TMR_hybrid.spec` may begin as follows:

```
start_state NA=3, NUS=2, R=yes

initial_probability init_prob(NA, NUS, R)

reward_rate (double) sys_up(NA, R)
```

The specification of the production rules consists of the keyword `production_rules` followed by a non-empty list of rules. Each production rule defines an action that, if enabled, changes, at a specified rate, the state of the CTMC by changing the values of some of the state variables. METFAC-2.1 offers two types of actions: Actions with a finite rate and actions with rate $+\infty$. In the remaining of this document, unless stated otherwise we will use the word "action" for actions with a finite rate, whereas actions with rate $+\infty$ will be referred to as *instantaneous* actions.[6]

An action consist of an optional condition specification, an optional identifier, a rate specification, and the specification of how state variables change their values. The optional condition specification consists of the keyword `if` followed by a C-like expression that may involve numerical constants, state variables, parameters, external model-specific functions, the standard mathematical C functions listed in Table 2.1 on page 29, and the operators `>`, `>=`, `<`, `<=`, `==`, `!=`, `+`, `-`, `*`, `/`, `&&`, `||`,`!`, `+` (unary), `-` (unary), `(int)`, and `(double)`. The type of the result of such an expression must be `int`. The action will be enabled if the condition expression yields a value distinct from zero. If no condition is specified, the action is always enabled.

Action identifiers are intended to enhance model readability. Valid action identifiers are sequences of alphabetic characters, digits and underscores, starting with an alphabetic character or an underscore, different from the keywords of the language listed in Section 2.1, page 25.

The rate specification consists of the keyword `with_rate` followed by a C-like expression that may involve numerical constants, state variables, parameters, external model-specific functions, the standard mathematical C functions listed in Table 2.1 on page 29, and the operators `>`, `>=`, `<`, `<=`, `==`, `!=`, `+`, `-`, `*`, `/`, `&&`, `||`,`!`, `+` (unary), `-` (unary), `(int)`, and `(double)`. Rate expressions are allowed to yield any positive result with type `double`.

[6] An instantaneous action can be thought of as an action that changes the values of some state variables infinitely fast, i.e., instantaneously or in zero time.

The specification of how state variables change their values can be simple or with responses. In the former case, the specification consists of the keyword `next_state` followed by a comma-separated list of assignments to state variables. The first production rule of the example is simple and models the fault of the restoring subsystem:

```
if sys_up(NA, R) action FAIL_RESTORE with_rate LD_R
   next_state R=no
```

The above production rule can be paraphrased as "if the system is up, with rate `LD_R` change to a state that differs from the current one in that the state variable `R` takes the value `no`." In general, there are three ways to specify how a state variable changes its value:

- By assigning using the operator `=` the result of evaluating an expression. Such a result must have type `int`.
- By increasing or decreasing by one using the `++` or `--` operator the current value of the state variable.
- By adding to or subtracting from the current value of the state variable the result of evaluating an expression, multiplying the current value of the state variable by the result of evaluating an expression, or dividing the current value of the state variable by the result of evaluating an expression. As in C, this is achieved by using the operators `+=`, `-=`, `*=`, and `/=` followed by the expression. In all cases, the type of the result of the expression must be `int`.

The second production rule of the example is also simple and models the fault of a spare processing module:

```
if sys_up(NA, R) && NUS>0 action FAIL_SPARE with_rate NUS*LD_M*THETA
   next_state NUS-- /* alternatives: NUS=NUS-1, NUS-=1 */
```

The third production rule of the example has two responses and models the fault of an active module. The use of responses is required because the state variables have to change in a different way depending on whether the system has unfailed spare processing modules or not:

```
if sys_up(NA, R) action FAIL_ACTIVE with_rate NA*LD_M
   if NUS>0 response CONTINUE
      next_state NUS--
   if NUS==0 response DUPLEX
      next_state NA--
end
```

The previous production rule can be paraphrased as "if the system is up, with rate `NA*LD_M` change to a state which, if there are unfailed spare processing modules, differs from the current one in that the state variable `NUS` is one less and, if there are not unfailed spare processing modules, differs from the current one in that `NA` is one less."

In general, production rules with responses may have one or more responses and must end with the keyword `end`. Each response consists of an optional condition specification, an optional identifier, an optional probability specification, and the specification of how state variables change their values. The optional condition specification consists of the keyword `if` followed by a C-like

expression as the one described for the condition specification of actions. The response will be enabled if the condition expression yields a value distinct from zero and will be always enabled if no condition is specified. Valid response identifiers are as valid action identifiers. The optional probability expression consists of the keyword `with_prob` followed by a C-like expression as the one described for the rate specification of actions. Probability expressions must yield a result with type `double` that is > 0 and ≤ 1. In addition, if the sum of the probabilities of the enabled responses of an enabled action is larger than 1 by more than 1,000 times the "epsilon" constant of the underlying hardware,[7] a warning will be issued. If no probability expression is given, the probability of the response has a default value of 1. The specification of how state variables change their values consists of the keyword `next_state` followed by a comma-separated list of assignments to state variables. The rate at which the specified changes in the state variables occur is the product of the value yielded by the rate expression of the action and the value yielded by the probability expression of the response or 1 if no probability expression is given.

Instantaneous actions have the same syntactical structure as actions except that:

1. The condition specification is mandatory.
2. The rate specification consists of the keyword `with_rate` followed by the keyword `infinity`.
3. The specification of how state variables change their values cannot have responses, i.e. it must consist of the keyword `next_state` followed by a comma-separated list of assignments to state variables.

The set of instantaneous actions of a model specification must fulfill the following two conditions:

- No instantaneous action is enabled in the start state.
- At most one instantaneous action is enabled in any state different from the start state.

Semantically, for an instantaneous action, the rate at which the specified changes in the state variables occur is $+\infty$.

When an instantaneous action is enabled in a state, with probability 1 the CTMC will spend zero time in it (we say that the state is *vanishing* or *instantaneous*), jumping instantaneously to the state to which the instantaneous action leads. Therefore, vanishing states have no impact on the behavior of the CTMC and can thus be deleted. In METFAC-2.1, vanishing states are always deleted before computing a measure using a numerical method. This fact makes instantaneous actions, if used judiciously, a useful device to reduce the number of states of the CTMC. In the example, no production rule will be enabled in a state in which the system is down, making such a state absorbing. Besides, there would be several of those absorbing states, each with the same reward rate. Therefore, it makes sense to "merge" them into a single absorbing state described by, say, the following values of the state variables: `NA=0`, `NUS=0`, `R=no`. This can be easily accomplished by means of an instantaneous action:

```
if !sys_up(NA, R) && !(NA==0 && NUS==0 && R==no) action MERGE with_rate infinity
next_state NA=0, NUS=0, R=no
```

[7] The "epsilon" constant is the difference between the smallest exactly representable number greater than 1 and 1.

This completes the description of TMR_hybrid.spec. Its contents is as follows:

```
state_variables
NA,   /* number of active modules */
NUS,  /* number of unfailed spare modules */
R     /* yes (1) if the restoring subsystem is up; no (0) otherwise */

parameters
double
LD_M, /* failure rate of active modules */
LD_R, /* failure rate of the restoring subsystem */
THETA /* dormancy factor for spare modules */

external
int sys_up(int, int),           /* yes (1) if system up; no (0) otherwise */
double init_prob(int, int, int) /* initial probability */

start_state NA=3, NUS=2, R=yes

initial_probability init_prob(NA, NUS, R)

reward_rate (double) sys_up(NA, R)

production_rules
if sys_up(NA, R) action FAIL_RESTORE with_rate LD_R
   next_state R=no
if sys_up(NA, R) && NUS>0 action FAIL_SPARE with_rate NUS*LD_M*THETA
   next_state NUS-- /* alternatives: NUS=NUS-1, NUS-=1 */
if sys_up(NA, R) action FAIL_ACTIVE with_rate NA*LD_M
   if NUS>0 response CONTINUE
      next_state NUS--
   if NUS==0 response DUPLEX
      next_state NA--
end
if !sys_up(NA, R) && !(NA==0 && NUS==0 && R==no) action MERGE with_rate infinity
next_state NA=0, NUS=0, R=no
```

1.1.2 The optional C file

The purpose of the file *name*.c is to define the external model-specific C functions used by the syntactic constructs of the model description included in the file *name*.spec and, if necessary, to define a set of C functions with predefined names and prototypes providing information required by some numerical methods or specifying the model checking to be performed. There are four such functions with predefined names and prototypes:

- block, pivot, regstat

  Used to specify information required by some numerical methods. (See Section 4.2, pages 47, 49, and Section 4.1, page 41.)

- check_state.

  Used for model checking. (See Section 3, page 31.)

The file TMR_hybrid.c includes the definition of two external model-specific C functions: sys_up and init_prob. The first of them has to return the value 1 if the system is up and the value 0 otherwise; the second one has to return the value 0.75 for the state without failed components, the value 0.25 for the state in which one processing module is failed, and the value 0 for the remaining states. Accordingly, the contents of the file TMR_hybrid.c is as follows:

```
#include "TMR_hybrid.h"

int
sys_up(int n_modules, int restoring_ok)
{
  if (n_modules > 1 && restoring_ok)
    return 1;
  else
    return 0;
}


double
init_prob(int n_modules, int n_spare, int restoring_ok)
{
  if (n_modules == 3 && n_spare == 2 && restoring_ok)
    return 0.75;
  else if (n_modules == 3 && n_spare == 1 && restoring_ok)
    return 0.25;
  else
    return 0.0;
}
```

Note the inclusion of the file TMR_hybrid.h. That file is generated automatically when the model is successfully compiled and is included so that the definitions of the external model-specific C functions are checked against the succinct prototypes of those functions declared in the file TMR_hybrid.spec. Inclusion of the file TMR_hybrid.h may have another important purpose: If that file is included and the statement "DECLARE_SYMBOLS;" is included in the declarative part of any function with predefined name and prototype, the body of that functions can use the names of state variables and the names of parameters, improving the readability of the definition of the function. (See, for instance, Section 3, page 31.)

1.2 Model Compilation

Model compilation is done with the utility m2build, which takes as input the model name . Thus, to compile the TMR_hybrid model we would type

```
m2build TMR_hybrid
```

getting

```
[preproc inform] Completed with 0 error(s) and 0 warning(s) on line 34 of file
'TMR_hybrid.spec'.
```

```
[m2build warning] Using default block() function.
[m2build warning] Using default check_state() function.
[m2build warning] Using default pivot() function.
[m2build warning] Using default regstat() function.

[m2build inform] 'TMR_hybrid.exe' has been created.
```

Compilation in METFAC-2.1 is a two-stage process. In the first stage, the file `TMR_hybrid.spec` is analyzed and translated into model-specific C code encapsulated in a file called `TMR_hybrid_prep.c`. (In general, a model specification file named *name*`.spec` is analyzed and translated into model-specific C code encapsulated in a file called *name*`_prep.c`.) Such an analysis is performed by the preprocessing module `preproc` of the tool. That module can also be used independently (see Appendix B, page 83).

If any error were encountered in `TMR_hybrid.spec`, then `preproc` would issue to the standard error stream (usually the computer terminal) one or more error messages including

```
[preproc error ...] File 'TMR_hybrid.spec'
```

(the "`...`" stands for the error number) followed by a short description of the error, the user would get on the standard output stream (usually the computer terminal) the messages

```
[preproc inform] No code generated.

[preproc inform] Completed with ... error(s) and ... warning(s) on line 34 of file
'TMR_hybrid.spec'.
```

(the "`...`" stand for the numbers of errors and warnings), and the compilation process would be aborted. The preprocessing module can also issue warning messages (see Appendix B, page 84). For TMR_hybrid, such messages would include

```
[preproc warning ...] file 'TMR_hybrid.spec'
```

(the '`...`' stands for the warning number) followed by a short description of the warning. Warning messages can (but should not) be ignored. Thus, the message

```
[preproc inform] Completed with 0 error(s) and 0 warning(s) on line 34 of file
'TMR_hybrid.spec'.
```

we get when compiling TMR_hybrid informs us that the file `TMR_hybrid.spec` has been successfully analyzed without errors and without warnings and that the file `TMR_hybrid_prep.c` has been generated.

The second stage of the compilation process is only activated if the first stage has been successful (no errors were encountered during the analysis of the model specification file). In that stage, the model-specific C code encapsulated in the file `TMR_hybrid_prep.c` and the file `TMR_hybrid.c` are compiled and linked to the model-independent core of the tool using a C compiler and several system utilities. The four warning messages

```
[m2build warning] Using default block() function.
[m2build warning] Using default check_state() function.
[m2build warning] Using default pivot() function.
[m2build warning] Using default regstat() function.
```

we get when compiling TMR_hybrid simply inform us that none of the functions with predefined names have been defined in `TMR_hybrid.c` and that, therefore, the respective default versions have been used. The last message

```
[m2build inform] 'TMR_hybrid.exe' has been created.
```

informs us that the executable file `TMR_hybrid.exe` has been successfully created. All messages generated by the C compiler and the system utilities are sent to a file named `TMR_hybrid_build.log`. (In general, if the model specification file is named *name*`.spec`, the executable file will be called *name*`.exe` and the file to which all messages generated by the C compiler and system utilities are sent will be called *name*`_build.log`.) That file is generated to help the user fix errors in the file `TMR_hybrid.c` causing a model compilation failure. Thus, assuming, for instance, that we mistyped `n_active` as `n_activ` in the first line of the body of the function `sys_up` (see page 14), we would get the messages:

```
[preproc inform] Completed with 0 error(s) and 0 warning(s) on line 34 of file
'TMR_hybrid.spec'.


[m2build warning] Using default block() function.
[m2build warning] Using default check_state() function.
[m2build warning] Using default pivot() function.
[m2build warning] Using default regstat() function.
[m2build error] Compilation failed.
(See file 'TMR_hybrid_build.log' for details.)
```

and the corresponding error messages of the compiler would have been written to the file `TMR_hybrid_build.log`.

1.3 Model Execution

Once successfully compiled, a model is executed by running the corresponding executable file. Thus, we can execute TMR_hybrid by typing

```
TMR_hybrid.exe
```

If the model has parameters, the executable starts by prompting for numerical values for them. Under the assignments $\lambda_{\mathrm{M}} = 10^{-4}$ h^{-1}, $\lambda_{\mathrm{R}} = 10^{-6}$ h^{-1}, and $\theta = 0.2$, the interaction would look like:

```
                 Model parameters
                 ----------------


LD_M: 1e-4
LD_R: 1e-6
THETA: 0.2
```

We are prompted next to select the task to be performed by the tool. METFAC-2.1 offers four tasks:

1. Computation of a measure using numerical methods.

2. Estimation of a measure using simulation.

3. Generation of a verbose description of a rewarded CTMC.

4. Generation of a compact description of a rewarded CTMC.

Computing or estimating a measure are the main tasks the tool is intended for; generating a verbose description of the rewarded CTMC is intended for model debugging, an issue covered in Section 3; generating a compact description of the rewarded CTMC makes it possible to interface METFAC-2.1 with other tools. We note that the compact description is always generated after deleting the vanishing states of the model.

After performing either of these tasks, the tool will create a file called `TMR_hybrid.log`. (For a model specification file named *name*`.spec`, that file would be called *name*`.log`.) That file will contain: 1) the date and time of the day on which the model was compiled; 2) the date and time of the day on which the model was executed; 3) if the model had parameters, the values given to them; 4) a brief summary of the characteristics of the rewarded CTMC, 5) if the selected task was to compute or estimate a measure, the computed or estimated value(s) of the measure along with some statistics about how the numerical method(s) or the simulation performed; and 6) if the selected task was to generate a verbose description of the rewarded CTMC, that description in textual form. (Section C.1 of Appendix C describes the contents of the file *name*`.log` in that case.) There are two other files than can possibly be created:

1. `TMR_hybrid.rng`

   This file, which for a model named *name* will be called *name*`.rng`, will be created after estimating a measure using simulation. It contains the state of the pseudo-random number generator upon termination of the simulation and can be used to restore the state of the generator in a new simulation (see page 21). This means that in the new simulation, the pseudo-random number generator will continue the sequence of pseudo-random numbers that was being generated when the file was created.

2. `TMR_hybrid.ctmc`

   This file, which for a model named *name* will be called *name*`.ctmc`, will be created after generating a compact description of the rewarded CTMC with vanishing states deleted and will contain the description in textual form. Section C.2 provides a detailed description of the contents of this file.

Assuming we want to compute a measure using numerical methods, the interaction would look like:

```
          Tasks
          -----

1. Compute a measure using numerical methods.
2. Estimate a measure using simulation.
3. Generate a verbose description of the
   rewarded CTMC.
4. Generate a compact description of the
   rewarded CTMC.
```

```
Option (1-4)? 1
```

After that, we would be asked to select the measure to be computed, the numerical method to be used, and the values of the control parameters associated with that numerical method. (See Section 4 for a description of the numerical methods and measures offered by the tool.) Assuming that we chose the measure "`Expected Transient Reward Rate`", selected the numerical method "`Standard Randomization with control of the Relative Error`", and set the allowed relative error and the allowed CPU time to, respectively, 10^{-6} and 10 s,[8] the interaction would look like:

```
                 Measures
                 --------

           1. Expected Transient Reward Rate (ETRR(t)).
           2. Expected Steady-State Reward Rate (ESSRR).
           3. Expected Averaged Reward Rate (EARR(t)).
           4. Cumulative Reward Complementary
              Distribution (CRCD(t,s)).
           5. Interval Availability Complementary
              Distribution (IAVCD(t,p)).
           6. Expected Cumulative Reward Till Exit of a
              subset of states (ECRTE).
           7. Cumulative Reward Distribution Till Exit
              of a subset of states (CRDTE(s)).

Option (1-7)? 1

                 Numerical methods
                 -----------------

           1. Standard Randomization.
           2. Standard Randomization with control of the
              Relative Error.
           3. Randomization with Stationarity Detection.
           4. Randomization with Quasistationarity
              Detection.
           5. Regenerative Randomization.
           6. Regenerative Randomization with Laplace
              Transforms.
           7. Bounding Regenerative Randomization.
           8. Explicit Runge-Kutta ODE Solver.
           9. Implicit Runge-Kutta ODE Solver.

Option (1-9)? 2
Allowed relative error (> 0)? 1e-6
Allowed CPU time in s (> 0)? 10
```

After that, we would be asked to define the set of time abscissae t at which the measure has to be computed. In general, sets of abscissae are defined as follows. First, we give the number, n, of abscissae. If $n = 1$, we give the value of the single abscissa; if $n > 1$, we specify the set of values

[8] The allowed CPU must be always given as an integer number of seconds.

using either a linearly or logarithmically scaled grid of abscissae or an arbitrary, not necessarily sorted, list of abscissae. In the first case (linearly or logarithmically scaled grid), we give the smallest and largest abscissae. Denoting them by, respectively, a_{m} and a_{M}, the measure will be computed at the abscissae $a_{\mathrm{m}}+(i-1)(a_{\mathrm{M}}-a_{\mathrm{m}})/(n-1)$, $i=1,\ldots,n$ if the grid is linearly scaled and at the abscissae $a_{\mathrm{m}}\sqrt[n-1]{(a_{\mathrm{M}}/a_{\mathrm{m}})^{(i-1)}}$, $i=1,\ldots,n$ if the grid is logarithmically scaled. In the second case (list), the values making up the list are given separated by commas. To illustrate, the interaction would look like:

```
Number of abscissae (>= 1)? 1
Abscissa (>= 0)? 100
```

if we selected to compute the measure at the single abscissa $t=100$, would look like:

```
Number of abscissae (>= 1)? 3
Lin. scaled grid, log. scaled grid or list of values (n/N/g/G/s/S)? n
Initial abscissa (>= 0)? 100
Final abscissa (> initial abscissa)? 1000
```

if we selected to compute the measure at a grid of three linearly scaled abscissae with minimum value 100 and maximum value 1,000, and would look like:

```
Number of abscissae (>= 1)? 3
Lin. scaled grid, log. scaled grid or list of values (n/N/g/G/s/S)? s
Comma-separated list of non-negative values? 100, 750, 1e3
```

if we selected to compute the measure at the list of abscissae $t=100$, 750, 1,000.

Finally, we would be asked whether we want a verbose or a concise output:

```
Verbose output (y/Y/n/N)?
```

Assuming verbose output were selected and that a grid of three linearly scaled abscissae with minimum value 100 and maximum value 10,000 were specified, the tool would write to the standard output:[9]

```
Generation:

CTMC characteristics:
# states=5
# classes of states=5
(# transient=4, # recurrent=1)
# transition rates=7

Spent time in s (user, system, total)=0.000000E+00, 0.000000E+00, 0.000000E+00

Solution:

# steps=1.600000E+01

Spent time in s (user, system, total)=0.000000E+00, 0.000000E+00, 0.000000E+00

Results:
```

[9] CPU times may differ from those given here.

```
t=1.000000000000000E+02 ETRR(t)=9.998992016713486E-01
t=5.050000000000000E+03 ETRR(t)=9.145616142776791E-01
t=1.000000000000000E+04 ETRR(t)=6.382929169534498E-01

Total time in s (user, system, total)=0.000000E+00, 4.000000E-03, 4.000000E-03

[metfac2 warning 1] Vanishing states have been deleted.
There were 5 vanishing states.
```

whereas if concise output were selected, the tool would write to the standard output:

```
Results:

t=1.000000000000000E+02 ETRR(t)=9.998992016713486E-01
t=5.050000000000000E+03 ETRR(t)=9.145616142776791E-01
t=1.000000000000000E+04 ETRR(t)=6.382929169534498E-01

Total time in s (user, system, total)=4.000000E-03, 0.000000E+00, 4.000000E-03

[metfac2 warning 1] Some states have been deleted.
There were 5 vanishing states.
```

In either case, those outputs together with the date and time of compilation and execution as well as the values of the model parameters would also be written to the file `TMR_hybrid.log`.

If we wanted to estimate the measure using simulation, we would execute TMR_hybrid, introduce the numerical values of the model parameters, choose the task "`Estimate a measure using simulation`", choose the measure "`Expected Transient Reward Rate`", and select the only available simulation method for that measure. The interaction would look like:

```
                Model parameters
                ----------------

LD_M: 1e-4
LD_R: 1e-6
THETA: 0.2

                Tasks
                -----

          1. Compute a measure using numerical methods.
          2. Estimate a measure using simulation.
          3. Generate a verbose description of the
             rewarded CTMC.
          4. Generate a compact description of the
             rewarded CTMC.

Option (1-4)? 2

                Measures
                --------

          1. Expected Transient Reward Rate (ETRR(t)).
          2. Expected Steady-State Reward Rate (ESSRR).
          3. Expected Averaged Reward Rate (EARR(t)).
```

```
          4. Cumulative Reward Complementary
             Distribution (CRCD(t,s)).
          5. Interval Availability Complementary
             Distribution (IAVCD(t,p)).
          6. Expected Cumulative Reward Till Exit of a
             subset of states (ECRTE).
          7. Cumulative Reward Distribution Till Exit
             of a subset of states (CRDTE(s)).

Option (1-7)? 1

                Simulation methods
                ------------------

          1. Independent Realizations with Forced
             Transitions of the Reward Rate.

Option (1-1)? 1
```

After that, we would introduce the values of the control parameters associated with that method. Those values are: The confidence level as a decimal value, the allowed relative half-width of the confidence interval, whether the pseudo-random number generator is to be initialized using a seed or else the generator's state is to be restored from a file generated in a previous simulation (see page 17), the value of the seed or the name of the file, the allowed number of forced transitions, the minimum number of realizations with positive estimator, and the allowed CPU time in seconds. Assuming that we set the confidence level and relative half-width to, respectively, 0.99 and 0.01, chose starting the pseudo-random number generator using the default seed, set both the allowed number of forced transitions and the minimum number of realizations with positive estimator to 10,000, and set the allowed CPU time to 10 s, the interaction would look like:

```
Confidence level (> 0, < 1)? 0.99
Relative half-width of the confidence interval (> 0)? 0.01
Pseudo-random number generator: Initialize using seed or restore state from file (s/S/f/F)? s
Seed (>= 1; hit enter key to choose the default: 5489)?
Allowed number of forced transitions (>= 0)? 10000
Minimum number of realizations with positive estimator (>= 2)? 10000
Allowed CPU time in s (> 0)? 10
```

Finally, we would be asked to specify the time abscissa and select between verbose and concise output. Assuming we set the time abscissa to 10,000 and chose verbose output, the interaction would look like:

```
Abscissa (> 0)? 1e4

Verbose output (y/Y/n/N)? y
```

After performing the simulation, the tool would write to the standard output:[10]

```
Simulation:
```

[10]CPU times may differ from those given here.

```
Spent time in s (user, system, total)=6.000400E-02, 0.000000E+00, 6.000400E-02

Results:

t=1.000000000000000E+04 ETRR(t)=6.853991596638668E-01 [+/- 6.853027385660408E-03]

Total time in s (user, system, total)=6.800400E-02, 0.000000E+00, 6.800400E-02
```

Two important remarks are in order here. The first remark is that the results of a simulation are always given as

$$value_1 \text{ [+/- } value_2 \text{]},$$

where $value_1$ is the computed estimate and $value_2$ is the half-width of the computed confidence interval with confidence level the one given by the user. The second remark is that what we have computed for the TMR_hybrid example is an estimate for the measure "Expected Transient Reward Rate" at $t = 10,000$ with the following initial probability distribution: initial probability equal to 1 for the start state and equal to 0 for the remaining states, i.e., assuming that with probability 1 the system is initially in the state with no processing module failed. (We recall that when a measure is estimated using simulation, the specification of the initial probability distribution has no effect.) Therefore, if we wanted to estimate the measure at $t = 10,000$ assuming that the system is initially in the state with no processing module failed with probability 0.75 and in the state with one processing module failed with probability 0.25, we would have to:

1. Modify `TMR_hybrid.spec`, replacing "`start_state NA=3, NUS=2, R=yes`" by "`start_state NA=3, NUS=1, R=yes`", compile the model, and execute it with the same settings as before save that we would restore the state of the pseudo-random number generator using the file `TMR_hybrid.rng`. The result could be:

   ```
   Simulation:


   Spent time in s (user, system, total)=1.600100E-01, 4.000000E-03, 1.640100E-01

   Results:

   t=1.000000000000000E+04 ETRR(t)=5.043266506363613E-01 [+/- 5.040119544597909E-03]

   Total time in s (user, system, total)=1.640100E-01, 4.000000E-03, 1.680100E-01
   ```

2. Estimate the measure by adding the new estimate (`5.043266506363613E-01`) times 0.25 and the previous one (`6.853991596638668E-01`) times 0.75, getting 0.64. We note that this result is in accordance with the value of the measure at $t = 10,000$ obtained using a numerical method (see page 20).

Finally, if we wanted to generate a compact description of the rewarded CTMC with vanishing states deleted, we would execute TMR_hybrid, introduce the numerical values of the model parameters, and choose the task "`Generate a compact description of the rewarded CTMC`". The interaction would look like:

```
                Model parameters
                ----------------

LD_M: 1e-4
LD_R: 1e-6
THETA: 0.2

                Tasks
                -----

          1. Compute a measure using numerical methods.
          2. Estimate a measure using simulation.
          3. Generate a verbose description of the
             rewarded CTMC.
          4. Generate a compact description of the
             rewarded CTMC.

Option (1-4)? 4
```

The description would be written in textual form to the file `TMR_hybrid.ctmc`. (For a model named *name*, that file would be called *name*`.ctmc` —see Section C.2, page 99 for a detailed description of the contents of that file.)

Section 2

Model Specification Language

This section describes in detail the lexis, syntax, and semantics of the model specification language based on production rules that is supported by METFAC-2.1.

2.1 Lexis

Syntactic elements of the language include keywords, numerical constants, operators, identifiers, and delimiters. Comments are enclosed by `/*` and `*/` and cannot be nested. Syntactic elements can be separated by any number of white spaces, tabs, or newline characters. The language is case sensitive.

The keywords of the language are the following:

`action`	`int`	`start_state`
`double`	`next_state`	`state_variables`
`end`	`no`	`with_prob`
`external`	`parameters`	`with_rate`
`if`	`production_rules`	`yes`
`infinity`	`response`	
`initial_probability`	`reward_rate`	

Reserved words are the keywords of the language, words ending in an underscore, and the words:

`DECLARE_SYMBOLS`	`check_state`	`regstat`
`block`	`pivot`	`subset`

Valid numerical constants are decimal and floating-point constants given in the usual way (e.g. `15`, `-27`, `1.415`, `-1.27e-5`, `1.12E6`, etc). Moreover, the language supports the constants `yes` and `no`, standing for, respectively, the integer values 1 and 0.

The language supports the following subset of operators taken from the C programming language:

arithmetic assignment operators: `=`, `+=`, `-=`, `*=`, `/=`

relational operators: `>`, `>=`, `<`, `<=`, `==`, `!=`

binary arithmetic operators: `+`, `-`, `*`, `/`

logical operators: `&&`, `||`, `!`

increment operator: `++`

decrement operator: `--`

unary plus operator: `+`

unary minus operator: `-`

cast operator: `( )`

Precedence and associativity of the operators are as in C.

Identifiers consist of any sequence of letters, digits, and underscores beginning with a letter or an underscore. Identifiers may or may not have semantic value. Identifiers with semantic value are parameters, state variables, and model-specific function identifiers. Identifiers without semantic value are action and response identifiers. Identifiers with semantic value must be different from any reserved word. Identifiers without semantic value must be different from any keyword.

The delimiters of the language are parenthesis and commas.

2.2 Syntax and Semantics

The syntax of the language will be described in Extended BNF (EBNF). We recall that, in EBNF, the comma indicates concatenation, vertical bars | separate alternatives, square brackets [ ] indicate zero or one occurrence of what they surround, curly braces { } indicate zero or more occurrences of what they surround, curly braces followed by a plus sign { }+ indicate one or more occurrences of what they surround, parentheses ( ) group what they surround, and literal text is enclosed with double quotes like "this". For the sake of clarity, the keywords and operators of the language will be written in typewriter font like `this` and identifiers will be written in bold font like **this**. We give next an EBNF description of the model specification language up to the level of expression and model-specific succinct function prototype. The syntax of model-specific succinct function prototypes and expressions will be discussed afterwards.

spec-file = dec, exec;
dec = par-dec | sv-dec | ext-dec | (dec, par-dec) | (dec, sv-dec)
  | (dec, ext-dec);
exec = pr-spec | ss-spec | rr-spec | ip-spec | (exec, pr-spec) | (exec, ss-spec)
  | (exec, rr-spec) | (exec, ip-spec);
par-dec = "`parameters`",
  ("`int`" | "`double`"), **id**, {",", (**id** | (("`int`" | "`double`"), **id**))};
sv-dec = "`state_variables`",
  id, {",", **id**};
ext-dec = "`external`",
  ("`int`" | "`double`"), func-proto,

```
                    {",", (func-proto | (("int" | "double"), func-proto))};
pr-spec           = "production_rules",
                    {["if", exp], "action", [label], "with_rate", (exp | "infinity"),
                     "(next_state", chg-exp, {",", chg-exp})
                       | ({["if", exp], "response", [label], ["with_prob" exp],
                          "next_state" chg-exp, {",", chg-exp}}+,
                        "end"
                        )
                    }+;
ss-spec           = "start_state",
                    init-exp, {",", init-exp};
rr-spec           = "reward_rate",
                    exp;
ip-spec           = "initial_probability",
                    exp;
```

where

id	is an identifier with semantic value,
label	is an identifier without semantic value,
func-proto	is a model-specific succinct function prototype, and
exp, chg-exp, init-exp	are expressions.

The syntax in EBNF for a model-specific succinct function prototype is as follows:

```
func-proto        = id, "(", [("int" | "double"), {",", ("int" | "double")}]")";
```

Finally, the syntax in EBNF for expressions is as follows:

```
chg-exp           = sv-id, (post-op | (ass-op, exp));
init-exp          = sv-id, "=", restrict-exp;
post-op           = "++" | "--";
ass-op            = "=" | "+=" | "-=" | "*=" | "/=";
exp               = and-exp | (exp, or-op, and-exp);
restrict-exp      = restrict-and-exp | (restrict-exp, or-op, restrict-and-exp);
and-exp           = eq-exp | (and-exp, and-op, eq-exp);
restrict-and-exp  = restrict-eq-exp | (restrict-and-exp, and-op, restrict-eq-exp);
eq-exp            = rel-exp | (eq-exp, eq-op, rel-exp);
restrict-eq-exp   = restrict-rel-exp | (restrict-eq-exp, eq-op, restrict-rel-exp);
rel-exp           = arith-exp | (rel-exp, rel-op, arith-exp);
restrict-rel-exp  = restrict-arith-exp | (restrict-rel-exp, rel-op, restrict-arith-exp);
arith-exp         = mult-exp | (arith-exp, add-op, mult-exp);
restrict-arith-exp = restrict-mult-exp | (restrict-arith-exp, add-op, restrict-mult-exp);
mult-exp          = cast-exp | (mult-exp, mult-op, cast-exp);
```

restrict-mult-exp = restrict-cast-exp | (restrict-mult-exp, mult-op, restrict-cast-exp);
cast-exp = unary-exp | (“(“, type, “)”, cast-exp);
restrict-cast-exp = restrict-unary-exp | (“(“, type, “)”, restrict-cast-exp);
unary-exp = **sv-id** | **par-id**
| (**func-id**, [“(“[exp, {“,”, exp}]”)”])
| cntnd | (“(“, exp, “)”) | (unary-op, cast-exp);
restrict-unary-exp = **par-id**
| (**func-id**, [“(“[restrict-exp, {“,”, restrict-exp}]”)”]) |
cntnd | (“(“, restrict-exp, “)”) | (restrict-unary-op, restrict-cast-exp);
cntnd = “`yes`” | “`no`” | numcons;
type = “`int`” | “`double`”;
or-op = “`||`”;
and-op = “`&&`”;
eq-op = “`==`” | “`!=`”;
rel-op = “`<`” | “`<=`” | “`>`” | “`>=`”;
add-op = “`+`” | “`-`”;
mult-op = “`*`” | “`/`”;
unary-op = “`+`” | “`-`” | “`!`”;

where

sv-id	is a state variable identifier
par-id	is a parameter identifier
func-id	is the identifier of either a model-specific function declared within the `external` construct or a supported standard C function (see Table 2.1), and
numcons	is a valid numerical constant.

Semantic restrictions of the language not captured by the previous descriptions are the following:

- There can be at most one declaration of parameters or model-specific functions.
- There must be one and only one declaration of state variables.
- There must be one and only one specification of production rules.
- Instantaneous actions (those in which the keyword `with_rate` is followed by the keyword `infinity`) must have a condition and must not have responses.
- Each state variable may change its value at most once within the same `next_state` construct.
- There must be one and only one specification of the start state and such a specification must be such that each state variable is initialized once and only once.
- There must be one and only one specification of the reward rate structure.

Table 2.1: Supported standard C functions.

prototype	description
`double acos(double x)`	principal value of the arc cosine of `x`
`double asin(double x)`	principal value of the arc sine of `x`
`double atan(double x)`	principal value of the arc tangent of `x`
`double atan2(double x, double y)`	principal value of the arc tangent of `y/x`
`double ceil(double x)`	smaller integer non smaller than `x`
`double cos(double x)`	cosine of `x` (`x` measured in radians)
`double cosh(double x)`	hyperbolic cosine of `x`
`double exp(double x)`	exponential function of `x`
`double fabs(double x)`	absolute value of `x`
`double floor(double x)`	largest integer non greater than `x`
`double fmod(double x, double y)`	remainder of `x/y` (sign equal to that of `x`)
`double log(double x)`	natural logarithm of `x`
`double log10(double x)`	base-ten logarithm of `x`
`double pow(double x, double y)`	`x` raised to the power of `y`
`double sin(double x)`	sine of `x` (`x` measured in radians)
`double sinh(double x)`	hyperbolic sine of `x`
`double sqrt(double x)`	nonnegative square root of `x`
`double tan(double x)`	tangent of `x` (`x` measured in radians)
`double tanh(double x)`	hyperbolic tangent of `x`

- There can be at most one specification of the initial probability distribution.
- The number and type of the arguments of a model-specific function must match its succinct prototype.
- The number and type of the arguments of a supported standard C function must match the prototype given in Table 2.1.
- The type of the result of the expressions following the keywords `with_rate`, `with_prob`, `reward_rate`, and `initial_probability` must be `double`.[1]
- The type of the result of the expressions following the operators `=`, `+=`, `-=`, `*=`, and `/=` must be `int`.

[1] The type of the result of an expression is determined as in C.

Section 3

Model Debugging

METFAC-2.1 provides two mechanisms for model debugging. The first mechanism consists in checking assertions on the descriptions in terms of the values of the state variables of the generated or sampled states using a function with predefined name and prototype called `check_state`. The definition of that function has to be included in the optional C file of a model specification. The prototype of the function is:

```
int check_state(int sv[], int ipar[], double dpar[])
```

where

- `sv[]` holds, starting at location 0, the values of the state variables, given in the order they have been declared in the model specification file;
- `ipar[]` holds, starting at location 0, the values of the `int` parameters, given in the order they have been declared in the model specification file;
- `dpar[]` holds, starting at location 0, the values of the `double` parameters, given in the order they have been declared in the model specification file.

Inclusion of the statement "`DECLARE_SYMBOLS;`" in the declarative part of the function allows the use in its body of the names of the state variables and parameters declared in the model specification file.[1] We recall that to be able to use that statement, the optional C file of the model specification, *name*`.c`, where *name* is the model name (`TMR_hybrid` for the example of Section 1), has to include the header file *mod*`.h` that is generated automatically when the model is successfully compiled.

The function `check_state` is invoked for every generated or sampled state, and an error occurs if the function returns 0 for some state. If the optional C file of the model specification does not include a definition for the function or the model specification does not include a C file, a default `check_state` function that returns 1 for each state is used.

A possible `check_state` function for the TMR_hybrid example described in Section 1 is:

[1] `DECLARE_SYMBOLS` is a C macro that defines local variables with the names of the state variables and the parameters of the model specification and assigns to the former elements of the array `sv[]` and to the latter elements of the arrays `ipar[]` and `dpar[]` (see Section B.2, page 92).

```
int
check_state(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (NA < 0 || NA > 3 || NUS < 0 || NUS > 2
      || (R != 1 && R != 0))
    return 0;
  else
    return 1;
}
```

Alternatively, that function could be defined without making use of the C macro DECLARE_SYMBOLS as follows:

```
int
check_state(int sv[], int ipar[], double dpar[])
{
  if (sv[0] < 0 || sv[0] > 3 || sv[1] < 0 || sv[1] > 2
      || (sv[2] != 1 && sv[2] != 0))
    return 0;
  else
    return 1;
}
```

To illustrate the use of the assertion-checking mechanism, let us assume that the previous check_state function has been defined in the optional C file of the TMR_hybrid example and that the third production rule of the example, which models the failure of an active processing module, and should read as

```
if sys_up(NA, R) action FAIL_ACTIVE with_rate NA*LD_M
   if NUS>0 response CONTINUE
      next_state NUS--
   if NUS==0 response DUPLEX
      next_state NA--
end
```

has been actually typed as

```
if sys_up(NA, R) action FAIL_ACTIVE with_rate NA*LD_M
   if NUS>=0 response CONTINUE
      next_state NUS--
   if NUS==0 response DUPLEX
      next_state NA--
end
```

i.e., we have mistakenly typed "NUS>=0" instead of "NUS>0" as condition of the first response. After compiling the example and running TMR_hybrid.exe with the same input as in Section 1.3, we would get on the standard error stream (usually the computer terminal) the following error messages and the execution would be aborted without computing the measure:

```
[metfac2 error 1] Wrong state.
State (8):
NA=3, NUS=-1, R=1
reached from state (6):
```

```
NA=3, NUS=0, R=1
through response=1 of action=3.

[metfac2 error 2] Wrong state.
State (10):
NA=3, NUS=-1, R=0
reached from state (8):
NA=3, NUS=-1, R=1
through response=1 of action=1.
```

The second mechanism implemented in METFAC-2.1 for model debugging is the generation in textual format of a verbose description of the rewarded CTMC. The description is written in the file *name*`.log`, where *name* is the name of the model (`TMR_hybrid` for the example of Section 1). This is task number 3 in the interaction shown on page 18. The description includes, for each state, the state index,[2] the values of the state variables, the initial probability of the state, the reward rate of the state, whether the state is vanishing, whether the state is absorbing, the value returned by the function `block` for the state (see Section 4.2, page 47), called "block_index" in the file, the pivot flag, which is equal to "yes" if the state has been selected as a "pivot" state (see Section 4.2, page 49) and is equal to "no" otherwise, and the regenerative flag, which is equal to "yes" if the state has been selected as the "regenerative" state (see Section 4.1, page 41) and is equal to "no" otherwise. In addition, for each state the description includes the list of the pairs action-response and instantaneous action-response that are enabled in the state, giving for each pair the identifier and index of the action or instantaneous action, the identifier and index of the response,[3] the index of the reached state, and the value of the corresponding transition rate or the string "+infinity" if the action is instantaneous. Section C.1 of Appendix C defines formally the description of the rewarded CTMC included in the file *name*`.log`.

Since generating a verbose description of the rewarded CTMC is intended for model debugging, the user is allowed to set a limit on the number of states the CTMC may have (a faulty model description may easily result in a CTMC with a very large state space or even a CTMC with infinite number of states). Then, to generate that description, the user has to execute the model, introduce the numerical values of the model parameters, choose the task "Generate a verbose description of the rewarded CTMC", and set the maximum number of states. Thus, for the example `TMR_hybrid.log`, assuming the assignments $\lambda_{\mathrm{M}} = 10^{-4}$ h^{-1}, $\lambda_{\mathrm{R}} = 10^{-6}$ h^{-1}, and $\theta = 0.2$ and assuming that the user chose to set to $1,000$ the maximum number of states the rewarded CTMC may have, the interaction would look like

```
                    Model parameters
                    ----------------


LD_M: 1e-4
LD_R: 1e-6
THETA: 0.2
```

[2] States are numbered consecutively starting at 1, following the order in which they are created.

[3] Actions, instantaneous actions, and responses without identifier are regarded as having the identifier "nolabel"; actions without responses and instantaneous actions are always dealt with as having one response which is always enabled and has identifier "nolabel" and probability equal to 1. Actions and instantaneous actions are indexed 1, 2, ... in the order they appear in the model specification file. Responses are indexed 1, 2 ... within each action in the order they appear in the model specification file.

```
                Tasks
                -----

          1. Compute a measure using numerical methods.
          2. Estimate a measure using simulation.
          3. Generate a verbose description of the
             rewarded CTMC.
          4. Generate a compact description of the
             rewarded CTMC.

Option (1-4)? 3

Maximum number of states (>=1; hit enter key to choose
the default: 2147483647)? 1000
```

A sketch of the contents of the file TMR_hybrid.log follows:

```
Model compiled on Feb  4 2012 at 17:36:03.
Model executed on Feb  4 2012 at 17:36:25.

Parameters:

LD_M=1.000000E-04
LD_R=1.000000E-06
THETA=2.000000E-01

Generation:

CTMC characteristics:
(Warning: vanishing and unreachable states, self transitions, and
null transition rates are included)
# states=10
(# vanishing_states=5)
# transition rates=13

Spent time in s (user, system, total)=0.000000E+00, 0.000000E+00, 0.000000E+00

description of states
---------------------

state=1
NA=3, NUS=2, R=1
initial_probability=7.500000E-01
reward_rate=1.000000E+00
vanishing=no
absorbing=no
block_index=0
pivot_flag=no
regenerative_flag=no

...

state=4
NA=0, NUS=0, R=0
```

```
initial_probability=0.000000E+00
reward_rate=0.000000E+00
vanishing=no
absorbing=yes
block_index=0
pivot_flag=no
regenerative_flag=no

...

state=10
NA=1, NUS=0, R=1
initial_probability=0.000000E+00
reward_rate=0.000000E+00
vanishing=yes
absorbing=no
block_index=0
pivot_flag=no
regenerative_flag=no

action-response and instantaneous action-response pairs
-------------------------------------------------------

state=1
  action=FAIL_RESTORE(1)
    response=nolabel(1)
      next_state=2
      transition_rate=1.000000E-06
  action=FAIL_SPARE(2)
    response=nolabel(1)
      next_state=3
      transition_rate=4.000000E-05
  action=FAIL_ACTIVE(3)
    response=CONTINUE(1)
      next_state=3
      transition_rate=3.000000E-04

...

state=4
  no action-response or instantaneous action-response pair is enabled in this state

...

state=10
  instantaneous_action=MERGE(4)
    response=nolabel(1)
      next_state=4
      transition_rate=+infinity

# states=10
(# absorbing_states=1, # vanishing_states=5)

Total time in s (user, system, total)=4.000000E-03, 4.000000E-03, 8.000000E-03
```

To ease model debugging, the description includes vanishing states, unreachable states,[4] self-transitions, and null transition rates. However, if the user-given limit on the number of states is reached, then the description is generated up to the point of reaching (approximately) that limit, an error is reported to the standard error output, and the string "`ABORTED!`" is added to the file *name*`.log`. Also, if there is a state for which any of the following conditions hold:

1. the function `check_state` returns 0 for the state;
2. it is the state specified by means of the `start_state` construct and one or more instantaneous actions are enabled in it; or
3. it is not the state specified by means of the `start_state` construct and two or more instantaneous actions are enabled in it;

then the description is generated up to such a state, an error is reported to the standard error output, and the string "`ABORTED!`" is added to the file *name*`.log`.

[4] A non-vanishing state i of a CTMC X is said to be unreachable if $P[X(t) = i] = 0$ for all $t \geq 0$. A necessary and sufficient condition for that is that the initial probability of the state be null and there does not exist any path in the transition diagram of the CTMC from some state with nonnull initial probability to that state.

Section 4

Measures

METFAC-2.1 offers the following seven reward measures:

- Expected Transient Reward Rate ($\mathrm{ETRR}(t)$).
- Expected Steady-State Reward Rate (ESSRR).
- Expected Averaged Reward Rate ($\mathrm{EARR}(t)$).
- Cumulative Reward Complementary Distribution ($\mathrm{CRCD}(t, s)$).
- Interval Availability Complementary Distribution ($\mathrm{IAVCD}(t, p)$).
- Expected Cumulative Reward Till Exit of a subset of states (ECRTE).
- Cumulative Reward Distribution Till Exit of a subset of states ($\mathrm{CRDTE}(s)$).

In this section, we will define all those measures, describing the numerical and simulation methods offered by the tool to, respectively, compute and estimate each measure. Among the numerical methods, some can only be applied on rewarded CTMCs fulfilling some conditions, which will be clearly identified. To fully understand them, though, the reader must take account of the fact that in METFAC-2.1, computing a measure using numerical methods involves the following steps, which are carried out in order:

1. Generation.

   The rewarded CTMC that results from the model specification is generated. During the generation, it is checked that:

   - the function `check_state` returns a value $\neq 0$ for every state;
   - no instantaneous action gets enabled in the state specified by means of the `start_state` construct and at most one instantaneous action gets enabled in any other state;
   - the reward rate of every vanishing state is a finite C `double`;[1]

[1] I.e., it is neither infinite nor a "not-a-number".

- the rate of every (non instantaneous) action that gets enabled in a state is a finite C `double` > 0;
- the probability of every response that gets enabled in a state is > 0 and ≤ 1;
- for each (non instantaneous) action-response pair that gets enabled in a state, the product of the action's rate and the response's probability is a finite C `double` > 0;
- there are not self-transitions; and
- the initial probability of every state is ≥ 0 and ≤ 1.

If any of the above tests fails, an error occurs and the execution is aborted with an explanatory message.

2. Elimination of vanishing states.

   If the rewarded CTMC has vanishing states, it is checked that no cycle of vanishing states exists.[2] If it does, then an error occurs and the execution is aborted with an explanatory message. Otherwise, vanishing states are "eliminated" using a variant of Gaussian elimination without subtractions and a warning is issued.

3. Verification of initial probabilities.

   If the difference, in absolute value, between the sum of the initial probabilities of the states of the rewarded CTMC with vanishing states eliminated and 1 is larger than 50,000 times the "epsilon" constant of the underlying hardware,[3] then a warning is issued.

4. Reachability test.

   If one or more states of the rewarded CTMC with vanishing states eliminated are unreachable,[4] the execution is aborted with an explanatory message.

5. Measure-specific tests.

   If the measure is not well-defined for the rewarded CTMC with vanishing states eliminated or that rewarded CTMC does not fulfill some of the assumptions on which the computation of the measure relies, the execution is aborted with an explanatory message.

6. Computation.

   The measure is computed using the selected numerical method(s).

In the remaining of this section, we will use the following notation.

$X^{(\mathrm{v})}$ Finite rewarded CTMC that results from the model specification, i.e., with vanishing states *not* eliminated. If the measure is to be computed using numerical methods, the initial probability distribution is the one specified by means of the `initial_probability` construct (see Section 1.1.1, page 9). If the specification is absent or the measure is to be estimated using simulation, the initial probability distribution is: Initial probability equal to 1 for

[2] A cycle of vanishing states is a set of states mutually reachable via instantaneous actions.

[3] The "epsilon" constant is the difference between the smallest exactly representable number greater than 1 and 1.

[4] A non-vanishing state i of a CTMC X is said to be unreachable if $P[X(t) = i] = 0$ for all $t \geq 0$.

	the state specified by means of the `start_state` construct and equal to 0 for the remaining states.
$\Omega^{(\mathrm{v})}$	State space of $X^{(\mathrm{v})}$.
$\lambda_{i,j}^{(\mathrm{v})}$	Transition rate of $X^{(\mathrm{v})}$ from state i to state j, $i,j \in \Omega^{(\mathrm{v})}$. (If state i is vanishing, then $\sum_{k\neq i} \lambda_{i,k}^{(\mathrm{v})} = +\infty$ and there exists one and only one state k such that $\lambda_{i,k}^{(\mathrm{v})} = +\infty$.)
$r_i^{(\mathrm{v})}$	Reward rate of state $i \in \Omega^{(\mathrm{v})}$.
X	Finite rewarded CTMC that results from the model specification with vanishing states *eliminated*.
Ω	State space of X.
$\boldsymbol{\alpha}$	Initial probability distribution column vector of X: $(\alpha_i)_{i\in\Omega}$, $\alpha_i = P[X(0) = i]$.
r_i	Reward rate of state $i \in \Omega$.
$\mathbf{A}$	Infinitesimal generator of X: $(a_{i,j})_{i,j\in\Omega}$. For $i \neq j$, $a_{i,j} = \lambda_{i,j}$, where $\lambda_{i,j}$ is the transition rate of X from state i to state j; $a_{i,i} = -\lambda_i$, where $\lambda_i = \sum_{k\in\Omega,k\neq i} \lambda_{i,k}$ is the output rate of X from state i.
$\mathbf{A}^{B,B'}$	Block of $\mathbf{A}$ including the elements $a_{i,j}$, $i \in B$, $j \in B'$, $B, B' \subset \Omega$: $(a_{i,j})_{i\in B,j\in B'}$.
$\mathrm{diag}\left(\mathbf{A}^{B,B'}\right)$	Matrix with same diagonal elements as $\mathbf{A}^{B,B'}$ and null off-diagonal elements.
$\lambda_{i,B}$	$\sum_{j\in B} \lambda_{i,j}$, $B \subset \Omega - \{i\}$.
α_B	$\sum_{i\in B} \alpha_i$.
$\boldsymbol{\alpha}^B$	$(\alpha_i)_{i\in B}$, $B \subset \Omega$.
$\mathbf{p}(t)$	Probability distribution column vector of X at time t: $(p_i(t))_{i\in\Omega}$, $p_i(t) = P[X(t) = i]$.
$\mathbf{I}$	Identity matrix of appropriate dimensions.
$\mathbf{0}$	Column vector of appropriate dimension with all its elements equal to 0.
$\mathbf{1}$	Column vector of appropriate dimension with all its elements equal to 1.
$\mathbf{1}_c$	Indicator function returning value 1 when condition c is satisfied and value 0 otherwise.
$\lVert\cdot\rVert_1$	One norm.
$\lVert\cdot\rVert_2$	Two norm.
$\lVert\cdot\rVert_{\mathrm{F}}$	Frobenius norm.

$|\cdot|$ Absolute value or cardinality.

4.1 Expected Transient Reward Rate

The measure is defined as the expected value of the random variable "reward rate at time t". Formally,

$$\mathrm{ETRR}(t) = E\left[r_{X(t)}\right] .$$

It is assumed that the reward rate of each state of X is ≥ 0. That restriction can be, however, circumvented by shifting otherwise the reward rates by a positive amount d so that the new reward rates $r'_i = r_i + d$ are all ≥ 0. The expected transient reward rate of X is related to the expected transient reward rate measure, $\mathrm{ETRR}'(t)$, of the rewarded CTMC with shifted reward rates by $\mathrm{ETRR}(t) = \mathrm{ETRR}'(t) - d$. Rewarded CTMCs with impulse rewards $r_{i,j}$ that are earned each time X makes a transition from state i to state j can be also accommodated by adding the contribution $\lambda_{i,j} r_{i,j}$ to the reward rate associated with state i. The mapping requires redefining the $\mathrm{ETRR}(t)$ measure as

$$\mathrm{ETRR}(t) = \lim_{\Delta t \to 0^+} \frac{E[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]]}{\Delta t}$$

and is justified in Appendix E.

As an example of the measure, assume that X models a fault-tolerant system that can be either up or down, and that a reward rate 1 is assigned to the states of X in which the system is up and a reward rate 0 is assigned to the states of X in which the system is down. Then, $\mathrm{ETRR}(t)$ would be the availability of the system at time t (probability that the system is up at time t).

Available numerical methods for computing this measure are:

- Standard Randomization (SR).
- Standard Randomization with control of the Relative Error (SRRE).
- Randomization with Stationarity Detection (RSD).
- Randomization with Quasistationarity Detection (RQSD).
- Regenerative Randomization (RR).
- Regenerative Randomization with Laplace Transform Inversion (RRLT).
- Bounding Regenerative Randomization (BRR).
- Explicit Runge-Kutta ODE Solver (ERKODES).
- Implicit Runge-Kutta ODE Solver (IRKODES).

Methods SR and SRRE are directly based on the interpretation of X in terms of a Poisson process and a randomized discrete-time Markov chain (see Appendix F). The first of such methods is quite standard and is implemented in METFAC-2.1 as described in [1]. The second method is more elaborated and is described in detail in [2]. The difference between these methods is that

in SR the measure is computed with control of the absolute error while in method SRRE what is controlled is the relative error.

Method RSD combines the randomization interpretation with stationarity detection and is described in [3].

Method RQSD combines the randomization interpretation with quasistationarity detection and is described in [4].

Methods RR and RRLT follow a different approach [5, 1, 6]. In these methods, a truncated transformed rewarded CTMC model Y is obtained from X of (hopefully) smaller size by characterizing with enough accuracy the behavior of X till the time it hits a "regenerative" state and between consecutive hits to that state. The rewarded CTMC model Y, which has with some arbitrarily small error the same $\mathrm{ETRR}(t)$ measure as X, is then solved either using the SR method (RR) or using a numerical Laplace transform inversion algorithm on a closed-form Laplace transform solution of Y (RRLT). The regenerative state is selected by the user using the function with predefined name and prototype `regstat` whose definition has to be included in the optional C file of a model specification. The prototype of that function is:

```
int regstat(int sv[], int ipar[], double dpar[], long index)
```

where

- `sv[]` holds, starting at location 0, the values of the state variables, given in the order they have been declared in the model specification file;
- `ipar[]` holds, starting at location 0, the values of the `int` parameters, given in the order they have been declared in the model specification file;
- `dpar[]` holds, starting at location 0, the values of the `double` parameters, given in the order they have been declared in the model specification file;
- `index` holds the state index, which is equal to 1 for the state specified through the `start_state` construct.

Inclusion of the statement "`DECLARE_SYMBOLS;`" in the declarative part of the function allows the use in the body of the function of the names of state variables and parameters declared in the model specification file. We recall that to be able to use that statement, the optional C file of the model specification, *name*`.c`, *name* being the model name, has to include the header file *name*`.h` that is generated automatically when the model is compiled. The function is invoked for every generated state. The regenerative state will be the state of X for which the function returns a value different from zero. (If a value different from zero is returned for more than one state or zero is returned for every state, an error occurs.) If the optional C file of the model specification does not include a definition for the function or the model specification does not include a C file, a default `regstat` function that returns 0 for every state is used.

Method BRR [7] computes a lower bound, an upper bound, or both for $\mathrm{ETRR}(t)$ and also requires the selection of a regenerative state. In the method, the transition rates from all states except the regenerative state are scaled using a user-given *equalization* parameter and the resulting rewarded CTMC is then solved using method RR. Of course, the advantage is that method BRR can be much faster than method RR. The regenerative state has to be selected using the function with predefined name and prototype `regstat`.

Unlike the previous methods, which are all based on the randomization interpretation, methods ERKODES and IRKODES compute $\mathbf{p}(t)$ as the solution at $\tau = t$ of the linear ODE

$$\frac{\mathrm{d}\,\mathbf{p}(\tau)}{\mathrm{d}\,\tau} = \mathbf{A}^{\mathrm{T}}\mathbf{p}(\tau) \tag{4.1}$$

with the initial condition $\mathbf{p}(0) = \boldsymbol{\alpha}$ and next compute $\mathrm{ETRR}(t) = \sum_{i\in\Omega} r_i p_i(t)$. In method ERKODES, the ODE (4.1) is solved using one of two explicit Runge-Kutta ODE solvers: RK5(4)7M [8] and RK6(5)8M [9]. The respective orders are 5 and, for linear ODEs, 7, and the user selects the ODE solver by choosing the order. In method IRKODES, the ODE (4.1) is solved using the implicit Runge-Kutta ODE solver Radau IIA [10] with $s = 2, 3, 4, 5$, or 6 stages. The respective orders are $2s - 1 = 3, 5, 7, 9, 11$ and the user selects the number of stages by choosing the order. The ODE solver Radau IIA is A- and L-stable. The solver has been implemented along the lines of the 3-stage Radau IIA ODE solver described in [11, Sect. IV.8] with two significant modifications: 1) The selection of the step size is based on a lower order method developed following the methodology described in [12], and 2) using results from [13, 14, 15], the involved linear systems are solved using the Gauss-Seidel and Bi-CGSTAB [16] methods with strict control of the 1-norm of the error.

Methods SR, SRRE, ERKODES, and IRKODES can be used for any X. The remaining methods are less general and require the following conditions to hold, where r denotes the chosen regenerative state:

- Method RSD: X is irreducible.
- Method RQSD:
  - The state space Ω of X is of the form $\Omega = S \cup \{f_1, \ldots, f_A\}$, $A \geq 1$, where all states in S make up a single transient class of states and the states f_i, $1 \leq i \leq A$ are absorbing and have associated with them different reward rates.
  - The sum of the initial probabilities of the states in S is > 0.
- Methods RR, RRLT:

  C1. $\Omega = S \cup \{f_1, f_2, \ldots, f_A\}$, $\left|S\right| \geq 2$, $A \geq 0$, where either all states in S are transient or S includes a single recurrent class of states C and the states f_i are absorbing and have associated with them different reward rates.

  C2. $r \in S$.

  C3. If S includes a single recurrent class of states C, $r \in C$.

  C4. There exists some transition rate from r to some state in $S - \{r\}$.
- Method BRR: Conditions C1 through C4 above with $A > 0$ and the condition:

  C5. All states in S have null reward rate.

  Moreover, the equalization parameter, D, has to have a value satisfying $1 \leq D < \max_{i\in S-\{r\}} \lambda_i / \min_{i\in S-\{r\}} \lambda_i$. The bounds become tighter and computationally more costly as D increases.

To compute the measure $\mathrm{ETRR}(t)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)

2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:

   (a) Assign values to the parameters of the model if any.

   (b) Choose the task "`Compute a measure using numerical methods`".

   (c) Choose the measure "`Expected Transient Reward Rate (ETRR(t))`".

   (d) Choose the numerical method.

   (e) Assign values to the parameters controlling the chosen numerical method. These parameters are, in the order in which the user will be prompted for them in the interaction:

      - SR: Allowed absolute error and allowed CPU time in seconds (an integer value).
      - SRRE: Allowed relative error and allowed CPU time in seconds.
      - RSD, RQSD, RR: Allowed absolute error and allowed CPU time in seconds.
      - RRLT: Absolute tolerance and allowed CPU time in seconds.
      - BRR: Bounds that have to be computed (lower bound, upper bound or both), equalization parameter (the larger the parameter, the tighter and more computationally costly the bounds will be), allowed absolute error, and allowed CPU time in seconds.
      - ERKODES: Order of the ODE solver (either 5 or 7), absolute tolerance, and allowed CPU time in seconds.
      - IRKODES: Order of the ODE solver (either 3, 5, 7, 9, or 11), absolute tolerance, and allowed CPU time in seconds.

   (f) Define the grid of time abscissae t at which the measure has to be computed. (See Section 1.3, page 18.)

   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected method can be used and exits providing an explanatory message if the method cannot.

- Neglecting round-off errors, in methods SR, SRRE, RSD, RQSD, RR, and BRR, the error with which the measure or bounds for it are computed is guaranteed to be non greater than the specified allowed error. In the remaining methods, the absolute error with which the measure is computed can be larger than the specified absolute tolerance.[5]

[5] However, numerical experiments [6] seem to indicate that for method RRLT, the actual error is non greater than the specified absolute tolerance.

- Let $\rho = \max_{i \in \Omega} \lambda_i t_{\max}$, where $t_{\max}$ is the largest value of t for which the measure has to be computed. For small values of ρ, method ERKODES can be faster than the remaining methods. For large values of ρ, methods RSD, RQSD, RR, RRLT, BRR, and IRKODES can be significantly faster than methods SR, SRRE, and ERKODES (the CPU time required by methods SR and SRRE is approximately directly proportional to ρ). For very large values of ρ, method IRKODES can be the fastest one. Recall, however, than methods RSD, RQSD, RR, RRLT, and BRR are less general than methods SR and SRRE, and that methods RRLT, ERKODES, and IRKODES do not provide strict error control.

- Selecting an appropriate regenerative state for methods RR, RRLT, and BRR is a delicate issue. As a general rule, the regenerative state should be a state visited often by the randomized discrete-time Markov chain of X with randomization rate slightly larger than $\max_{i \in \Omega} \lambda_i$. When the choice is not very clear, the user should be aware that a "bad" selection for the regenerative state can degrade severely the performance of the methods. For exact and bounding failure/repair rewarded CTMCs of fault-tolerant systems with exponential failure and repair time distributions and repair in every state with failed components with failure rates much smaller than repair rates, a good choice for the regenerative state is the state without failed components. For other types of models for which a good selection for the regenerative state exists, see [5, 1, 6, 7].

- The method RRLT can be significantly less costly than the method RR when in the latter the computational cost of the second phase of the method (solution of the truncated transformed rewarded CTMC using method SR) dominates the computational cost of the first phase (generation of the truncated transformed rewarded CTMC).

- For methods RR, RRLT, and BRR, the condition that there exist some transition rate from the regenerative state, r, to some state in $S - \{r\}$ (condition C4 on page 42) can be circumvented by adding a tiny transition rate $\lambda \leq (10^{-10}\varepsilon)/(2r_{\max}t_{\max})$, where ε is the allowed absolute error, $r_{\max} = \max_{i \in \Omega} r_i$, and $t_{\max}$ is the largest value of t for which the measure has to be computed, with a negligible impact on the measure non greater than $10^{-10}\varepsilon$.

At present, only one simulation method is available for estimating the measure. In the interaction with the user, that method is listed under the name "`Independent Realizations with Forced Transitions of the Reward Rate`". The method consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the standard normal approximation confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded embedded (homogeneous) discrete-time Markov chain (DTMC) described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct (the "start state") and equal to 0 for the remaining states. Accordingly, each realization starts at the start state and ends when the sum of the sampled sojourn times is non smaller than the time point of interest t. During a realization, a transition is forced whenever: 1) the number of transitions that have been forced up to that point throughout the simulation has not yet reached a user-given limit, 2) the current

state, a, of the realization is neither absorbing nor vanishing, and 3) $r_a^{(\mathrm{v})} = 0$. If $T_a < t$ denotes the sum of the sampled sojourn times in the states of the current realization up to entry into state a, the transition is forced by limiting the sojourn time in that state to $t - T_a$. This is achieved by changing the probability distribution function of the random variable "sojourn time in state a" from $F(u) = 1-\mathrm{e}^{-\lambda_a^{(\mathrm{v})}u}$, $u \geq 0$, to $F'(u) = (1-\mathrm{e}^{-\lambda_a^{(\mathrm{v})}u})/(1-\mathrm{e}^{-\lambda_a^{(\mathrm{v})}(t-T_a)})$, $0 \leq u \leq t-Ta$. For each realization, the sample of the estimator is the reward rate of the last visited state multiplied, if one or more transitions have been forced, by a factor that takes account of those forced transitions. If $\mathcal{F}$ denotes the collection of (possibly repeated) states of the realization where transitions have been forced, that factor is $\prod_{b\in\mathcal{F}}(1 - \mathrm{e}^{-\lambda_b^{(\mathrm{v})}(t-T_b)})$. The method uses the 2002-version of the pseudo-random number generator (RNG) MT19937 [17].

To estimate the measure $\mathrm{ETRR}(t)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Estimate a measure using simulation`".
   (c) Choose the measure "`Expected Transient Reward Rate (ETRR(t))`".
   (d) Choose the simulation method "`Independent Realizations with Forced Transitions of the Reward Rate`".
   (e) Assign values to the parameters controlling the method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - Confidence level as a decimal value (e.g., 0.99).
      - Allowed relative half-width of the confidence interval.
      - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
      - Value of the seed or name of the file.
      - Allowed number of forced transitions.
      - Minimum number of realizations for which the estimator is positive.
      - Allowed CPU time in seconds (an integer value).
   (f) Introduce the value of the time point of interest $t > 0$.
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.

- In a realization, there is a state in which two or more instantaneous actions get enabled.
- In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.
- In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
- In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
- In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
- In a realization, there is a self-transition.
- In a realization, there is a cycle of vanishing states.

4.2 Expected Steady-State Reward Rate

The measure is defined as the limit for $t \to \infty$ of the expected value of the random variable "reward rate at time t". Formally,

$$\mathrm{ESSRR} = \lim_{t\to\infty} E\left[r_{X(t)}\right] .$$

It is assumed that all reward rates of the recurrent states of X are ≥ 0. Rewarded CTMCs with negative reward rates in recurrent states and rewarded CTMCs with impulse rewards can be accommodated, however, as explained for the $\mathrm{ETRR}(t)$ measure by noting that $\mathrm{ESSRR} = \lim_{t\to\infty} \mathrm{ETRR}(t)$.

As an example of the measure, assume that X models a fault-tolerant system that can be either up or down, and that a reward rate 1 is assigned to the states of X in which the system is up and a reward rate 0 is assigned to the states of X in which the system is down. Then, ESSRR would be the steady-state availability of the system.

Let m denote the number of recurrent classes of states of X. As shown in Appendix E, computation of the measure involves computing the 1-normalized solution ($\left\|\mathbf{p}^k\right\|_1 = 1$) of the singular linear systems

$$\left(\mathbf{p}^k\right)^{\mathrm{T}} \mathbf{A}^{C_k,C_k} = \mathbf{0}^{\mathrm{T}}, \quad 1 \leq k \leq m\,, \tag{4.2}$$

and, if $m > 1$ and the set, S, of transient states of X is non-empty, solving the non-singular linear system

$$\boldsymbol{\tau}^{\mathrm{T}} \mathbf{A}^{S,S} = -\left(\boldsymbol{\alpha}^S\right)^{\mathrm{T}} . \tag{4.3}$$

Let $S_1, \ldots, S_n$ denote the transient classes of states of X, so that $S = \cup_{k=1}^{n} S_k$, In METFAC-2.1, if $n > 1$, the matrix $\mathbf{A}^{S,S}$ is permuted into block upper triangular form

$$\mathbf{A}^{S,S} = \begin{pmatrix} \mathbf{A}^{S_1,S_1} & \mathbf{A}^{S_1,S_2} & \ldots & \mathbf{A}^{S_1,S_n} \\ & \mathbf{A}^{S_2,S_2} & \ldots & \mathbf{A}^{S_2,S_n} \\ & & \ddots & \vdots \\ & & & \mathbf{A}^{S_n,S_n} \end{pmatrix},$$

and the solution $\boldsymbol{\tau}^{\mathrm{T}} = ((\boldsymbol{\tau}^1)^{\mathrm{T}}, \ldots, (\boldsymbol{\tau}^n)^{\mathrm{T}})$ of (4.3) is obtained by solving, for increasing k starting at $k = 1$, the non-singular linear systems

$$(\boldsymbol{\tau}^k)^{\mathrm{T}} \mathbf{A}^{S_k,S_k} = -(\boldsymbol{\alpha}^{S_k})^{\mathrm{T}} - \sum_{j=1}^{k-1} (\boldsymbol{\tau}^j)^{\mathrm{T}} \mathbf{A}^{S_j,S_k}, \quad 1 \leq k \leq n\,. \tag{4.4}$$

The available numerical methods for solving singular linear systems are:

- LU Decomposition (LUD).
- Gauss-Seidel (GS).
- Block Gauss-Seidel (BGS).
- Adaptive Successive Overrelaxation (ASOR).
- Adaptive Generalized Minimal Residual (AGMRES).
- Stationarity Detection (SD).

Method LUD is a sparse implementation of the LU decomposition computed using the so-called GTH algorithm [18]. Using that algorithm, in which subtractions are avoided altogether, the decomposition is computed with low relative error [19]. The downside is a large memory consumption as compared to a sparse implementation of standard Gaussian elimination [20], in which the lower-triangular matrix of the decomposition needs not be stored.

Descriptions of the well-known methods GS and BGS can be found in [21]. For method BGS, the blocks of states have to be identified by the user using the function with predefined name and prototype `block` whose definition has to be included in the optional C file of the model specification. The prototype of that function is:

```
long block(int sv[], int ipar[], double dpar[], long index)
```

where

`sv[]`	holds, starting at location 0, the values of the state variables, given in the order they have been declared in the model specification file;
`ipar[]`	holds, starting at location 0, the values of the `int` parameters, given in the order they have been declared in the model specification file;
`dpar[]`	holds, starting at location 0, the values of the `double` parameters, given in the order they have been declared in the model specification file;
`index`	holds the state index, which is equal to 1 for the state specified through the `start_state` construct.

Inclusion of the statement "`DECLARE_SYMBOLS;`" in the declarative part of the function allows the use in the body of the function of the names of the state variables and parameters declared in the model specification file. We recall that to be able to use that statement, the optional C file of the model specification, *name*`.c`, *name* being the model name, has to include the header file *name*`.h`

that is generated automatically when the model is compiled. The function is invoked for every generated state. A block consists of all the states of the recurrent class of states of X for which the function returns the same value. (If there is only one block or as many blocks as states the recurrent class of states has, a warning is issued.) If the optional C file of the model specification does not include a definition for the function or the model specification does not include a C file, a default `block` function that returns 0 for every state is used.

Method ASOR is a Successive Overrelaxation method [21] with the relaxation parameter dynamically optimized so as to reduce the number of iterations required to achieve convergence. Save for the stopping criterion (see page 50), the method implemented in METFAC-2.1 is the one described in [22].

Method AGMRES is a variant [23] of the Generalized Minimal Residual method [24] in which the dimension of the Krylov subspace is changed dynamically instead of having to be provided in advance.

Method SD is a modified version of the method called "Randomization with Stationarity Detection" in Section 4.1 and can only be used when X is irreducible. Strictly speaking, the method obtains ESSRR without computing explicitly the 1-normalized solution of the involved singular linear system.

The available numerical methods for solving non-singular linear systems are:

- LUD.[6]
- GS.
- BGS.
- ASOR.
- AGMRES.
- Accelerated Gauss-Seidel (acc_GS).
- Accelerated Adaptive Successive Overrelaxation (acc_ASOR).

Method BGS requires the user to identify the blocks of states using the function with predefined name and prototype `block` as explained on page 47 save that now, a block consists of all the states of the transient class of states of X for which the function returns the same value.

Method acc_GS is loosely based on the technique described in [26]. The method requires the user to select a non-empty collection of "pivot" states for each class of transient states S_k, $1 \leq k \leq n$, of X and works as follows. For each pivot state in S_k, a modified linear system is solved using method GS. In case there does not exist any pivot state i such that $\alpha_i > 0$ and $\alpha_j = 0$ for all $j \in S_k$, $j \neq i$, then another modified linear system is solved using method GS. Finally, the sought solution is obtained as a linear combination of the solutions of these modified linear systems. If the number of such systems is larger than 1, computing the weights of that combination requires solving a dense linear system with size equal to the number of pivot states in S_k. That

[6] The GTH algorithm requires the row sums of the matrix of the linear system under study to be 0. To apply it to the involved non-singular linear systems, in which at least one row does not add up 0, we use a technique that amounts to the one described in [25].

system is solved using Gaussian elimination with row pivoting. Method acc_ASOR is similar with method GS replaced by method ASOR.

Pivot states are selected using the function with predefined name and prototype `pivot` whose definition has to be included in the optional C file of the model specification. The prototype of that function is:

```
int pivot(int sv[], int ipar[], double dpar[], long index)
```

where the contents of the arrays `sv[]`, `ipar[]`, and `dpar[]`, and the value of `index` are the same as for the `block` function described on page 47. Inclusion of the statement "`DECLARE_SYMBOLS;`" in the declarative part of the function `pivot` allows the use in the body of the function of the names of state variables and parameters declared in the model specification file. The function is invoked for every generated state. The pivot states are the states of the transient class of states of X for which the function returns a value different from zero. (If zero is returned for every state, an error occurs; if there are as many pivot states as states the class has, a warning is issued.) If the C file of the model specification does not include a definition for the function or the model specification does not include a C file, a default `pivot` function that returns 0 for every state is used.

To compute the measure ESSRR for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Compute a measure using numerical methods`".
   (c) Choose the measure "`Expected Steady-State Reward Rate (ESSRR)`".
   (d) Choose the numerical method to solve singular linear systems. (In the interaction, the available methods appear listed under the heading "`Numerical methods (singular linear systems)`".)
   (e) If the chosen method is GS, BGS, or ASOR, set the relative tolerance to solve each singular linear system; if the chosen method is AGMRES, set the absolute tolerance to solve each singular linear system; and, if the chosen method is SD, set the allowed relative error.
   (f) Choose the numerical method to solve non-singular linear systems. (In the interaction, the available methods appear listed under the heading "`Numerical methods (non-singular linear systems)`".)
   (g) If the chosen method is GS, BGS, ASOR, acc_GS, or acc_ASOR, set the relative tolerance to solve each non-singular linear system; if the chosen method is AGMRES, set the absolute tolerance to solve each non-singular linear system.
   (h) Set the allowed CPU time in seconds (an integer value).[7]

[7] This value sets an (approximate) upper limit for the CPU time that can be spent in solving all the involved linear systems.

(i) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected methods can be used and exits providing an explanatory message if any of them cannot.

- For methods GS, BGS, ASOR, and AGMRES, the singular linear systems actually solved are

$$\mathbf{P}^{C_k,C_k}\mathbf{x}^k = \mathbf{0}, \quad 1 \le k \le m\,, \tag{4.5}$$

where $\mathbf{P}^{C_k,C_k} = (\mathbf{A}^{C_k,C_k})^{\mathrm{T}}\left(\mathrm{diag}\left(\mathbf{A}^{C_k,C_k}\right)\right)^{-1}$. The 1-normalized solutions of the linear systems (4.2) are obtained from non-null solutions of the linear systems (4.5) by using

$$\mathbf{p}^k = \frac{\left(\mathrm{diag}\left(\mathbf{A}^{C_k,C_k}\right)\right)^{-1}\mathbf{x}^k}{\left\|\left(\mathrm{diag}\left(\mathbf{A}^{C_k,C_k}\right)\right)^{-1}\mathbf{x}^k\right\|_1}\,.$$

Each linear system (4.5) is solved as follows. Let $\mathbf{x}^{k,l} = (x_i^{k,l})_{i\in C_k}$ denote the solution vector at iteration k and let ε be a user-given tolerance. The iterations start with $\mathbf{x}^{k,0} = (1/|C_k|)\mathbf{1}$. For methods GS, BGS, and ASOR, the stopping criterion is based on comparing $\mathbf{x}^{k,l}$ and $\mathbf{x}^{k,l-j}$, $j \ge 1$, and is

$$\max_{i\in C_k}\left(\frac{\left|x_i^{k,l} - x_i^{k,l-j}\right|}{\left|x_i^{k,l}\right|}\right) \le \varepsilon\,.$$

Following the suggestion given in [21], the value of j is made depend on the iteration index l and is 5 for $l < 100$, 10 for $100 \le l < 500$, 20 for $500 \le l < 1000$, and 50 for $l \ge 1000$. The stopping criterion for method AGMRES is

$$\left\|\boldsymbol{P}^{C_k,C_k}\mathbf{x}^{k,l}\right\|_2 \le \frac{\varepsilon}{\left\|\left(\mathrm{diag}\left(\mathbf{A}^{C_k,C_k}\right)\right)^{-1}\right\|_{\mathrm{F}}\left\|\mathrm{diag}\left(\mathbf{A}^{C_k,C_k}\right)\right\|_\infty}\left\|\boldsymbol{P}^{C_k,C_k}\right\|_{\mathrm{F}}\left\|\mathbf{x}^{k,l}\right\|_2\,.$$

- For all methods save LUD, the non-singular linear systems actually solved are

$$\mathbf{P}^{S_k,S_k}\mathbf{x}^k = -\Big(\boldsymbol{\alpha}^{S_k} + \sum_{j=1}^{k-1}(\mathbf{A}^{S_j,S_k})^{\mathrm{T}}\left(\mathrm{diag}\left(\mathbf{A}^{S_j,S_j}\right)\right)^{-1}\mathbf{x}^j\Big), \quad 1 \le k \le n\,, \tag{4.6}$$

where $\mathbf{P}^{S_k,S_k} = (\mathbf{A}^{S_k,S_k})^{\mathrm{T}}\left(\mathrm{diag}\left(\mathbf{A}^{S_k,S_k}\right)\right)^{-1}$. The solutions of the linear systems (4.4) are obtained from those of (4.6) using $\boldsymbol{\tau}^k = \left(\mathrm{diag}\left(\mathbf{A}^{S_k,S_k}\right)\right)^{-1}\mathbf{x}^k$. Each linear system (4.6) is solved as follows. Let $\mathbf{x}^{k,l} = (x_i^{k,l})_{i\in S_k}$ denote the solution vector at iteration l, $l \ge 0$, and let ε' be a user-given tolerance. The iterations start with $\mathbf{x}^{k,0} = \mathbf{1}$. For methods GS, BGS, ASOR, acc_GS, and acc_ASOR, the stopping criterion is

$$\max_{i\in S_k}\left(\frac{\left|x_i^{k,l} - x_i^{k,l-j}\right|}{\left|x_i^{k,l}\right|}\right) \le \varepsilon\,,$$

with $j = 5$ for $l < 100$, $j = 10$ for $100 \leq l < 500$, $j = 20$ for $500 \leq l < 1000$, and $j = 50$ for $l \geq 1000$. Letting $\mathbf{q}^{k,l} = \boldsymbol{\alpha}^{S_k} + \sum_{j=1}^{k-1} \left(\mathbf{A}^{S_j,S_k}\right)^{\mathrm{T}} \left(\mathrm{diag}\left(\mathbf{A}^{S_j,S_j}\right)\right)^{-1} \widetilde{\mathbf{x}}^j$, where $\widetilde{\mathbf{x}}^j$ denotes the computed $\mathbf{x}^j$, $1 \leq j \leq k-1$, the stopping criterion for method AGMRES is

$$\left\|\mathbf{q}^{k,l} + \mathbf{P}^{S_k,S_k}\mathbf{x}^{k,l}\right\|_2 \leq \frac{\varepsilon'}{\left\|\left(\mathrm{diag}\left(\mathbf{A}^{S_k,S_k}\right)\right)^{-1}\right\|_{\mathrm{F}}\left\|\mathrm{diag}\left(\mathbf{A}^{S_k,S_k}\right)\right\|_\infty} \times \left(\left\|\mathbf{q}^{k,l}\right\|_2 + \left\|\mathbf{P}^{S_k,S_k}\right\|_{\mathrm{F}}\left\|\mathbf{x}^{k,l}\right\|_2\right).$$

- Neglecting round-off errors, if X is irreducible and the chosen method is SD, the actual relative error with which the measure is computed is guaranteed to be non larger than the user-given allowed relative error. For the remaining methods save AGMRES, the actual relative error can be expected to be proportional to the relative tolerance specified for solving singular linear systems if X has only one class of recurrent states.

- For solving singular linear systems:

  - Methods GS and SD are guaranteed to converge.[8] Methods ASOR and BGS can diverge. Method AGMRES can stagnate and never reach the solution and can also break down because of numerical instability.
  - For rewarded CTMCs of fault-tolerant systems with exponential failure and repair times distributions and repair in every state with failed components with failure rates much smaller than repair rates, GS tends to work well. For other types of models, ASOR is usually faster than GS.
  - Method BGS can be very fast if blocks are chosen so that transition rates between states in the same block are appreciably larger than transition rates between states in different blocks.
  - Very often, method AGMRES requires fewer iterations than methods GS, BGS and ASOR. However, method AGMRES has larger memory consumption than the other methods and its iterations are more expensive in terms of CPU time.
  - Typically, method SD will be more expensive in terms of CPU time than the other methods. However, it has the advantage of estimating the measure with controlled (relative) error, ignoring round-off errors.

- For solving non-singular linear systems:

  - Methods GS and acc_GS are guaranteed to converge.
  - Method AGMRES can stagnate and never reach the solution with the specified absolute tolerance and can also break down because of numerical instability.

The available simulation methods for estimating the measure are:

- Regenerative Simulation (RSIM).

[8] Convergence of GS is guaranteed if the ordering of states satisfies some conditions which can always be fulfilled [22] and METFAC-2.1 sorts the states so that the conditions are fulfilled.

- Independent Realizations of the Averaged Reward Rate (IRARR).

Method RSIM requires X to be irreducible and consists in sampling regenerative cycles of the embedded DTMC of $X^{(\mathrm{v})}$ (see Appendix F). The regenerative state is the state specified by means of the `start_state` construct. The simulation is carried out until: 1) the number of regenerative cycles for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the standard normal approximation confidence interval is non larger than a user-given value. If $\mathcal{V}$ denotes the collection of (possibly repeated) non vanishing states of a regenerative cycle, the sample of the estimator for that cycle is $(\sum_{i\in\mathcal{V}} r_i^{(\mathrm{v})}(\lambda_i^{(\mathrm{v})})^{-1})/(\sum_{i\in\mathcal{V}}(\lambda_i^{(\mathrm{v})})^{-1})$. The method uses the 2002-version of the RNG MT19937 [17].

Method IRARR consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the standard normal approximation confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the state specified by means of the `start_state` construct and ends when the sum of the sampled sojourn times is non smaller than a user-given time $t_{\max}$. The sample of the estimator is the reward accumulated over the time interval $[t_{\min}, t_{\max}]$, where $t_{\min} < t_{\max}$ is another user-given time point, divided by $t_{\max} - t_{\min}$. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure ESSRR for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Estimate a measure using simulation`".
   (c) Choose the measure "`Expected Steady-State Reward Rate (ESSRR)`".
   (d) Choose the simulation method.
   (e) Assign values to the parameters controlling the chosen simulation method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - RSIM:
        - Confidence level as a decimal value (e.g., 0.99).
        - Allowed relative half-width of the confidence interval.
        - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).

  - Value of the seed or name of the file.
  - Minimum number of regenerative cycles for which the estimator is positive.
  - Allowed CPU time in seconds (an integer value).
- IRARR:
  - Confidence level as a decimal value (e.g., 0.99).
  - Allowed relative half-width of the confidence interval.
  - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation.
  - Value of the seed or name of the file.
  - Minimum number of realizations for which the estimator is positive.
  - $t_{\min}$
  - $t_{\max} > t_{\min}$
  - Allowed CPU time in seconds (an integer value).

(f) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization or a regenerative cycle, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.
  - In a realization or a regenerative cycle, there is a state in which two or more instantaneous actions get enabled.
  - In a realization or a regenerative cycle, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.
  - In a realization or a regenerative cycle, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
  - In a realization or a regenerative cycle, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
  - In a realization or a regenerative cycle, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
  - In a realization or a regenerative cycle, there is a self-transition.
  - In a realization or a regenerative cycle, there is a cycle of vanishing states.
  - In a regenerative cycle, there is an absorbing state.

4.3 Expected Averaged Reward Rate

The measure is the expected value of the random variable "reward rate averaged over the time interval $[0, t]$". Formally,

$$\mathrm{EARR}(t) = E\left[\frac{\int_0^t r_{X(\tau)}\,\mathrm{d}\tau}{t}\right].$$

Note that from $\mathrm{EARR}(t)$ it is immediate to compute the expected value of the random variable "reward accumulated by X over the time interval $[0, t]$", $\int_0^t r_{X(\tau)}\,\mathrm{d}\tau$, as

$$E\left[\int_0^t r_{X(\tau)}\,\mathrm{d}\tau\right] = t \times \mathrm{EARR}(t)\,.$$

Measures $\mathrm{EARR}(t)$ and $\mathrm{ETRR}(t)$ are related as follows:

$$\mathrm{EARR}(t) = \frac{1}{t}\int_0^t E\left[r_{X(\tau)}\right]\,\mathrm{d}\tau = \frac{1}{t}\int_0^t \mathrm{ETRR}(\tau)\,\mathrm{d}\tau\,.$$

It is assumed that the reward rate of each state of X is ≥ 0. That restriction can be circumvented by shifting otherwise the reward rates by a positive amount d so that the new reward rates $r_i' = r_i + d$ are all ≥ 0. The expected averaged reward rate of X is related to the expected averaged reward rate measure, $\mathrm{EARR}'(t)$, of the rewarded CTMC with shifted reward rates by $\mathrm{EARR}(t) = \mathrm{EARR}'(t) - d$. Rewarded CTMCs with impulse rewards $r_{i,j}$ which are earned each time X makes a transition from state i to state j can be also accommodated by adding the contribution $\lambda_{i,j} r_{i,j}$ to the reward rate associated with state i. The mapping requires redefining the $\mathrm{EARR}(t)$ measure as

$$\mathrm{EARR}(t) = E\left[\frac{\text{reward accumulated by } X \text{ in } [0, t]}{t}\right]$$

and is justified in Appendix E.

As an example of the measure, assume that X models a fault-tolerant system that can be either up or down, and that a reward rate 1 is assigned to the states of X in which the system is up and a reward rate 0 is assigned to the states of X in which the system is down. Then, $\mathrm{EARR}(t)$ would be the expected interval availability of the system (expected fraction of the time interval $[0, t]$ in which the system is up) and $t \times \mathrm{EARR}(t)$ would be the expected amount of time the system is up in $[0, t]$.

Available numerical methods for computing this measure are:

- Standard Randomization (SR).
- Standard Randomization with control of the Relative Error (SRRE).
- Randomization with Stationarity Detection (RSD).
- Randomization with Quasistationarity Detection (RQSD).
- Regenerative Randomization (RR).
- Regenerative Randomization with Laplace Transform Inversion (RRLT).
- Explicit Runge-Kutta ODE Solver (ERKODES).

- Implicit Runge-Kutta ODE Solver (IRKODES).

Methods SR and SRRE are directly based on the interpretation of X in terms of a Poisson process and a randomized discrete-time Markov chain (see Appendix F) and are described in [1] and [2], respectively. The difference between these methods is that in method SR the measure is computed with control of the absolute error whereas in method SRRE what is controlled is the relative error.

Method RSD combines the randomization interpretation with stationarity detection and is described in [3].

Method RQSD combines the randomization interpretation with quasistationarity detection and is described in [4].

Methods RR and RRLT follow a different approach [5, 1, 6]. In these methods, a truncated transformed rewarded CTMC model Y is obtained from X of (hopefully) smaller size by characterizing with enough accuracy the behavior of X till the time it hits a "regenerative" state and between consecutive hits to that state. The rewarded CTMC model Y, which has with some arbitrarily small error the same $\mathrm{EARR}(t)$ measure as X, is then solved either using the SR method (RR) or using a numerical Laplace transform inversion algorithm on a closed-form Laplace transform solution of Y (RRLT). The regenerative state is selected by the user by means of the function with predefined name and prototype `regstat` as described in Section 4.1, page 41.

Let $l_i(x) = \int_0^x p_i(y)\,\mathrm{d}y$ and $\boldsymbol{l}(x) = (l_i(x))_{i\in\Omega}$. Methods ERKODES and IRKODES compute $\boldsymbol{l}(t)$ as the solution at $t = \tau$ of the linear ODE

$$\frac{\mathrm{d}\,\boldsymbol{l}(\tau)}{\mathrm{d}\,\tau} = \mathbf{A}^{\mathrm{T}}\boldsymbol{l}(\tau) + \boldsymbol{\alpha}$$

with the initial condition $\boldsymbol{l}(0) = \mathbf{0}$ and next compute $\mathrm{EARR}(t) = (1/t)\sum_{i\in\Omega} r_i l_i(t)$. In method ERKODES, the ODE (4.1) is solved using one of the explicit Runge-Kutta ODE solvers RK5(4)7M and RK6(5)8M described when dealing with the $\mathrm{ETRR}(t)$ measure (see page 42), and in method IRKODES, the ODE is solved using using the implicit Runge-Kutta ODE solver Radau IIA with order 3, 5, 7, 9, or 11 described when dealing with that measure.

Methods SR, SRRE, ERKODES, and IRKODES can be used for any X. The remaining methods are less general and require the following conditions to hold, where r denotes the chosen regenerative state for methods RR and RRLT:

- Method RSD: X is irreducible.
- Method RQSD:
  - The state space Ω of X is of the form $\Omega = S \cup \{f_1, \ldots, f_A\}$, $A \geq 1$, where all states in S make up a single transient class of states and the states f_i, $1 \leq i \leq A$ are absorbing and have associated with them different reward rates.
  - The sum of the initial probabilities of the states in S is > 0.
- Methods RR, RRLT:

  C1. $\Omega = S \cup \{f_1, f_2, \ldots, f_A\}$, $\left|S\right| \geq 2$, $A \geq 0$, where either all states in S are transient or S includes a single recurrent class of states C and the states f_i are absorbing and have associated with them different reward rates.

C2. $r \in S$.

C3. If S includes a single recurrent class of states C, $r \in C$.

C4. There exists some transition rate from r to some state in $S - \{r\}$.

To compute the measure $\mathrm{EARR}(t)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Compute a measure using numerical methods`".
   (c) Choose the measure "`Expected Averaged Reward Rate (EARR(t))`".
   (d) Choose the numerical method.
   (e) Assign values to the parameters controlling the chosen numerical method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - SR: Allowed absolute error and allowed CPU time in seconds (an integer value).
      - SRRE: Allowed relative error and allowed CPU time in seconds.
      - RSD, RQSD, RR: Allowed absolute error and allowed CPU time in seconds.
      - RRLT: Absolute tolerance and allowed CPU time in seconds.
      - ERKODES: Order of the ODE solver (either 5 or 7), absolute tolerance, and allowed CPU time in seconds.
      - IRKODES: Order of the ODE solver (either 3, 5, 7, 9, or 11), absolute tolerance, and allowed CPU time in seconds.
   (f) Define the grid of time abscissae t at which the measure has to be computed. (See Section 1.3, page 18.)
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected method can be used and exits providing an explanatory message if the method cannot.
- Let $\rho = \max_{i \in \Omega} \lambda_i t_{\max}$, where $t_{\max}$ is the largest value of t for which the measure has to be computed. For small values of ρ, method ERKODES can be faster than the remaining methods. For large values of ρ, methods RSD, RQSD, RR, RRLT, and IRKODES can be significantly faster than methods SR, SRRE, and ERKODES (the CPU time required by methods SR and SRRE is approximately directly proportional to ρ). For very large values of ρ, method IRKODES can be the fastest one. Recall, however, than methods RSD, RQSD, RR, and RRLT are less general than methods SR and SRRE, and that methods RRLT, ERKODES, and IRKODES do not provide strict error control.

- Selecting an appropriate regenerative state for methods RR and RRLT is a delicate issue. As a general rule, the regenerative state should be a state visited often by the randomized discrete-time Markov chain of X with randomization rate slightly larger than $\max_{i\in\Omega}\lambda_i$. When the choice is not very clear, the user should be aware that a "bad" selection for the regenerative state can degrade severely the performance of the methods. For failure/repair exact and bounding rewarded CTMCs of fault-tolerant systems with exponential failure and repair time distributions and repair in every state with failed components with failure rates much smaller than repair rates, a good choice for the regenerative state is the state without failed components. For other types of models for which a good selection for the regenerative state exists, see [5, 1, 6, 7].

- Method RRLT can be significantly less costly than the method RR when in the latter the computational cost of the second phase of the method (solution of the truncated transformed rewarded CTMC using method SR) dominates the computational cost of the first phase (generation of the truncated transformed rewarded CTMC).

- For methods RR and RRLT, the condition that there exist some transition rate from the regenerative state, r, to some state in $S-\{r\}$ (condition C4 on page 56) can be circumvented by adding a tiny transition rate $\lambda \leq (10^{-10}\varepsilon)/(2r_{\max}t_{\max})$, where ε is the allowed absolute error, $r_{\max} = \max_{i\in\Omega} r_i$, and $t_{\max}$ is the largest value of t for which the measure has to be computed, with a negligible impact on the measure non greater than $10^{-10}\varepsilon$.

At present, only one simulation method is available for estimating the measure. In the interaction with the user, that method is listed under the name "`Independent Realizations with Forced Transitions of the Averaged Reward Rate`". The method consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the standard normal approximation confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the state specified by means of the `start_state` construct and ends when the sum of the sampled sojourn times is non smaller than the time point of interest t. During a realization, a transition is forced whenever: 1) the number of transitions that have been forced up to that point throughout the simulation has not yet reached a user-given limit, 2) the current state, a, of the realization is neither absorbing nor vanishing, 3) the cumulative reward up to entry into state a is 0, and 4) $r_a^{(\mathrm{v})} = 0$. If $T_a < t$ denotes the sum of the sampled sojourn times in the states of the current realization up to entry into state a, the transition is forced by limiting the sojourn time in that state to $t - T_a$. This is achieved by changing the probability distribution function of the random variable "sojourn time in state a" from $F(u) = 1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})}u}$, $u \geq 0$, to $F'(u) = (1-\mathrm{e}^{-\lambda_a^{(\mathrm{v})}u})/(1-\mathrm{e}^{-\lambda_a^{(\mathrm{v})}(t-T_a)})$, $0 \leq u \leq t - T_a$. For each realization, the sample of the estimator is the reward accumulated over the time interval $[0, t]$ divided by t and multiplied, if one or more transitions have been forced, by a a factor that takes account of those forced transitions.

If $\mathcal{F}$ denotes the collection of (possibly repeated) states of the realization where transitions have been forced, that factor is $\prod_{b\in\mathcal{F}}(1-\mathrm{e}^{-\lambda_b^{(\mathrm{v})}(t-T_b)})$. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure $\mathrm{EARR}(t)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Estimate a measure using simulation`".
   (c) Choose the measure "`Expected Averaged Reward Rate (EARR(t))`".
   (d) Choose the simulation method "`Independent Realizations with Forced Transitions of the Averaged Reward Rate`".
   (e) Assign values to the parameters controlling the method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - Confidence level as a decimal value (e.g., 0.99).
      - Allowed relative half-width of the confidence interval.
      - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
      - Value of the seed or name of the file.
      - Allowed number of forced transitions.
      - Minimum number of realizations for which the estimator is positive.
      - Allowed CPU time in seconds (an integer value).
   (f) Introduce the value of the time point of interest $t > 0$.
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.
  - In a realization, there is a state in which two or more instantaneous actions get enabled.
  - In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.
  - In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
  - In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.

- In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
- In a realization, there is a self-transition.
- In a realization, there is a cycle of vanishing states.

4.4 Cumulative Reward Complementary Distribution

The measure is defined as the complementary distribution function of the random variable "reward accumulated by X over the time interval $[0, t]$". Formally,

$$\mathrm{CRCD}(t, s) = P\left[\int_0^t r_{X(\tau)}\,\mathrm{d}\tau > s\right].$$

It is assumed that the reward rate of each state of X is ≥ 0. That restriction can be circumvented by shifting otherwise the reward rates by a positive amount d so that the new reward rates $r'_i = r_i + d$ are all ≥ 0. The cumulative reward complementary distribution of X is related to the cumulative reward complementary distribution measure, $\mathrm{CRCD}'(t, s)$, of the rewarded CTMC with shifted reward rates by $\mathrm{CRCD}(t, s) = \mathrm{CRCD}'(t, s + td)$.

As an example of the measure, assume that X models a gracefully degradable system. Assume also that each state of X is assigned a reward rate equal to the rate at which the system performs its job in the state. Then, $\mathrm{CRCD}(t, s)$ is the probability that the amount of job the system has done in the time interval $[0, t]$ is greater than s.

Available numerical methods for computing this measure are:

- Method of Nabli and Sericola (NS).
- Bounding Transformation/Regenerative Transformation (BT/RT).
- Bounding Transformation/Bounding Regenerative Transformation (BT/BRT).

Method NS [27] (see also [28]) is based on the interpretation of X in terms of a Poisson process and a randomized discrete-time Markov chain (see Appendix F).

Methods BT/RT and BT/BRT [29]:

1. Are based on building a truncated transformed rewarded CTMC with reward rates 0 and 1 (of, hopefully, much smaller size than X). The truncated transformed rewarded CTMC is solved using Algorithm A of [30] for computing the measure "Interval Availability Complementary Distribution" (see Section 4.5).
2. Compute a lower bound, an upper bound, or both for $\mathrm{CRCD}(t, s)$.
3. Require the selection of a "regenerative" state. That selection is carried out by means of the function with predefined name and prototype `regstat` as explained in Section 4.1, page 41.

In addition, in method BT/BRT there is a control parameter, called equalization parameter in the interaction with the user, that allows to trade-off bounds tightness with computational cost.

Method NS can be used for any X. Methods BT/RT and BT/BRT are only applicable under some conditions. First, denoting $r_{\max} = \max_{i\in\Omega} r_i$, $r_{\min} = \min_{i\in\Omega} r_i$, and $r_{\text{fmax}} = \max_{i\in\Omega\,:\,r_i\neq r_{\max}} r_i$, both methods require the abscissa s to satisfy $r_{\min}\,t < s < r_{\max}\,t$ if the lower bound is to be computed and $r_{\text{fmax}}\,t < s < r_{\max}\,t$ if the upper bound is to be computed. Obviously, this implies $t > 0$ and $s > 0$. Second, X and the chosen regenerative state r have to fulfill the following conditions, where $\Omega_{\max} = \{i \in \Omega \,:\, r_i = r_{\max}\}$, $\Omega_{\text{fmax}} = \{i \in \Omega \,:\, r_i = r_{\text{fmax}}\}$, $\Omega_{\min} = \{i \in \Omega \,:\, r_i = r_{\min}\}$, $\overline{\Omega} = \Omega - \Omega_{\max} - \Omega_{\text{fmax}} - \Omega_{\min}$, $S_{\max} = S \cap \Omega_{\max}$, $S_{\text{fmax}} = S \cap \Omega_{\text{fmax}}$, $S_{\min} = S \cap \Omega_{\min}$, $\overline{S} = S \cap \overline{\Omega}$, $S'_{\max} = S_{\max} - \{r\}$, $S'_{\text{fmax}} = S_{\text{fmax}} - \{r\}$, $S'_{\min} = S_{\min} - \{r\}$, and $\overline{S}' = \overline{S} - \{r\}$:

- Method BT/RT:

  C1. The reward rates r_i, $i \in \Omega$ take at least three different values.

  C2. $\Omega = S$ or $\Omega = S \cup \{f\}$, where f is an absorbing state.

  C3. $\left|S\right| \geq 2$.

  C4. Either all states in S are transient or X has a single recurrent class of states $C \subset S$.

  C5. $\max_{i\in\Omega_{\max}} \lambda_i > 0$ and $\max_{i\in\Omega_{\text{fmax}}\cup\overline{\Omega}\cup\Omega_{\min}} \lambda_i > 0$.

  C6. $r \in S$ and, if X has a single recurrent class of states $C \subset S$, $r \in C$.

  C7. If $S'_{\max} \neq \emptyset$, $\lambda_{r,S'_{\max}} > 0$.

  C8. If $S'_{\max} \neq \emptyset$, $\alpha_{S'_{\text{fmax}}\cup\overline{S}'\cup S'_{\min}} > 0$ and $\alpha_{S'_{\max}} = 0$, $\lambda_{i,S'_{\max}} > 0$ for some $i \in S'_{\text{fmax}} \cup \overline{S}' \cup S'_{\min}$ with $\alpha_i > 0$.

- Method BT/BRT: Conditions C1 through C8 above and the condition:

  C9. $S'_{\max} \neq \emptyset$.

  Moreover, the equalization parameter, D_C, has to have a value satisfying $1 \leq D_C < \max_{i\in S'_{\max}} \lambda_i / \min_{i\in S'_{\max}} \lambda_i$. The bounds become tighter and computationally more costly as D_C increases.

To compute the measure $\text{CRCD}(t, s)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)

2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:

   (a) Assign values to the parameters of the model if any.

   (b) Choose the task "`Compute a measure using numerical methods`".

   (c) Choose the measure "`Cumulative Reward Complementary Distribution (CRCD(t,s))`".

   (d) Choose the numerical method.

   (e) Assign values to the parameters controlling the chosen numerical method. These parameters are, in the order in which the user will be prompted for them in the interaction:

- NS: Allowed absolute error and allowed CPU time in seconds (an integer value).
- BT/RT: Bounds that have to be computed (lower bound, upper bound or both), allowed absolute error, and allowed CPU time in seconds.
- BT/BRT: Bounds that have to be computed (lower bound, upper bound or both), equalization parameter (the larger the parameter, the tighter and more computationally costly the bounds will be), allowed absolute error, and allowed CPU time in seconds.

(f) Define the set of (t, s) pairs for which the measure has to be computed. Such a set must be given as a comma-separated list of pairs enclosed with parentheses. For instance, to compute the measure for $t = 1,000$, $s = 3$ and $t = 2,500$, $s = 22$, the user could type "`(1000,3), (2.5e2,2.2e1)`". The list needs not be sorted.

(g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected method can be used and exits providing an explanatory message if the method cannot.
- Neglecting round-off errors, in all methods the error with which the measure or bounds for it are computed is guaranteed to be non larger than the specified allowed absolute error.
- Method NS has much smaller computational cost in terms of CPU time for the case $r_{\text{fmax}}\, t \leq s < r_{\max}\, t$ than for the case $s < r_{\text{fmax}}\, t$. For the latter case, the computational cost of the method in terms of CPU time is often prohibitive.
- In method BT/RT, there exists a unique subset of states S for which conditions C2, C4, and C6 can be satisfied for a given selection of the regenerative state r: S must be Ω if X has no absorbing state or X has a single absorbing state a and $r = a$; S must be $\Omega - \{a\}$ if X has a single absorbing state a and $r \neq a$ or X has two absorbing states a, b and $b = r$; and, in any other case, no S exists for which conditions C2, C4, and C6 can all be satisfied. This makes it easy to check whether X with a given selection for the regenerative state r is covered by the method. Conditions C1 and C5 are mild, in the sense that, when these conditions are not satisfied, computation of $\text{CRCD}(t, s)$ or of bounds for $\text{CRCD}(t, s)$ can be reduced to simpler problems. Thus, when the reward rates of X are finite but take only two different values, $r_{\max}$ and $r_{\min}$, $\text{CRCD}(t, s)$ can be formulated in terms of the simpler interval availability complementary distribution measure (see Section 4.5), $\text{IAVCD}(t, p)$, using $\text{CRCD}(t, s) = \text{IAVCD}(t, (s/t - r_{\min})/(r_{\max} - r_{\min}))$. When condition C1 is satisfied but $\max_{i \in \Omega_{\max}} \lambda_i = 0$, lower an upper bounds for $\text{CRCD}(t, s)$ can be computed as $P[X^{\text{lb}}((1 - (s/t - r_{\min})/(r_{\max} - r_{\min}))t) \in \Omega_{\max}]$ and $P[X^{\text{ub}}((1 - (s/t - r_{\text{fmax}})/(r_{\max} - r_{\text{fmax}}))t) \in \Omega_{\max}]$, where X^{lb} and X^{ub} are described in detail in [29]. Similarly, assuming C1 satisfied but $\max_{i \in \Omega_{\text{fmax}} \cup \overline{\Omega} \cup \Omega_{\min}} \lambda_i = 0$, lower and upper bounds for $\text{CRCD}(t, s)$ can be computed as $P[X^{\text{lb}}(((s/t - r_{\min})/(r_{\max} - r_{\min}))t) \in \Omega_{\max}]$ and $P[X^{\text{ub}}(((s/t - r_{\text{fmax}})/(r_{\max} - r_{\text{fmax}}))t) \in \Omega_{\max}]$. Finally, conditions C7 and C8 can be circumvented by adding to X a tiny transition rate $\lambda \leq 10^{-10}\varepsilon/(2t_{\max})$, where ε is the allowed error and $t_{\max}$ is the largest time t at which bounds for $\text{CRCD}(t, s)$ have to be computed, with a negligible impact on $\text{CRCD}(t, s)$ non greater than $10^{-10}\varepsilon$.

- Conditions C7 and C9 imply that the regenerative state cannot be absorbing and, then, according to the discussion regarding the possibilities for S in the BT/RT method, in BT/BRT, the set S must include precisely the non-absorbing states. We point out that, when condition C9 is not satisfied, the BT/RT method will be relatively inexpensive for s close to $r_{\max}\, t$, obviating the need for a potentially more efficient method to compute looser bounds such as method BT/BRT.

- For exact and bounding failure/repair rewarded CTMCs of fault-tolerant systems with exponential failure and repair time distributions and repair in every state with failed components with failure rates much smaller than repair rates and a reward rate structure which is non-increasing with the collection of failed components, a good choice for the regenerative state is the state without failed components. For other types of models for which a good selection for the regenerative state exists, see [29].

At present, only one simulation method is available for estimating the measure. In the interaction with the user, that method is listed under the name "`Independent Realizations with Forced Transitions of the Cumulative Reward`". The method consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the state specified by means of the `start_state` construct and ends when the sum, T, of the sampled sojourn times is non smaller than the time point of interest t, or, being $T < t$, the sampled cumulative reward, R, is $> s$ or $\leq s - r_{\mathrm{ub}}(t-T)$, where r_{ub} is a user-given upper bound for the reward rates of the non-vanishing states of the rewarded CTMC. During a realization, a transition is forced whenever: 1) the number of transitions that have been forced up to that point throughout the simulation has not yet reached a user-given limit, 2) the current state, a, of the realization is neither absorbing nor vanishing, and 3) $r_a^{(\mathrm{v})} \leq (s - R_a)/(t - T_a)$, where T_a denotes the sum of the sampled sojourn times in the states of the current realization up to entry into state a and R_a denotes the sampled cumulative reward in the current realization up to entry into state a. The transition is forced by limiting the sojourn time in state a to $t_{\mathrm{h}(a)} = \min\{(R_a + r_{\mathrm{ub}}(t - T_a) - s)/(r_{\mathrm{ub}} - r_a^{(\mathrm{v})}), t - T_a\} > 0$. This is achieved by changing the probability distribution function of the random variable "sojourn time in state a" from $F(u) = 1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})} u}$, $u \geq 0$, to $F'(u) = (1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})} u})/(1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})} t_{\mathrm{h}(a)}})$, $0 \leq u \leq t_{\mathrm{h}(a)}$. For each realization, the sample of the estimator is 0 if the reward accumulated over the time interval $[0, t]$ is $\leq s$ and otherwise is 1 multiplied, if one or more transitions have been forced, by a factor that takes account of those forced transitions. If $\mathcal{F}$ denotes the collection of (possibly repeated) states of the realization where transitions have been forced, that factor is $\prod_{b \in \mathcal{F}} (1 - \mathrm{e}^{-\lambda_b^{(\mathrm{v})} t_{\mathrm{h}(b)}})$. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure $\mathrm{CRCD}(t, s)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)

2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:

   (a) Assign values to the parameters of the model if any.

   (b) Choose the task "`Estimate a measure using simulation`".

   (c) Choose the measure "`Cumulative Reward Complementary Distribution (CRCD(t,s))`".

   (d) Choose the simulation method '`Independent Realizations with Forced Transitions of the Cumulative Reward`".

   (e) Assign values to the parameters controlling the method. These parameters are, in the order in which the user will be prompted for them in the interaction:

      - Confidence level as a decimal value (e.g., 0.99).
      - Allowed relative half-width of the confidence interval.
      - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
      - Value of the seed or name of the file.
      - Allowed number of forced transitions.
      - Minimum number of realizations for which the estimator is positive.
      - Upper bound r_{ub} for the reward rates of the non-vanishing states of the rewarded CTMC.
      - Allowed CPU time in seconds (an integer value).

   (f) Specify the pair (t, s), $t > 0$, $0 < s < r_{\mathrm{ub}} \times t$. The pair must be enclosed with parentheses. For instance, to estimate the measure for $t = 1,000$ and $s = 3$, the user could type "`(1000,3)`".

   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- If the allowed number of forced transitions is set to a value > 0, the method uses the standard normal approximation confidence interval. Otherwise, the method uses the Clopper-Pearson confidence interval [31].

- An error occurs and the simulation is aborted with an explanatory message if:

  – In a realization, there is a state for which the function `check_state` returns 0.

  – One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.

  – In a realization, there is a state in which two or more instantaneous actions get enabled.

  – In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.

- In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
- In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
- In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
- In a realization, there is a self-transition.
- In a realization, there is a cycle of vanishing states.
- In a realization, there is a non-vanishing state whose reward rate is larger than r_{ub}.

4.5 Interval Availability Complementary Distribution

For the measure to be well-defined, the reward rate of each state of X must be either 0 or 1. States with reward rate equal to 1 are interpreted as states in which the system is up; states with reward rate equal to 0 are interpreted as states in which the system is down; and, with that interpretation, the measure is defined as the complementary probability distribution function of the random variable "fraction of time in the time interval $[0, t]$ spent by X in up states". Formally, being U the subset of up states:

$$\mathrm{IAVCD}(t, p) = P\left[\frac{\int_0^t \mathbf{1}_{X(\tau)\in U}\,\mathrm{d}\tau}{t} > p\right].$$

The measure can be considered a particular case of the $\mathrm{CRCD}(t, s)$ measure ($\mathrm{IAVCD}(t, p) = \mathrm{CRCD}(t, pt)$) when all reward rates are either 0 or 1. It is offered as an independent measure because specialized, more efficient numerical methods exist for computing the measure. It is assumed $t > 0$ and $0 < p < 1$.

Available numerical methods for computing this measure are:

- Algorithm A of Rubino and Sericola (RS).
- Algorithm with Two Randomization Rates (IAVCD-2RR).
- Regenerative Transformation (RT).
- Bounding Regenerative Transformation (BRT).

Method RS is based on the interpretation of X in terms of a Poisson process and a randomized discrete-time Markov chain (see Appendix F) and is described in [30].

Method IAVCD-2RR is based on a randomization construct with different randomization rates for the up and the down states and can be substantially faster than method RS when the maximum output rates of the up and down states are significantly different [32].

In method RT, a truncated transformed rewarded CTMC model is obtained from X of (hopefully) smaller size by characterizing with enough accuracy the behavior of X from a "regenerative" state till next hit of either that state or, if existing, an absorbing state, and if X has some initial probability distribution outside the regenerative and the absorbing states, the behavior of X till first

hit of the regenerative state or, if existing, hit of the absorbing state. The truncated transformed rewarded CTMC model is then solved using method RS. The regenerative state is selected by means of the function with predefined name and prototype `regstat` as explained in Section 4.1, page 41.

Method BRT [33] computes a lower bound, an upper bound, or both for $\mathrm{IAVCD}(t, p)$ and also requires the selection of a regenerative state r. The method combines: 1) a model transformation step in which the transition rates from up states different from the regenerative state and, if existing, the absorbing state are scaled, and 2) the solution of the CTMC model with scaled transition rates using method RT. The scaling is controlled by an equalization parameter D_C. The regenerative state has to be selected using the function with predefined name and prototype `regstat` as explained in Section 4.1, page 41.

Method RS can be used for any X. The remaining methods are less general and require the following conditions to hold, where r denotes the chosen regenerative state for methods RT and BRT:

- Method IAVCD-2RR: $U \neq \emptyset$, $D \neq \emptyset$, $\max_{i \in U} \lambda_i > 0$, and $\max_{i \in D} \lambda_i > 0$.
- Method RT: Denoting $D = \Omega - U$, $U_S = U \cap S$, $D_S = D \cap S$, $U'_S = U_S - \{r\}$, and $D'_S = D_S - \{r\}$,

  C1. $\Omega = S$ or $\Omega = S \cup \{f\}$, where f is an absorbing state;

  C2. $|S| \geq 2$;

  C3. either all states in S are transient of S has a single recurrent class of states $C \subset S$;

  C4. $U \neq \emptyset$ and $D \neq \emptyset$;

  C5. $\max_{i \in U} \lambda_i > 0$ and $\max_{i \in D} \lambda_i > 0$;

  C6. $r \in S$ and, if X has a single recurrent class of states $C \subset S$, $r \in C$;

  C7. if $U'_S \neq \emptyset$, $\lambda_{r,U'_S} > 0$;

  C8. if $U'_S \neq \emptyset$, $\alpha_{D'_S} > 0$ and $\alpha_{U'_S} = 0$, $\lambda_{i,U'_S} > 0$ for some $i \in D'_S$ with $\alpha_i > 0$.
- Method BRT: Conditions C1 through C9 above and the condition:

  C9. $U'_S \neq \emptyset$.

  Moreover, the equalization parameter D_C has to satisfy $1 \leq D_C < \max_{i \in U'_S} \lambda_i / \min_{i \in U'_S} \lambda_i$. The bounds become tighter and computationally more costly as D_C increases.

To compute the measure $\mathrm{IAVCD}(t, p)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:

   (a) Assign values to the parameters of the model if any.

(b) Choose the task "`Compute a measure using numerical methods`".

(c) Choose the measure "`Interval Availability Complementary Distribution (IAVCD(t,p))`".

(d) Choose the numerical method.

(e) Assign values to the parameters controlling the chosen numerical method. These parameters are, in the order in which the user will be prompted for them in the interaction:

- RS, IAVCD-2RR, RT: Allowed absolute error and allowed CPU time in seconds (an integer value).
- BRT: Bounds that have to be computed (lower bound, upper bound or both), equalization parameter (the larger the parameter, the more tight and computationally costly the bounds will be), allowed absolute error, and allowed CPU time in seconds.

(f) Define the set of (t, p) pairs for which the measure has to be computed. Such a set must be given as a comma-separated list of pairs enclosed with parentheses. For instance, to compute the measure for $t = 1,000$, $p = 0.9$ and $t = 2,500$, $p = 0.8$, the user could type "`(1000,0.9), (2.5e2,0.8)`". The list needs not be sorted.

(g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- Neglecting round-off errors, in all methods the error with which the measure or bounds for it are computed is guaranteed to be non larger than the specified allowed error.
- When the maximum output rates of up and down states are significantly different, method IAVCD-2RR can be significantly faster than method RS.
- Method RT can be significantly faster than method RS.
- Method BRT can be much faster than methods RS and RT.
- The conditions required by method IAVCD-2RR are mild in the sense that when any of them is not satisfied, computation of $\mathrm{IAVCD}(t, p)$ can be reduced to a simpler problem: If there are not up states, $\mathrm{IAVCD}(t, p) = 0$; if there are not down states, $\mathrm{IAVCD}(t, p) = 1$; if all up states are absorbing, $\mathrm{IAVCD}(t, p) = \mathrm{ETRR}(pt)$; and, if all down states are absorbing, $\mathrm{IAVCD}(t, p) = \mathrm{ETRR}((1 - p)t)$.
- In method RT, there exists a unique subset of states S for which conditions C1, C3 and C6 can be satisfied for a given selection of the regenerative state r: S must be Ω if X has no absorbing state or X has a single absorbing state a and $r = a$; S must be $\Omega - \{a\}$ if X has a single absorbing state a and $r \neq a$ or X has two absorbing states a, b and $b = r$; and, in any other case, no S exists for which conditions C1, C3 and C6 can all be satisfied. This makes it easy to check whether X with a given selection for the regenerative state r is covered by the method. Condition C5 is mild, in the sense that, when that condition is not satisfied, computation of $\mathrm{IAVCD}(t, p)$ can be reduced to a simpler problem: When all up states are absorbing, $\mathrm{IAVCD}(t, p) = \mathrm{ETRR}(pt)$ and, when all down states are absorbing,

$\mathrm{IAVCD}(t,p) = \mathrm{ETRR}((1-p)t)$. Finally, conditions C7 and C8 can be circumvented by adding to X a tiny transition rate $\lambda \leq 10^{-10}\varepsilon/(2t_{\max})$, where ε is the allowed error and $t_{\max}$ is the largest time t at which $\mathrm{IAVCD}(t,p)$ has to be computed, with negligible impact on $\mathrm{IAVCD}(t,p)$ non greater than $10^{-10}\varepsilon$.

- Conditions C7 and C9 imply that the regenerative state cannot be absorbing and, then, according to the discussion regarding the possibilities for S in the method RT, in BRT the set S must include precisely the non-absorbing states. We point out that, when condition C9 is not satisfied, method RT will be relatively inexpensive for p close to 1, obviating the need for a potentially more efficient method to compute bounds such as method BRT.

- For typical failure/repair exact and bounding CTMC models of fault-tolerant systems with increasing structure function, exponential failure and repair time distributions and repair in every state with failed components, a good choice for the regenerative state is the state without failed components. For other types of models for which a good selection for the regenerative state exists, see [34] for method RT and [33] for method BRT.

At present, only one simulation method is available for estimating the measure. In the interaction with the user, that method is listed under the name "`Independent Realizations with Forced Transitions of the Cumulative Up Time`". The method consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the specified by means of the `start_state` construct and ends when the sum, T, of the sampled sojourn times is non smaller than the time point of interest t, or, being $T < t$, the sum, T^{U}, of the sampled sojourn times in up states is $> pt$ or $T - T^{\mathrm{U}} \geq (1-p)t$. During a realization, a transition is forced whenever: 1) the number of transitions that have been forced up to that point throughout the simulation has not yet reached a user-given limit, 2) the current state, a, of the realization is neither absorbing nor vanishing, and 3) $r_a^{(\mathrm{v})} = 0$. If $T_a < t$ denotes the sum of the sampled sojourn times in the states of the current realization up to entry into state a and $T_a^{\mathrm{U}} < t$ denotes the sum of the sampled sojourn times in the up states of the current realization up to entry into state a, the transition is forced by limiting the sojourn time in state a to $t_{\mathrm{h}(a)} = \min\{T_a^{\mathrm{U}} + (1-p)t - T_a, t - T_a\}$. This is achieved by changing the probability distribution function of the random variable "sojourn time in state a" from $F(u) = 1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})}u}$, $u \geq 0$, to $F'(u) = (1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})}u})/(1 - \mathrm{e}^{-\lambda_a^{(\mathrm{v})}t_{\mathrm{h}(a)}})$, $0 \leq u \leq t_{\mathrm{h}(a)}$. For each realization, the sample of the estimator is 0 if the sum of the sampled sojourn times in up states during the time interval $[0,t]$ is $\leq pt$ and otherwise is 1 multiplied, if one or more transitions have been forced, by a factor that takes account of those forced transitions. If $\mathcal{F}$ denotes the collection of (possibly repeated) states of the realization where transitions have been forced, that factor is $\prod_{b\in\mathcal{F}}(1 - \mathrm{e}^{-\lambda_b^{(\mathrm{v})}t_{\mathrm{h}(b)}})$. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure IAVCD(t, p) for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Estimate a measure using simulation`".
   (c) Choose the measure "`Interval Availability Complementary Distribution (IAVCD(t,p))`".
   (d) Choose the simulation method '`Independent Realizations with Forced Transitions of the Cumulative Up Time`".
   (e) Assign values to the parameters controlling the method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - Confidence level as a decimal value (e.g., 0.99).
      - Allowed relative half-width of the confidence interval.
      - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
      - Value of the seed or name of the file.
      - Allowed number of forced transitions.
      - Minimum number of realizations for which the estimator is positive.
      - Allowed CPU time in seconds (an integer value).
   (f) Specify the pair (t, p), $t > 0$, $0 < p < 1$. The pair must be enclosed with parentheses. For instance, to estimate the measure for $t = 1,000$ and $p = 0.9$, the user could type "`(1000,0.9)`".
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- If the allowed number of forced transitions is set to a value > 0, the method uses the standard normal approximation confidence interval. Otherwise, the method uses the Clopper-Pearson confidence interval [31].
- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.
  - In a realization, there is a state in which two or more instantaneous actions get enabled.
  - In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.

- In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
- In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
- In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
- In a realization, there is a self-transition.
- In a realization, there is a cycle of vanishing states.
- In a realization, there is a non-vanishing state whose reward rate is neither 0 nor 1.

4.6 Expected Cumulative Reward Till Exit of a subset of states

For the measure to be well-defined, Ω must be of the form $\Omega = B \cup \{a\}$, $B \neq \emptyset$, where all states in the subset B are transient and a is an absorbing state. From a user's perspective, X must have a unique absorbing state. Such a state will be regarded as state a and the remaining states will make up the subset B.

The measure is defined as the expected value of the random variable "reward accumulated by X till exit of B". Formally:

$$\mathrm{ECRTE} = E\left[\int_0^T r_{X(t)}\,\mathrm{d}t\right],$$

where $T = \min\{t \geq 0 : X(t) \notin B\}$. It is assumed that the reward rate of each state of B is ≥ 0. However, rewarded CTMC models with negative reward rates in B can be dealt with by shifting the reward rates by a positive amount d so that the new reward rates $r'_i = r_i + d$, $i \in B$ are all ≥ 0. Calling ECRTE' the expected cumulative reward till exit of B of the model with shifted reward rates and calling MTTE the expected cumulative reward till exit of B of a rewarded CTMC with reward rates equal to 1 in all states in B (which has the meaning of mean time till exit of B), we have $\mathrm{ECRTE} = \mathrm{ECRTE}' - d \times \mathrm{MTTE}$. Rewarded CTMCs with impulse rewards $r_{i,j}$ which are earned each time X makes a transition from state i to state j can be also accommodated by adding the contribution $\lambda_{i,j} r_{i,j}$ to the reward rate associated with state i. The mapping requires redefining the measure as

$$\mathrm{ECRTE} = E\left[\text{reward accumulated by } X \text{ in } [0, T]\right]$$

and is justified in Appendix E

From a modeling point of view, X can be seen as a rewarded CTMC keeping track of the behavior till exit from B of a larger rewarded CTMC Y actually modeling the system under study. Formally, X can be defined from Y as

$$X(t) = \begin{cases} Y(t) & \text{if } Y(\tau) \in B,\ 0 \leq \tau \leq t\,, \\ a & \text{otherwise}\,. \end{cases}$$

The state diagram of X can be obtained from the state diagram of Y by deleting the states not in B, adding the absorbing state a, and directing to a the transition rates from states in B to states

not in B. The initial probability distribution of X is related to the initial probability distribution of Y by $\alpha_i = P[Y(0) = i]$, $i \in B$, $\alpha_a = P[Y(0) \notin B]$, and X has associated with the states $i \in B$ the same reward rates as Y. With that interpretation, ECRTE is just the expected value of the reward accumulated by Y till exit from B.

As an example of the ECRTE measure, assume that a rewarded CTMC Y models a fault-tolerant system that can be either up or down, that B is the subset of states of Y in which the system is up, and that each state in B has a reward rate equal to 1. Then, the value of the measure ECRTE for X would be the mean time to failure of the system.

Let $\boldsymbol{\tau} = (\tau_i)_{i \in B}$. As shown in Appendix E, computation of the measure involves solving the non-singular linear system

$$\boldsymbol{\tau}^{\mathrm{T}} \mathbf{A}^{B,B} = -(\boldsymbol{\alpha}^B)^{\mathrm{T}} . \tag{4.7}$$

Let $B_1, \ldots, B_n$ denote the transient classes of states of X, so that $B = \cup_{k=1}^{n} B_k$, In METFAC-2.1, if $n > 1$, the matrix $\mathbf{A}^{B,B}$ is permuted into block upper triangular form

$$\mathbf{A}^{B,B} = \begin{pmatrix} \mathbf{A}^{B_1,B_1} & \mathbf{A}^{B_1,B_2} & \ldots & \mathbf{A}^{B_1,B_n} \\ & \mathbf{A}^{B_2,B_2} & \ldots & \mathbf{A}^{B_2,B_n} \\ & & \ddots & \vdots \\ & & & \mathbf{A}^{B_n,B_n} \end{pmatrix} .$$

and the solution $\boldsymbol{\tau}^{\mathrm{T}} = ((\boldsymbol{\tau}^1)^{\mathrm{T}}, \ldots, (\boldsymbol{\tau}^n)^{\mathrm{T}})$ of (4.7) is obtained by solving, for increasing k starting at $k = 1$, the non-singular linear systems

$$(\boldsymbol{\tau}^k)^{\mathrm{T}} \mathbf{A}^{B_k,B_k} = -(\boldsymbol{\alpha}^{B_k})^{\mathrm{T}} - \sum_{j=1}^{k-1} (\boldsymbol{\tau}^j)^{\mathrm{T}} \mathbf{A}^{B_j,B_k}, \quad 1 \le k \le n . \tag{4.8}$$

Available numerical methods for computing this measure are the same as those offered to solve the non-singular linear systems involved in the computation of the measure ESSRR (Section 4.2, pages 47–49):

- LU Decomposition (LUD).
- Gauss-Seidel (GS).
- Block Gauss-Seidel (BGS).
- Adaptive Successive Overrelaxation (ASOR).
- Adaptive Generalized Minimal Residual (AGMRES).
- Accelerated Gauss-Seidel (acc_GS).
- Accelerated Adaptive Successive Overrelaxation (acc_ASOR).

Method BGS requires the user to identify the blocks of states using the function with predefined name and prototype `block` as explained on page 47 save that now, a block consists of all the states of the transient class of states of X for which the function returns the same value.

Methods acc_GS and acc_ASOR require the user to identify pivot states by means of the function with predefined name and prototype `pivot` as explained on page 49.

To compute the measure ECRTE for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Compute a measure using numerical methods`".
   (c) Choose the measure "`Expected Cumulative Reward Till Exit of a subset of states (ECRTE)`".
   (d) Choose the numerical method.
   (e) If the chosen method is GS, BGS, ASOR, acc_GS, or acc_ASOR, set the relative tolerance to solve each linear system; if the chosen method is AGMRES, set the absolute tolerance to solve each linear system.
   (f) Set the allowed CPU time in seconds (an integer value).[9]
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected method can be used and exits providing an explanatory message if it cannot.
- For all methods save LUD, the non-singular linear systems actually solved are

$$\mathbf{P}^{B_k,B_k}\mathbf{x}^k = -\big(\boldsymbol{\alpha}^{B_k} + \sum_{j=1}^{k-1} \left(\mathbf{A}^{B_j,B_k}\right)^{\mathrm{T}} \left(\mathrm{diag}\left(\mathbf{A}^{B_j,B_j}\right)\right)^{-1} \mathbf{x}^j\big), \quad 1 \le k \le n\,, \tag{4.9}$$

where $\mathbf{P}^{B_k,B_k} = \left(\mathbf{A}^{B_k,B_k}\right)^{\mathrm{T}} \left(\mathrm{diag}\left(\mathbf{A}^{B_k,B_k}\right)\right)^{-1}$. The solutions of the linear systems (4.8) are obtained from those of (4.9) using $\boldsymbol{\tau}^k = \left(\mathrm{diag}\left(\mathbf{A}^{B_k,B_k}\right)\right)^{-1} \mathbf{x}^k$. Each linear system (4.9) is solved as follows. Let $\mathbf{x}^{k,l} = (x_i^{k,l})_{i\in B_k}$ denote the solution vector at iteration l, $l \ge 0$, and let ε be a user-given tolerance. The iterations start with $\mathbf{x}^{k,0} = \mathbf{1}$. For methods GS, BGS, ASOR, acc_GS, and acc_ASOR, the stopping criterion is

$$\max_{i\in B_k} \left(\frac{\left|x_i^{k,l} - x_i^{k,l-j}\right|}{\left|x_i^{k,l}\right|} \right) \le \varepsilon\,,$$

with $j = 5$ for $l < 100$, $j = 10$ for $100 \le l < 500$, $j = 20$ for $500 \le l < 1000$, and $j = 50$ for $l \ge 1000$. Letting $\mathbf{q}^{k,l} = \boldsymbol{\alpha}^{B_k} + \sum_{j=1}^{k-1} \left(\mathbf{A}^{B_j,B_k}\right)^{\mathrm{T}} \left(\mathrm{diag}\left(\mathbf{A}^{B_j,B_j}\right)\right)^{-1} \widetilde{\mathbf{x}}^j$, where $\widetilde{\mathbf{x}}^j$ denotes the computed $\mathbf{x}^j$, $1 \le j \le k-1$, the stopping criterion for method AGMRES is

$$\left\|\mathbf{q}^{k,l} + \mathbf{P}^{B_k,B_k}\mathbf{x}^{k,l}\right\|_2 \le \frac{\varepsilon}{\left\|\left(\mathrm{diag}\left(\mathbf{A}^{B_k,B_k}\right)\right)^{-1}\right\|_{\mathrm{F}} \left\|\mathrm{diag}\left(\mathbf{A}^{B_k,B_k}\right)\right\|_\infty} \times \left(\left\|\mathbf{q}^{k,l}\right\|_2 + \left\|\mathbf{P}^{B_k,B_k}\right\|_{\mathrm{F}} \left\|\mathbf{x}^{k,l}\right\|_2\right).$$

[9] This value sets an (approximate) upper limit for the CPU time that can be spent in solving all the involved linear systems.

- Neglecting round-off errors, if there is only one transient class of states in B and the chosen method is GS, the actual relative error with which the measure is computed can be expected to be proportional to the user-specified relative tolerance.
- Methods GS and acc_GS are guaranteed to converge. Method AGMRES can stagnate and never reach a solution with the specified relative tolerance and can also break down because of numerical instability.
- For typical failure/repair rewarded CTMC models of fault-tolerant systems with exponential failure and repair time distributions and repair in every state with failed components, there should only be a single class of transient states in B. For those models, it is strongly advised the use of either acc_GS or acc_ASOR with a single pivot state equal to the state without failed components, since for such models those methods with that pivot state will be much faster than both GS and ASOR and the former will exhibit very slow convergence. For other types of models, ASOR is usually faster than GS.

At present, only one simulation method is available for estimating the measure. In the interaction with the user, that method is listed under the name "`Independent Realizations of the Expected Cumulative Reward Till Exit`". The method consists in sampling realizations of the embedded DTMC of $X^{(\mathrm{v})}$ (see Appendix F). The simulation is carried out until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the standard normal approximation confidence interval is non larger than a user-given value. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the state specified by means of the `start_state` construct and ends when the absorbing state a is hit. If $\mathcal{V}$ denotes the collection of (possibly repeated) non vanishing states of the realization, the sample of the estimator is $\sum_{i\in\mathcal{V}} r_i^{(\mathrm{v})}(\lambda_i^{(\mathrm{v})})^{-1}$. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure ECRTE for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Estimate a measure using simulation`".
   (c) Choose the measure "`Expected Cumulative Reward Till Exit of a subset of states (ECRTE)`".
   (d) Choose the simulation method "`Independent Realizations of the Expected Cumulative Reward Till Exit`"
   (e) Assign values to the parameters controlling the method. These parameters are, in the order in which the user will be prompted for them in the interaction:

- Confidence level as a decimal value (e.g., 0.99).
- Allowed relative half-width of the confidence interval.
- Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
- Value of the seed or name of the file.
- Minimum number of realizations for which the estimator is positive.
- Allowed CPU time in seconds (an integer value).

(f) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.
  - In a realization, there is a state in which two or more instantaneous actions get enabled.
  - In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0.
  - In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
  - In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
  - In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
  - In a realization, there is a self-transition.
  - In a realization, there is a cycle of vanishing states.
  - The state specified by means of the `start_state` construct is absorbing.

4.7 Cumulative Reward Distribution Till Exit of a subset of states

For the measure to be well-defined, Ω must be of the form $\Omega = B \cup \{a\}$, $B \neq \emptyset$, where all states in the subset B are transient and a is an absorbing state. From a user's perspective, X must have a unique absorbing state. Such a state will be regarded as state a and the remaining states will make up the subset B.

The measure is defined as the probability distribution function of the random variable "reward accumulated by X till exit of B". Formally,

$$\mathrm{CRDTE}(s) = P\left[\int_0^T r_{X(t)}\,\mathrm{d}t \leq s\right],$$

where $T = \min\{t \geq 0 : X(t) = a\}$.

Let $B_+ = \{i \in B : r_i > 0\}$ and $B_- = \{i \in B : r_i < 0\}$. If $B_- = \emptyset$ and $s < 0$, we have $\mathrm{CRDTE}(s) = 0$. If $B_- = \emptyset$ and $s \geq 0$, the measure can be formalized as a particular case of the measure $\mathrm{ETRR}(t)$ considered in Section 4.1 for a modified rewarded CTMC X' with state space $B_+ \cup \{a\}$. The CTMC X' is obtained from X by: 1) "deleting" the states in $B_0 = \{i \in B : r_i = 0\}$ using Gaussian elimination without subtractions as described in [35], and 2) dividing by r_i the elements $c_{i,j}$, $i \in B_+$ of the infinitesimal generator of the rewarded CTMC with states in B_0 deleted so that holding times in the new CTMC X' have identical probability distributions to the rewards earned in those holding times in the CTMC with states in B_0 deleted and unscaled infinitesimal generator elements. Then, $\mathrm{CRDTE}(s) = P[X'(s) = a]$, which is equal to the measure $\mathrm{ETRR}(s)$ of X' with reward rate structure $r'_a = 1$, $r'_i = 0$, $i \neq a$. The mapping has been justified in [36].

Available numerical methods for computing the measure $\mathrm{CRDTE}(s)$ are those offered to compute the measure $\mathrm{ETRR}(t)$ (Section 4.1, pages 40–42) that do not require X to be irreducible and a method to deal with the case $B_+ \neq \emptyset$, $B_- \neq \emptyset$:

- Standard Randomization (SR).
- Standard Randomization with control of the Relative Error (SRRE).
- Randomization with Quasistationarity Detection (RQSD).
- Regenerative Randomization (RR).
- Regenerative Randomization with Laplace Transform Inversion (RRLT).
- Bounding Regenerative Randomization (BRR).
- Explicit Runge-Kutta ODE Solver (ERKODES).
- Implicit Runge-Kutta ODE Solver (IRKODES).
- Algorithm for Positive and Negative Reward Rates (CRDTE-PNRR).

Method CRDTE-PNRR is based on a randomization construct with different randomization rates for the states in the sets B_+ and B_-. The method is described in detail in [35].

The available methods can only be applied if the following conditions are fulfilled:

- Methods SR, SRRE: $B_- = \emptyset$.
- Method RQSD: $B_- = \emptyset$ and the CTMC X' satisfies the conditions given in Section 4.1, page 42 with $S = B_+$, $f_1 = a$, and $A = 1$. (These conditions amount to the fact that in X', the states in B_+ make up a transient class of states and $\boldsymbol{\alpha}^{B_+} > 0$.)
- Methods RR, RRLT: $B_- = \emptyset$ and the CTMC X' and the regenerative state r satisfy the conditions given in Section 4.1, page 42 with $S = B_+$, $f_1 = a$, and $A = 1$. (These conditions require $\left|B_+\right| \geq 2$ and $r \in B_+$.)

- Method BRR: $B_- = \emptyset$ and the CTMC X' and the regenerative state r satisfy the conditions given in Section 4.1, page 42 with $S = B_+$ $f_1 = a$, and $A = 1$. (These conditions require $|B_+| \geq 2$ and $r \in B_+$.) Moreover, denoting by λ'_i the output rate of state i in X', the equalization parameter D must satisfy $1 \leq D < \max_{i \in B_+ - \{r\}} \lambda'_i / \min_{i \in B_+ - \{r\}} \lambda'_i$. The bounds become tighter and computationally more costly as D increases.
- Methods ERKODES, IRKODES: $B_- = \emptyset$.
- Method CRDTE-PNRR: $B_- \neq \emptyset$ and $B_+ \neq \emptyset$.

To compute the measure $\mathrm{CRDTE}(s)$ for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)
2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:
   (a) Assign values to the parameters of the model if any.
   (b) Choose the task "`Compute a measure using numerical methods`".
   (c) Choose the measure "`Cumulative Reward Distribution Till Exit of a subset of states (CRDTE(s))`".
   (d) Choose the numerical method.
   (e) Assign values to the parameters controlling the chosen numerical method. These parameters are, in the order in which the user will be prompted for them in the interaction:
      - SR: Allowed absolute error and allowed CPU time in seconds (an integer value).
      - SRRE: Allowed relative error and allowed CPU time in seconds.
      - RSD, RQSD, RR: Allowed absolute error and allowed CPU time in seconds.
      - RRLT: Absolute tolerance.
      - BRR: Bounds that have to be computed (lower bound, upper bound or both), equalization parameter (the larger the parameter, the tighter and more computationally costly the bounds will be), allowed absolute error, and allowed CPU time in seconds.
      - ERKODES: Order of the ODE solver (either 5 or 7), absolute tolerance, and allowed CPU time in seconds.
      - IRKODES: Order of the ODE solver (either 3, 5, 7, 9, or 11), absolute tolerance, and allowed CPU time in seconds.
      - CRDTE-PNRR: Allowed absolute error and allowed CPU time in seconds.
   (f) Define the grid of time abscissae s at which the measure has to be computed. (See Section 1.3, page 18.)
   (g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- The tool checks automatically whether the selected method can be used and exits providing an explanatory message if the method cannot.
- For methods SR, SRRE, RQSD, RR, RRLT, BRR, ERKODES, and IRKODES, the measure is obtained as the value of $\mathrm{ETRR}(s)$ for the CTMC X' with its reward rate structure ($r'_a = 1$, $r'_i = 0$, $i \neq a$).
- The case $B_- \neq \emptyset$, $B_+ = \emptyset$, which is not covered by any of the available methods, can be dealt with by reversing the sign of the negative reward rates so that the new reward rates $r^*_i = |r_i|$, $i \in B$ are all ≥ 0. Calling $\mathrm{CRDTE}^*(s)$ the probability distribution function of the reward accumulated till exit of B of the model with modified reward rates r^*_i, we have [35]
$$\mathrm{CRDTE}(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 1 - \mathrm{CRDTE}^*(-s) & \text{if } s < 0 \end{cases} .$$
- Neglecting round-off errors, in methods SR, SRRE, RQSD, RR, BRR, and CRDTE-PNRR the error with which the measure or bounds for it are computed is guaranteed to be non greater than the specified allowed error. In the remaining methods, the absolute error with which the measure is computed can be larger than the specified absolute tolerance.
- Let $\rho = \max_{i \in \Omega'} \lambda_i s_{\max}$, where Ω' denotes the state space of X' and $s_{\max}$ is the largest value of s for which the measure has to be computed. For small values of ρ, method ERKODES can be faster than the remaining methods. For large values of ρ, methods RQSD, RR, RRLT, BRR, and IRKODES can be significantly faster than methods SR, SRRE, and ERKODES (the CPU time required by methods SR and SRRE is approximately directly proportional to ρ). For very large values of ρ, method IRKODES can be the fastest one.
- Selecting an appropriate regenerative state for methods RR, RRLT, and BRR is a delicate issue. As a general rule, the regenerative state should be a state visited often by the randomized discrete-time Markov chain of X' with randomization rate slightly larger than $\max_{i \in \Omega'} \lambda_i$. When the choice is not very clear, the user should be aware that a "bad" selection for the regenerative state can degrade severely the performance of the methods. See [5, 1, 6, 7] for some types of models for which a good selection for the regenerative state exists.[10]
- The method RRLT can be significantly less costly than the method RR when in the latter the computational cost of the second phase of the method (solution of the truncated transformed rewarded CTMC using method SR) dominates the computational cost of the first phase (generation of the truncated transformed rewarded CTMC).

The available simulation methods for estimating the measure are:

- Independent Realizations with Forced Transitions of the Cumulative Reward Till Exit (IRFTCRTE).

[10] The statements made in these references should be applied to the CTMC X'.

- Independent Realizations of the Cumulative Reward Till Exit (IRCRTE).

Method IRFTCRTE assumes $B_- = \emptyset$. The method consists in sampling realizations of $X^{(\mathrm{v})}$ until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the confidence interval is non larger than a user-given value. The samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC described in Appendix F. We recall that the specification of the initial probability distribution that can be optionally included in the model specification file (see Section 1.1.1, page 9) has *no effect* and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the state specified by means of the `start_state` construct and ends when the absorbing state a is hit or the sampled cumulative reward, R, is $> s$. During a realization, a transition is forced whenever: 1) the number of transitions that have been forced up to that point throughout the simulation has not yet reached a user-given limit, 2) the current state, c, of the realization is not vanishing, 3) $r_c^{(\mathrm{v})} > 0$, and 4) $R_c < s$, where R_c denotes the sampled cumulative reward in the current realization up to entry into state c. The transition is forced by limiting the sojourn time in state c to $t_{\mathrm{h}(c)} = (s - R_c)/r_c^{(\mathrm{v})}$. This is achieved by changing the probability distribution function of the random variable "sojourn time in state c" from $F(u) = 1 - \mathrm{e}^{-\lambda_c^{(\mathrm{v})} u}$, $u \geq 0$, to $F'(u) = (1 - \mathrm{e}^{-\lambda_c^{(\mathrm{v})} u})/(1 - \mathrm{e}^{-\lambda_c^{(\mathrm{v})} t_{\mathrm{h}(c)}})$, $0 \leq u \leq t_{\mathrm{h}(c)}$. For each realization, the sample of the estimator is 0 if $R > s$ and otherwise is 1 multiplied, if one or more transitions have been forced, by a factor that takes account of those forced transitions. If $\mathcal{F}$ denotes the collection of (possibly repeated) states of the realization where transitions have been forced, that factor is $\prod_{d \in \mathcal{F}} (1 - \mathrm{e}^{-\lambda_d^{(\mathrm{v})} t_{\mathrm{h}(d)}})$. The method uses the 2002-version of the RNG MT19937 [17].

Method IRCRTE does not impose any restriction on the reward rates of the states in B but is intended for the case $B_+ \neq \emptyset, B_- \neq \emptyset$. The method consists in sampling realizations of X until: 1) the number of realizations for which the estimator of the measure is positive is non smaller than a user-given value, and 2) the relative half-width of the Clopper-Pearson [31] confidence interval is non larger than a user-given value. Again, the samples are generated using the interpretation of $X^{(\mathrm{v})}$ in terms of its embedded DTMC and the initial probability distribution *always used* is: Initial probability equal to 1 for the state specified by means of the `start_state` construct and equal to 0 for the remaining states. Accordingly, each realization starts at the start state and ends when the absorbing state a is hit. For each realization, the sample of the estimator is 0 if $R > s$ and is 1 otherwise. The method uses the 2002-version of the RNG MT19937 [17].

To estimate the measure CRDTE for a model named, say, "`model`", the user has to follow the following steps:

1. Compile the model by typing "`m2build model`". (See Section 1.2, page 14.)

2. Execute the model by typing "`model.exe`" (see Section 1.3, page 16) and, then, in the ensuing interaction:

   (a) Assign values to the parameters of the model if any.

   (b) Choose the task "`Estimate a measure using simulation`".

(c) Choose the measure "`Cumulative Reward Distribution Till Exit of a subset of states (CRDTE(s))`".

(d) Choose the simulation method.

(e) Assign values to the parameters controlling the chosen simulation method. These parameters are, in the order in which the user will be prompted for them in the interaction:

- IRFTCRTE:
  - Confidence level as a decimal value (e.g., 0.99).
  - Allowed relative half-width of the confidence interval.
  - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation (see Section 1, page 17).
  - Value of the seed or name of the file.
  - Allowed number of forced transitions.
  - Minimum number of realizations for which the estimator is positive.
  - Allowed CPU time in seconds (an integer value).
- IRCRTE:
  - Confidence level as a decimal value (e.g., 0.99).
  - Allowed relative half-width of the confidence interval.
  - Whether the RNG is to be initialized using a seed or the state of the RNG is to be read from a file generated in a previous simulation.
  - Value of the seed or name of the file.
  - Minimum number of realizations for which the estimator is positive.
  - Allowed CPU time in seconds.

(f) Introduce the value of the abscissa of interest s, which for method IRFTCRTE must be positive.

(g) Choose between verbose and concise output. (See Section 1.3, page 19.)

Comments:

- If $B_- = \emptyset$, both methods can be used. However, the user is strongly advised to use method IRFTCRTE because it can be much faster than method IRCRTE.
- If the chosen method is IRFTCRTE and the allowed number of forced transitions is set to a value > 0, the method uses the standard normal approximation confidence interval. Otherwise, the method uses the Clopper-Pearson confidence interval.
- An error occurs and the simulation is aborted with an explanatory message if:
  - In a realization, there is a state for which the function `check_state` returns 0.
  - One or more instantaneous actions get enabled in the state specified by means of the `start_state` construct.
  - In a realization, there is a state in which two or more instantaneous actions get enabled.

- In a realization, there is a vanishing state whose reward rate is not a finite C `double` or a non-vanishing state whose reward rate is not a finite C `double` ≥ 0 (method IRFTCRTE) or not a finite C `double` (method IRCRTE).
- In a realization, there is a state in which it gets enabled a (non instantaneous) action whose rate is not a finite C `double` > 0.
- In a realization, there is a state in which it gets enabled a response whose probability is not > 0 and ≤ 1.
- In a realization, there is a state in which it gets enabled a (non instantaneous) action-response pair such that the product of the action's rate and the response's probability is not a finite C `double` > 0.
- In a realization, there is a self-transition.
- In a realization, there is a cycle of vanishing states.
- The state specified by means of the `start_state` construct is absorbing.

Appendix A

Installing the Tool

This appendix describes how to install the tool and how to make it accessible to a user. To install and use the tool, the programming environment must provide:

1. A C compiler compliant with the ISO/IEC 9899:1999 standard (C99) with enabled support for the IEC 60559:1989 floating-point standard (also designated as ANSI/IEEE 754-1985).

2. A C-shell (`csh`).

3. The utilities `ar` to create and modify archives, `ranlib` to generate indices to archives, `nm` to list symbols from object files, and `grep` to search for a pattern in a file.

4. The functions `getrusage` to measure CPU times, `sigemptyset` and `sigaction` to handle signals, and `timer_create`, `timer_settime`, and `timer_delete` to handle timers, all with the syntax and semantics described in the POSIX.1-2001 standard (IEEE Std 1003.1-2001).

The tool is provided as a tar file named `metfac21.tar.gz` that has been compressed with the `gzip` utility. To install METFAC-2.1, first set up three environment variables named `METFAC2`, `METFAC2_CC`, and `METFAC2_CC_FLAGS`. The environment variable `METFAC2` must be set to hold the path to the directory where the tool is to be installed. That directory must exist and you must have writing permission on it. The variable `METFAC2_CC` must be set to hold the path to the C compiler. Finally, the variable `METFAC2_CC_FLAGS` must be set to hold appropriate flags for the C compiler. These flags should at least enable the support for the C99 standard and that the mathematical and the *librt* (realtime) library are searched in the linking stage of the compilation process.[1]

After setting the above variables, unfold the file `metfac21.tar.gz` by typing

```
gzip -d metfac21.tar.gz
tar -xvf metfac21.tar
```

and next type

```
chmod u+x m2setup
m2setup
```

[1]For the GNU compiler `gcc`, the flag enabling support for the C99 standard is "`-std=c99`" (or "`-std=iso9899:1999`"); the flag that enables that the mathematical library is searched in the linking stage is "`-lm`"; and the flag that enables that the *librt* library is searched in the linking stage is "`-lrt`".

You should get several messages as the installation proceeds and, finally, the message

```
++m2setup terminated successfully
 (The following files should be present in ...:
 preproc
 libslvr.a
 met_block.o
 met_check_state.o
 met_pivot.o
 met_regstat.o
 met_subset.o
 m2build
 guide.pdf
 README
 INSTALL)
```

informing you that the installation has terminated successfully. In that case, the following files should be present in the directory pointed to by the environment variable METFAC2:

- preproc: preprocessing module of the tool. (See Section 1, page 15 and Appendix B, page 83.)
- libslvr.a, met_block.o, met_check_state.o, met_pivot.o, met_regstat.o, met_subset.o: model-independent core of the tool.
- m2build: utility for model compilation. (See Section 1, page 14.)
- guide.pdf: this document in Portable Document Format.
- README: brief description of the tool.
- INSTALL: the installation instructions contained in this section.

The tool is made accessible to a user by first setting the environment variables METFAC2, METFAC2_CC, and METFAC2_CC_FLAGS as previously explained and next adding the string $METFAC2 to the user's path.

Comments:

- The installation utility m2setup is a C-shell script and hence the need of that shell to install the tool. The script assumes that the full path to the shell is /bin/csh and makes use of the C compiler and the following system utilities: ar, ranlib, pwd, cp, mv, and rm. If the full path to the shell is not /bin/csh or any of those utilities is not in the user's path, the installation of the tool will fail.
- The model compilation utility m2build is also a C-shell script and thus the need of that shell to use the tool. The script assumes that the full path to the shell is /bin/csh and makes use of the C compiler and the following system utilities: ar, nm, and grep. If the full path to the shell is not /bin/csh or any of those utilities is not in the user's path, model compilation will fail.

Appendix B

The Preprocessor

The analysis and translation into C code of a model specification file (usually called *name*.`spec`) has been implemented in METFAC-2.1 as a separate module called `preproc`. The module can be invoked from outside the tool on a model specification file, say `spec_file`, by typing

```
preproc spec_file
```

If the model specification file is correct, `preproc` will generate two files called, by default, `model.c` and `model.h`. These files contain C code described in, respectively, Section B.1 and Section B.2. The names and, up to some extend, the contents of these output files can be changed by invoking `preproc` with appropriate flags as explained below.

The first argument of `preproc` must be the name of the model specification file. The following optional flags can be given next in any order: `-c`, `-e`, `-h`, `-n`, `-p`, `-v`, `-w`, and `-y`. Flags `-c`, `-h`, `-n`, and `-p` must be followed by a non-empty name. Flag `-y` is for maintenance. The effect of the remaining flags is as follows.

`-c file_name`	Creates file `file_name` instead of `model.c`.
`-e`	Introduces some changes in the above file or `model.c`. (See Section B.1.)
`-h file_name`	Creates file `file_name` instead of `model.h`.
`-n model_name`	Sets to `model_name` the name of the model that will be returned by the appropriate function defined in either `model.c` or the file whose name is specified by means of the flag `-c`. (See Section B.1.)
`-p file_name`	Name of the file to which lexical and syntactical error messages and information messages are printed. The default is to print those messages to the standard error stream.
`-v`	Prints the version number of the preprocessor to the standard output stream and exits.
`-w`	Suppresses warning messages.

Neither `spec_file` nor any of the names given after the flags `-c`, `-h`, `-n`, or `-p` may begin with a dash (`-`).

By typing

```
preproc -flags
```

`preproc` prints to the standard output stream detailed information about its usage and exits.

Apart from checking the model specification file to be lexically, syntactically, and semantically correct, `preproc` performs some checks in order to increase the reliability of the specification, issuing warning messages in some cases. Specifically, `preproc` will issue a warning message in the following cases:

1. The two operands of any of the operators "`>`", "`>=`", "`<`", "`<=`", "`==`", "`!=`", "`&&`", or "`||`" are of different type.

2. The right-hand side of the operator "`/=`" is of type `int`. (This might result in undesired truncation.)

3. Both operands of the operator "`/`" are of type `int`. (This might result in undesired truncation.)

4. A model parameter is declared but not referenced.

5. A state variable only appears in the `start_state` construct.

6. A state variable never appears in any `next_state` construct.

If the model specification file has any error, no output file is generated apart from the file where the user has instructed `preproc` to print error messages through the `-p` flag as explained before.

B.1 File `model.c`

The file contains 1) "#include" directives for the C header files `string.h`, `float.h`, and `math.h`, 2) declarations of the model-specific functions *actually* referenced within the model specification, and 3) definitions of several C functions detailed next. In the description of each such C function that follows, we will use **input** and **output** to distinguish between input and output parameters, and will denote by n_{sv} the number of state variables of the model specification file, by $n_{\text{ac\_insac}}$ the number of actions plus the number of instantaneous actions of the model specification file, and by n_{res} the number of responses of the model specification file. Also, it must be taken into account that:

- Actions, instantaneous actions, and responses without identifier are assigned the identifier "nolabel".

- Actions and instantaneous actions are indexed 1, 2, ... in the order they appear in the model specification file. (I.e., actions are *not* indexed separately from instantaneous actions.)

- Responses are indexed 1, 2 ... within each action in the order they appear in the model specification file. Actions without responses and instantaneous actions are always dealt with as having one response with identifier "nolabel", probability 1, and index 1.

The C functions defined in the file `model.c` are the following:

```
void time_stamp_(char *date_[], char *time_[])
```

returns the time and date on which the model specification file was compiled

`date_[]`	(**output**) pointer to an array of characters holding the compilation date as given by the C macro `__DATE__`; the array is static and therefore should not be freed after calling the function
`time_[]`	(**output**) pointer to an array of characters holding the compilation time as given by the C macro `__TIME__`; the array is static and therefore should not be freed after calling the function

```
extern char *modname(void)
```

returns a pointer to an array of characters that holds the name given after the flag `-n` or the name of the model specification file if the preprocessor was invoked without that flag; the array is static and therefore should not be freed after calling the function

```
void
modinfo_(int *n_ipar_, int *n_dpar_, int *n_sv_, int *n_func_, int *n_act_,
         int *n_insact_, int *n_resp_, char **parsymb_[], int *partype_[],
         char **svsymb_[], char **proto_[], char **actsymb_[],
         int *actinsflag_[], int *nrespofact_[], char **respsymb_[])
```

returns information about the model

`n_ipar_`	(**output**) number of parameters declared as `int`
`n_dpar_`	(**output**) number of parameters declared as `double`
`n_sv_`	(**output**) number of state variables (i.e., n_{sv})
`n_func_`	(**output**) number of external functions
`n_act_`	(**output**) number of actions plus number of instantaneous actions (i.e., $n_{\text{ac\_insac}}$)
`n_insact_`	(**output**) number of instantaneous actions
`n_resp_`	(**output**) number of responses (i.e., n_{res})
`parsymb_[]`	(**output**) `NULL` if the model does not have parameters; otherwise, pointer to an array that holds, starting at location 0, pointers to arrays of characters holding the identifiers of the model parameters, given in the order they have been declared; the array is static and therefore should not be freed after calling the function

`partype_[]` (**output**) `NULL` if the model does not have parameters; otherwise, pointer to an array that holds in location i, $0 \leq i \leq$ `n_ipar_+n_dpar_` $- 1$, the value 0 if the parameter `parsymb_`$[i]$ has been declared as `int` and the value 1 otherwise; the array is static and therefore should not be freed after calling the function

`svsymb_[]` (**output**) pointer to an array that holds, starting at location 0, pointers to arrays of characters holding the identifiers of the state variables, given in the order they have been declared; the array is static and therefore should not be freed after calling the function

`proto_[]` (**output**) `NULL` if the model does not have external functions; otherwise, pointer to an array that holds, starting at location 0, pointers to arrays of characters holding the prototypes of these functions, given in the order in which the functions have been declared; the array is static and therefore should not be freed after calling the function

`actsymb_[]` (**output**) pointer to an array that holds, starting at location 0, pointers to arrays of characters holding the identifiers of the actions and the instantaneous actions, given in the order in which actions and instantaneous actions appear in the model specification file; the array is static and therefore should not be freed after calling the function

`actinsflag_[]` (**output**) pointer to an array that holds at location i, $0 \leq i <$ `n_act_`, the value 1 if `actsymb_`$[i]$ corresponds to an instantaneous action and the value 0 otherwise; the array is static and therefore should not be freed after calling the function

`nrespofact_[]` (**output**) pointer to an array that holds, starting at location 0, the number of responses of each action and instantaneous action, given in the order in which actions and instantaneous actions appear in the model specification file; the array is static and therefore should not be freed after calling the function

`respsymb_[]` (**output**) pointer to an array that holds, starting at location 0, pointers to arrays of characters holding the identifiers of responses of actions and instantaneous actions, given in the order in which actions and responses appear in the model specification file; the array is static and therefore should not be freed after calling the function

```
void
succes_(int ipar_[], double dpar_[], int sv_[], int *naact_,
        int *nainsact_, int aact_[], int aactinsflag_[],
        int ptar_[], int aresp_[], int ptfs_[], int fsv_[])
```

obtains the list of states reached from a given state

`ipar_[]` (**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared

`dpar_[]` (**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

`*naact_` (**output**) number of actions plus number of instantaneous actions enabled (active) in the given state

`*nainsact_` (**output**) number of instantaneous actions enabled in the given state

`aact_[]` (**output**) in locations 0 through `*naact_` -1, indices of actions and instantaneous actions enabled in the given state ordered from lower to higher; the array `aact_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}}$

`aactinsflag_[]` (**output**) in location i, $0 \leq i \leq$ `*nainsact_` -1, the value 1 if `aact_`$[i]$ is an instantaneous action and the value 0 otherwise; the array `aactinsflag_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}}$

`ptar_[]` (**output**) in locations 0 through `*naact_`, locations in the array `aresp_[]` below of the first enabled response of the actions and instantaneous actions enabled in the given state; the array `ptar_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}} + 1$

`aresp_[]` (**output**) in locations `ptar_`$[i-1]$ through `ptar_`$[i]-1$, $1 \leq i \leq$ `*naact_`, indices of enabled responses of the ith action or instantaneous action enabled in the given state, ordered from lower to higher; the array `aresp_[]` must exist before calling the function and its size must be $\geq n_{\text{res}}$

`ptfs_[]` (**output**) in locations 0 through `*naact_` -1, locations in the array `fsv_[]` below of the the descriptions (in terms of values of the state variables) of the states reached from the given state; the array `ptfs_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}}$

`fsv_[]` (**output**) in locations `ptfs_`$[i-1]+(j-1)*n_{\text{sv}}+k-1$, $1 \leq k \leq n_{\text{sv}}$, with $1 \leq i \leq$ `*naact_` and $1 \leq j \leq$ `ptar_`$[i]-$`ptar_`$[i-1]$, values of the state variables (in the order they have been declared) for the state reached from the given state as a result of the jth enabled response of the ith action or instantaneous action enabled in the given state; the array `fsv_[]` must be allocated before calling the function and its size must be $\geq n_{\text{res}} \times n_{\text{sv}}$

`void start_(int ipar_[], double dpar_[], int sv_[])`

obtains the description of the state specified through the `start_state` construct

`ipar_[]`	(**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared
`dpar_[]`	(**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared
`sv_[]`	(**output**) starting at location 0, values of the state variables for the state specified through the `start_state` construct, given in the order they have been declared; the array `sv_[]` must exist before calling the function and its size must be $\geq n_{\text{sv}}$

`double`
`rates_(int ipar_[], double dpar_[], int sv_[], int act_, int *err_)`

returns the rate of a given action or instantaneous action in a given state (if the action is instantaneous, the returned value is the one yielded by the C macro `INFINITY`), or the (special) value yielded by the C99 function `nan` if the index of the action or instantaneous action is not correct

`ipar_[]`	(**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared
`dpar_[]`	(**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared
`sv_[]`	(**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared
`act_`	(**input**) index of the action or instantaneous action
`*err_`	(**output**) 1 if the index of the action or instantaneous action is not correct, i.e., either `act_` < 1 or `act_` $> n_{\text{ac\_insac}}$; otherwise, 0

`double reward_(int ipar_[], double dpar_[], int sv_[])`

returns the reward rate of a given state

`ipar_[]`	(**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared
`dpar_[]`	(**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

```
double initprob_(int ipar_[], double dpar_[], int sv_[])
```

returns the initial probability of a given state as specified by means of the `initial_probability` construct; if the construct was absent from the model specification file, the function returns 1 for the state specified through the `start_state` construct and 0 for the remaining states

`ipar_[]` (**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared

`dpar_[]` (**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

```
double prob_(int ipar_[], double dpar_[], int sv_[], int act_, int resp_,
             int *err_)
```

returns the probability of a given response of a given action or instantaneous action in a given state, or the (special) value yielded by the C99 function `nan` if the index of the action or instantaneous action or the index of the response are not correct

`ipar_[]` (**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared

`dpar_[]` (**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

`act_` (**input**) index of the action or instantaneous action

`resp_` (**input**) index of the response of the action or instantaneous action

`*err_` (**output**) 1 if the index of the action or instantaneous action is not correct, i.e., either `act_` < 1 or `act_` $> n_{\text{ac\_insac}}$, or the index of the response is not correct, i.e., `resp_` < 1, or `resp_` is larger than the number of responses of the action or instantaneous action; otherwise, 0

The following C functions are defined only if the preprocessor is invoked with the flag `-e`.

```
void
actresp_(int ipar_[], double dpar_[], int sv_[], int *naact_,
         int *nainsact_, int aact_[], int aactinsflag_[], int ptar_[],
         int aresp_[])
```

obtains for a given state the list of actions and instantaneous actions enabled (active) in the state and the list of enabled responses of each such action or instantaneous action

`ipar_[]`	(**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared
`dpar_[]`	(**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared
`sv_[]`	(**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared
`*naact_`	(**output**) number of actions plus number of instantaneous actions enabled in the given state
`*nainsact_`	(**output**) number of instantaneous actions enabled in the given state
`aact_[]`	(**output**) in locations 0 through `*naact_` $-$ 1, indices of the actions and instantaneous actions that are enabled in the given state, ordered from lower to higher; the array `aact_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}}$
`aactinsflag_[]`	(**output**) in location i, $0 \leq i \leq$ `*nainsact_` $-$ 1, the value 1 if `aact_`$[i]$ is an instantaneous action and the value 0 otherwise; the array `aactinsflag_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}}$
`ptar_[]`	(**output**) in locations 0 through `*naact_`, locations in the array `aresp_[]` below of the first enabled response of the actions and instantaneous actions enabled in the given state; the array `ptar_[]` must exist before calling the function and its size must be $\geq n_{\text{ac\_insac}} + 1$
`aresp_[]`	(**output**) in locations `ptar_`$[i-1]$ through `ptar_`$[i] - 1$, $1 \leq i \leq$ `*naact_`, indices of enabled responses of the ith action or instantaneous action enabled in the given state, ordered from lower to higher; the array `aresp_[]` must exist before calling the function and its size must be $\geq n_{\text{res}}$

```
int
ar_active_(int ipar_[], double dpar_[], int sv_[], int act_, int resp_)
```

returns 1 if a given action or instantaneous action and a given response of that action or instantaneous action are correct and enabled in a given state, 0 if the action or instantaneous action and the response are correct but not enabled in the given state, and -1 otherwise (i.e., the index of the action or instantaneous action, `act_`, is < 1 or $> n_{\text{ac\_insac}}$, or the index of the response, `resp_`, is < 1 or larger than the number of responses of the action or instantaneous action)

`ipar_[]` (**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared

`dpar_[]` (**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

`act_` (**input**) index of the action or instantaneous action

`resp_` (**input**) index of the response of the action or instantaneous action

```
int
successor_(int ipar_[], double dpar_[], int sv_[], int act_, int resp_,
           int fsv_[])
```

obtains the state reached from a given state as a result of a given action or instantaneous action and a given response of that action or instantaneous action *without* checking whether the action or instantaneous action and the response are actually enabled in the state; returns 1 if both the index, `act_`, of the action or instantaneous action, and the index, `resp_`, of the response are correct and 0 if they are not (i.e., `act_` < 1, `act_` $> n_{\text{ac\_insac}}$, `resp_` < 1, or `resp_` is larger than the number of responses of the action or instantaneous action)

`ipar_[]` (**input**) starting at location 0, values of the `int` parameters, given in the order they have been declared

`dpar_[]` (**input**) starting at location 0, values of the `double` parameters, given in the order they have been declared

`sv_[]` (**input**) starting at location 0, values of the state variables for the given state, given in the order they have been declared

`act_` (**input**) index of the action or instantaneous action

`resp_` (**input**) index of the response of the action or instantaneous action

`fsv_[]` (**output**) starting at location 0, values of the state variables (in the order they have been declared) for the state reached from the given state as a result of the action or instantaneous action with index `act_` and the response with index `resp_`; the array `fsv_` must exist before calling the function and its size must be $\geq n_{\text{sv}}$

B.2 File `model.h`

The file `model.h` defines: 1) the prototypes of all the model-specific functions declared in the model specification file regardless of whether they are actually referenced within the model specification file, and 2) a C macro called "`DECLARE_SYMBOLS`". Typesetting in typewriter font and enclosing by " and " literal text like "`this`", the contents of the file can be formally described in C-pseudocode as follows:

```
"#ifndef "str;
"#define "str;
""
 for (i=1; i<=n_efunc;i++) "extern "proto[i]";";
"#define DECLARE_SYMBOLS \"
 if (n_ipar>0) {
   "int "iparsymb[1]"=ipar[0]";
   for (i=2;i<=n_ipar;i++) {
     ",\";
     iparsymb[i]"=ipar["i-1"]";
   }
   ";\";
 }
 if (n_dpar>0) {
   "double "dparsymb[1]"=dpar[0]";
   for (i=2;i<=n_dpar;i++) {
     ",\";
     dparsymb[i]"=dpar["i-1"]";
   }
   ";\";
 }
 "int "svsymb[1]"=sv[0]";
 for (i=2;i<=n_sv;i++) {
   ",\";
   svsymb[i]"=sv["i-1"]";
 }
"#endif";
```

where

`str`	is "`model_h`" if `preproc` was invoked without the flag `-h` and otherwise is the string obtained by replacing each occurrence of a "`.`" by a "`_`" in the name given after that flag
`n_efunc`	is the number of model-specific functions
`n_ipar`	is the number of `int` parameters
`n_dpar`	is the number of `double` parameters
`n_sv`	is the number of state variables
`proto`[i]	is the prototype of the ith model-specific function
`iparsymb`[i]	is the identifier of the ith `int` parameter
`dparsymb`[i]	is the identifier of the ith `double` parameter

`svsymb[i]` is the identifier of the ith state variable

Appendix C

Rewarded CTMCs Description

Apart from computing and estimating a measure, METFAC-2.1 can also generate a description either verbose or compact in textual form of a rewarded CTMC. The verbose description, which is intended for model debugging, includes unreachable states,[1] vanishing states, self-transitions, and null transition rates, and is printed to the file *name*`.log` (*name* is a string that identifies the model). The compact description, which is intended for interfacing METFAC-2.1 with other tools, is generated after deleting the vanishing states of the model and is printed to a separate file called *name*`.ctmc`. In this section we describe the contents of the file *name*`.log` when the user selects generating a verbose description of a rewarded CTMC and the contents of the file *name*`.ctmc`.

C.1 Verbose Description

When the chosen task is to generate a verbose description of the rewarded CTMC, that description is printed in textual form to the file *name*`.log`. Let n_{states} denote the number of states of the rewarded CTMC. Typesetting in typewriter font and enclosing by " and " literal text like "`this`" and using the C99 functions `isfinite`, `isinf`, and `signbit`, the contents of the file can be formally described in C-pseudocode as follows:

```
"Model compiled on "d_c" at "t_c;
"Model executed on "d_e" at "t_e;
"";
"Parameters:"
"";
 if (n_par==0) "none";
 else for (i=1; i<=n_par; i+) parsym[i]"="parval[i];
"";
"Generation:"
"";
"CTMC characteristics:";
"(Warning: vanishing and unreachable states, self transitions, and ";
"null transition rates are included)"
 if (all_st_ok) {
```

[1] A non-vanishing state i of a CTMC X is said to be unreachable if $P[X(t) = i] = 0$ for all $t \geq 0$. A necessary and sufficient condition for a state to be unreachable is that its initial probability is null and there does not exist a path in the state diagram of the CTMC from a state with nonnull initial probability to that state.

```
    "# states="n_st;
    "(# vanishing_states="n_vst")";
    "# transition rates="n_tr;
  } else {
    "# states=n_st (aborted)";
    "(# vanishing_states=unknown")";
    "# transition rates=unknown";
  }
"";
"Spent time in s (user, system, total)="cput_gu", "cput_gs", "cput_gt;
""
"description of states";
"---------------------";
"";
 for (i=1; i<=n_st; i++) {
    "state="i;
    for (j=1; j<=n_sv; j++) {
      if (j<n_sv) svsym[j]"="svval[i][j]",";
      else svsym[j]"="svval[i][j];
    }
    "initial_probability="stinitpval[i];
    "reward_rate="strwdval[i];
    if (all_st_ok) {
      if (vanish[i]) "vanishing=yes";
      else "vanishing=no";
      if (absorb[i]) "absorbing=yes";
      else "absorbing=no";
    } else {
      "vanishing=unknown";
      "absorbing=unknown";
    }
    "block_index="stblkval[i];
    "pivot_flag="stpivval[i];
    "regenerative_flag="stregvali];
 }
""
"action-response and instantaneous action-response pairs";
"-------------------------------------------------------";
""
 for (i=1; i<=n_e_st; i++) {
    "state="i;
    if (n_actact[i]==0 || n_tot_actres[i]==0)
      "  no action-response or instantaneous action-response pair is enabled in this state";
    else {
      for (j=1; j<=n_actact[i]; j++) {
        if (!instflag[i][j]) "  action="actactlabel[i][j]" ("actact[i][j]")";
        else "  instantaneous_action="actactlabel[i][j]" ("actact[i][j]")";
        for (k=1; k<=n_actres[i][j]; k++) {
          "    response="actreslabel[i][j][k]" ("actres[i][j][k]")";
          if (trval[i][j][k]==0)
            "      next_state=NONE (NULL-RATE TRANSITION)";
          else if (i==stidxval[i][j][k])
            "      next_state=SAME (SELF-TRANSITION)";
          else {
```

```
            "      next_state="stidxval[i][j][k];
            if (instflag[i])
              "      transition_rate=+infinity";
            else if (isfinite(trval[i][j][k]))
              "      transition_rate="trval[i][j][k];
            else if (isinf(trval[i][j][k]))
              if (!signbit(trval[i][j][k]))
                "      transition_rate=+infinity";
              else
                "      transition_rate=-infinity";
            else
              "      transition_rate=not a number";
          }
        }
        if (n_actres[i][j]==0)
          "    response=none";
      }
    }
  }
  if (all_st_ok) {
    "# states="n_st;
    "(# absorbing_states="n_ast", # vanishing_states="n_vst")";
  } else
    "ABORTED!";
 "";
 "Total time in s (user, system, total)="cput_tu", "cput_ts", "cput_tt;
```

where

`d_c`	is the date on which the model was compiled
`t_c`	is the time of the day on which the model was compiled
`d_e`	is the date on which the model was executed
`t_e`	is the time of the day on which the model was executed
`n_par`	is the number of parameters of the model
`parsym[i]`	is the identifier of the ith parameter
`parval[i]`	is the value of the ith parameter
`all_st_ok`	is a flag with value 0 if the user-given limit on the number of states has been reached (Section 3, page 33), or the function `check_state` returned 0 for some state whether vanishing or not, or there were one or more instantaneous actions enabled in the state specified by means of the `start_state` construct, or there were two or more instantaneous actions enabled in a state different from the one specified by means of the `start_state` construct; otherwise, the flag has a non-zero value
`n_st`	is the number of generated states (`n_st` $= n_{\text{states}}$ if `all_st_ok` $\neq 0$ and `n_st` $\leq n_{\text{states}}$ otherwise)

`n_vst`	is the number of vanishing states
`n_tr`	is the number of transition rates
`cput_gu`	is the CPU time in user mode spent in generating the rewarded finite CTMC measured using the function `getrusage`
`cput_gs`	is the CPU time in system mode spent in generating the rewarded finite CTMC measured using the function `getrusage`
`cput_gt`	`cput_gu` + `cput_gs`
`svsymb[j]`	is the identifier of the jth state variable
`svval[i][j]`	is the value of the jth state variable for state i
`stinitpval[i]`	is the initial probability of state i
`strwdval[i]`	is the reward rate of state i
`vanish[i]`	is a flag with value 0 if state i is not vanishing and value $\neq 0$ otherwise
`absorb[i]`	is a flag with value 0 if state i is not absorbing and value $\neq 0$ otherwise
`stblkval[i]`	is the value returned for state i by the function with predefined name and prototype `block` (Section 4.2, page 47)
`stpivval[i]`	is "`yes`" if the function with predefined name and prototype `pivot` (Section 4.2, page 49) returned a value $\neq 0$ for state i and is "`no`" otherwise
`stregval[i]`	is "`yes`" if the function with predefined name and prototype `regstat` (Section 4.1, page 41) function returned a value $\neq 0$ for state i and is "`no`" otherwise
`n_e_st`	is the number of states for which the set of enabled action-response and instantaneous action-response pairs has been actually obtained; that number is equal to `n_st` if `all_st_ok` $\neq 0$, is equal to the user-given limit on the number of states if that limit has been reached and otherwise is equal to the index of the first state for which one of the following conditions hold: a) The function `check_state` returns 0 for the state; b) it is the state specified by means of the `start_state` construct and one or more instantaneous actions are enabled in it; or c) it is not the state specified by means of the `start_state` construct and two or more instantaneous actions are enabled in it
`n_actact`[i]	is the number of actions plus the number of instantaneous actions enabled in state i

`n_tot_actres`[i]	is the number of responses enabled in state i
`insflag`[i][j]	is a flag with value 0 if the jth action enabled in state i is instantaneous and value $\neq 0$ otherwise
`actactlabel`[i][j]	is the identifier of the jth action or instantaneous action enabled in state i
`actact`[i][j]	is the index of the jth action or instantaneous action enabled in state i
`actreslabel`[i][j][k]	is the identifier of the kth enabled response of the jth action or instantaneous action enabled in state i
`actres`[i][j][k]	is the index of the kth enabled response of the jth action or instantaneous action enabled in state i
`trval`[i][j][k]	is the transition rate resulting from action or instantaneous action `actact`[i][j] and response `actres`[i][j][k]
`stidxval`[i][j][k]	is the index of the state reached from state i through transition rate `trval`[i][j][k]
`n_ast`	is the number of absorbing states
`cput_tu`	is the CPU time in user mode spent in carrying out the task measured using the function `getrusage`
`cput_ts`	is the CPU time in system mode spent in carrying out the task CTMC measured using the function `getrusage`
`cput_tt`	`cput_tu` + `cput_ts`

Comments:

- Parameters and state variables are listed in the order in which they have been declared in the model specification file.
- Actions, instantaneous actions, and responses without identifier are assigned the identifier "nolabel."
- Actions and instantaneous actions are indexed 1, 2, ... in the order they appear in the model specification file. (I.e., actions are *not* indexed separately from instantaneous actions.)
- Responses are indexed 1, 2 ... within each action in the order they appear in the model specification file. Actions without responses and instantaneous actions are always dealt with as having one response with identifier "nolabel", probability 1, and index 1.

C.2 Compact Description

The file *name*`.ctmc` contanins a description of the state transition diagram, the initial probability distribution, and the reward rate structure of the rewarded CTMC with vanishing states deleted.

The contents of the file can be defined in C-pseudocode as follows:

```
n_st;
for (i=1; i<=n_st; i++) {
  stinitpval[i];
  strwdval[i];
 }
for (i=1; i<=n_st; i++) {
  n_succ[i];
  for (j=1; j<=n_succ[i]; j++) {
    succindex[i][j];
    transrate[i][j];
  }
 }
```

where

`n_st`	is the number of states, printed as a C `long`
`stinitpval[i]`	is the initial probability of state i, printed as a C `double`
`strwdval[i]`	is the reward rate of state i, printed as a C `double`
`n_succ`[i]	is the number of outgoing transition rates of state i, printed as a C `long`
`succindex`[i][j]	is the index, k, $k \neq i$, of the jth state such that there exists a non-null transition rate from state i to state k, printed as a C `long`
`transrate`[i][j]	is the transition rate from state i to state `succindex`$[i][j]$, printed as a C `double`

Comments:

- During the generation, it is checked that:
  - the function `check_state` returns a value $\neq 0$ for every state;
  - no instantaneous action gets enabled in the state specified by means of the `start_state` construct and at most one instantaneous action gets enabled in any other state;
  - the reward rate of every vanishing state is a finite C `double`;[2]
  - the rate of every (non instantaneous) action that gets enabled in a state is a finite C `double` > 0;
  - the probability of every response that gets enabled in a state is > 0 and ≤ 1;
  - for each (non instantaneous) action-response pair that gets enabled in a state, the product of the action's rate and the response's probability is a finite C `double` > 0;
  - there are not self-transitions; and
  - the initial probability of every state is ≥ 0 and ≤ 1.

  If any of the above tests fails, an error occurs and the generation is aborted with an explanatory message.

[2] I.e., it is neither infinite nor a "not-a-number."

Appendix D

Error and warning messages

This section describes the error and warning messages that can be generated when a model is executed.

D.1 Error Messages

There are three types of errors: 1) internal errors, which signal bugs of the tool, 2) resource errors, which signal exhaustion or unavailability of some resource, and 3) external errors (usually) resulting from erroneous input provided by the user. Any error produces one or more error messages and aborts the execution.

Messages associated with internal errors are printed to the standard error stream and consist of the header "`[metfac2 internal error]`" followed by a brief description of the error that includes the name and the line number of the module of the model-independent core of the tool where the error happened.

Resource errors have to do with memory exhaustion and failures in opening or closing a file, writing to or reading from a file, and deleting an auxiliary file. The corresponding messages are printed to the standard error stream and consist of the header "`[metfac2 resource error]`" followed by one of the following strings:

```
Not enough memory.

File could not be opened.

File could not be closed.

Unable to write to file.

Unable to read from file.

Unable to delete file.
```

In the last five cases the message also includes a portion `File: '...'`, indicating the name of the involved file.

Messages associated with external errors are printed to the standard output stream and appended to the file *name*`.log`, where *name* is a string that identifies the model. The message begins

with the header "`[metfac2 error ...]`" (the '`...`' stands for the error number) and is followed by one or more lines of text including a brief description of the error. We list next in alphabetical order the messages associated with external errors (excluding the header) together with their possible causes.

`Action rate is negative or zero.`

The rewarded CTMC includes a state such that the rate of one of the (non instantaneous) actions enabled in the state is ≤ 0. The message is always followed by the rate and index of the action and the values of the state variables for the state in which the action is enabled.

`Action rate is not finite.`

The rewarded CTMC includes a state such that the rate of one of the (non instantaneous) actions enabled in the state is not a finite C `double`.[1] The message is always followed by: 1) the "value" of the rate (either of: "`+infinity`", "`-infinity`", "`not a number`"), 2) the index of the action, and 3) the values of the state variables for the state in which the action is enabled.

`Initial probability of state is negative or larger than one.`

The rewarded CTMC includes a state whose initial probability is < 0 or > 1. The message is always followed by the values of the state variables for the state and the value of the initial probability of the state.

`Initial probability of state is not finite.`

The rewarded CTMC includes a state whose initial probability is not a finite C `double`. The message is always followed by the values of the state variables for the state and the "value" of the initial probability of the state (either of: "`+infinity`", "`-infinity`", "`not a number`").

```
Measure IAVCD is not-well defined.
Reward rate of state is neither zero nor one.
```

The rewarded CTMC includes a non-vanishing, reachable state whose reward rate is neither 0 nor 1. The message is always followed by the values of the state variables for the state and the value of the reward rate of the state.

```
Measures ECRTE, CRDTE are not well-defined.
CTMC does not have transient states.
```

The rewarded CTMC with vanishing states deleted does not include any transient state. (This may happen, for instance, if the rewarded CTMC with vanishing states deleted is irreducible.)

[1] I.e., it is either infinite or a "not-a-number."

`Measures ECRTE, CRDTE are not well-defined.`
`CTMC has two or more recurrent states.`

The rewarded CTMC with vanishing states deleted has two or more recurrent states. (This may happen, for instance, if there are two or more absorbing states.) If the rewarded CTMC with vanishing states deleted has any transient state, the message is followed by the list of recurrent states.

`Numerical method: computation failed.`

The chosen numerical method failed in computing the selected measure or one of the chosen numerical methods failed in solving either a singular or a non-singular linear system of equations involved in computing the measure. The message is always followed by a brief explanation that includes, in this order, 1) for the measures ESSRR and ECRTE, whose computation amounts to solving linear systems of equations, the string "`Singular linear system`" if the failure occurred in the course of solving a singular linear system and the string "`Non-singular linear system`" if the failure occurred in the course of solving a non-singular system; 2) the method's name according to Section 4; 3) for transformation-based methods, a string that tells the user whether the failure occurred when obtaining the truncated transformed model or when solving that model (the strings are "`transformation`" and "`solution`"); and 4) one of the following strings identifying the reason of the failure: "`absolute stagnation`", "`allowed CPU time exhausted`", "`integration step too small`", "'`numerical breakdown`", and "`LU decomposition of auxiliary linear system failed`".[2] Thus, for instance, assuming the user chose to compute the measure $\mathrm{ETRR}(t)$ using the numerical method "Regenerative Randomization" (Section 4.1, page 41) and that the allowed CPU time was too small to obtain the transformed rewarded CTMC given the allowed absolute error, the explanation that would appear after the message would be: "`Regenerative Randomization: transformation: allowed CPU time exhausted`".

`Numerical method: condition is not fulfilled.`

Some of the conditions required by one of the selected numerical methods is not fulfilled. The message is always followed by a brief explanation that includes, in this order, 1) for the measures ESSRR and ECRTE, whose computation amounts to solving linear systems of equations, the string "`Singular linear system`" if the conditions have to do with singular linear systems and the string "`Non-singular linear system`" if the conditions have to do with non-singular linear systems; 2) the method's name according to Section 4; and 3) a string that identifies the condition that does not hold according to Section 4. Thus, for instance, assuming the user chose to compute the measure $\mathrm{CRCD}(t, s)$ using the numerical method "Bounding Transformation/Regenerative Transformation" and that the chosen regenerative state did not belong to the S set, thus violating condition C6 of the method (Section 4.4, page 60), the explanation that would appear after the message would be: "`Bounding Transformation/Regenerative Transformation: C6`".

[2]The only method that can breakdown or stagnate is "Adaptive Generalized Minimal Residual" (Section 4.2, page 48); the only methods that may require solving an auxiliary linear system are "Accelerated Gauss-Seidel" and "Accelerated Adaptive Successive Overrelaxation" (Section 4.2, page 48).

`Numerical method: wrong information.`

The information provided by the user for one of the chosen numerical methods is not correct. The message is always followed by a brief explanation that includes, in this order, 1) for the measures ESSRR and ECRTE, whose computation amounts to solving linear systems of equations, the string "`Singular linear system`" if the information has to do with singular linear systems and the string "`Non-singular linear system`" if the information has to do with non-singular linear systems; 2) the method's name according to Section 4; 3) one of the following strings identifying the reason by which the provided information is not correct: "`in transient class of states no pivot state has been defined`", "`no regenerative state has been defined`", "`no regenerative state has been defined in the set of transient states with non null reward rate`", '`two or more regenerative states have been defined`", and "`two or more regenerative states have been defined in the set of transient states with non null reward rate`", and 4) if two or more regenerative states have been defined, the list of such states. Thus, for instance, assuming the user chose to compute the ECRTE measure using the numerical method "Accelerated Gauss-Seidel" (Section 4.2, page 48) and that he/she forgot to define a suitable `pivot` function and the default `pivot` function, returning 0 for every state, was used, the explanation that would appear after the message would be: "`Non-singular linear system: Accelerated Gauss-Seidel: in transient class of states no pivot state has been defined`".

`One or more instantaneous actions are enabled in the ‘‘start state’’.`

One or more instantaneous actions are enabled in the state specified by means of the `start_state` construct. The message is always followed by the values of the state variables for the state and the indices of the instantaneous actions that are enabled in it.

`One or more states are unreachable.`

The rewarded CTMC with vanishing states deleted includes one or more unreachable states.[3] The message is always followed by the number of unreachable states and the values of the state variables for each unreachable state.

`Response probability is negative, zero, or larger than one.`

The rewarded CTMC includes a state such that the probability of the response in one of the (non instantaneous) action-response pairs enabled in the state is ≤ 0 or > 1. The message is always followed by: 1) the value of the probability, 2) the index of the response, 3) the index of the action, and 4) the values of the state variables for the state in which the action-response pair is enabled.

[3] A non-vanishing state i of a CTMC X is said to be unreachable if $P[X(t) = i] = 0$ for all $t \geq 0$.

`Response probability is not finite.`

The rewarded CTMC includes a state such that the probability of the response in one of the (non instantaneous) action-response pairs enabled in the state is not a finite C `double`. The message is always followed by: 1) the "value" of the probability (either of: "`+infinity`", "`-infinity`", "`not a number`"), 2) the index of the response, 3) the index of the action, and 4) the values of the state variables for the state in which the action-response pair is enabled.

`Reward rate of state is negative.`

The rewarded CTMC includes a non-vanishing, reachable state whose reward rate is < 0. The message is always followed by the values of the state variables for the state and the value of the reward rate of the state.

`Reward rate of state is not finite.`

The rewarded CTMC includes a vanishing state or a non-vanishing, reachable state whose reward rate is not a finite C `double`. The message is always followed by the values of the state variables for the state and the "value" of the reward rate of the state (either of: "`+infinity`", "`-infinity`", "`not a number`").

`Self transition rate.`

The rewarded CTMC includes a state such that for one of the action-response or instantaneous action-response pairs enabled in the state, the values of the state variables do not change. The message is always followed by: 1) the values of the state variables for the state in which the action-response or instantaneous action-response pair is enabled, 2) the index of the response, and 3) the index of the action or the instantaneous action.

```
Simulation: condition is not fulfilled.
Reward rate of state is larger than user-given upper bound.
```

The reward rate of one of the states of a realization is larger than the user-given upper bound (see Section 4.4, page 63). The message is always followed by the values of the state variables for the state, the state's reward rate, and the value of the user-given upper bound.

```
Simulation: condition is not fulfilled.
State is absorbing.
```

One of the states of a realization or a regenerative cycle is absorbing. The message is always followed by: 1) the values of the state variables for the state, 2) the values of the state variables for the state of the realization or regenerative cycle from which the absorbing state was reached, and 3) the index of the action or the instantaneous action and the index of the response of the corresponding transition.

```
Simulation: condition is not fulfilled.
The ‘‘start state’’ is absorbing.
```

The state specified by means of the `start_state` construct is absorbing. The message is always followed by the values of the state variables for the state.

```
Simulation failed.
Allowed CPU time exhausted; confidence interval is wider than requested
```

The chosen simulation method and the allowed CPU time were such that it was possible to generate the required minimum number of realizations or regenerative cycles with positive estimator but it was not possible to meet the requested maximum width of the confidence interval.

```
Simulation failed.
Allowed CPU time exhausted; minimum number of realizations with positive
estimator not reached.
```

The chosen simulation method and the allowed CPU time were such that it was not possible to generate the required minimum number of realizations with positive estimator.

```
Simulation failed.
Allowed CPU time exhausted; minimum number of regenerative cycles with
positive estimator not reached.
```

The chosen simulation method and the allowed CPU time were such that it was not possible to generate the required minimum number of regenerative cycles with positive estimator.

```
The limit on the number of states has been reached.
```

The user-given limit on the number of states has been reached (see Section 3, page 33). The message is always followed by the value of that limit.

```
There is a cycle of vanishing states connected by instantaneous transitions
```

The rewarded CTMC includes a set of states mutually reachable via instantaneous actions. The message is always followed by the values of the states variables for the states that make up the set.

```
Transition rate is negative or zero.
```

The rewarded CTMC includes a state such that for one of the (non instantaneous) action-response pairs enabled in the state, the product of the action rate and the response probability is ≤ 0. The message is always followed by: 1) the value of the transition rate, 2) the values of the state variables for the state in which the action-response pair is enabled (the “from” state), 3) the index of the response, and 4) the index of the action.

`Transition rate is not finite.`

The rewarded CTMC includes a state such that for one of the (non instantaneous) action-response pairs enabled in the state, the product of the action rate and the response probability is not a finite C `double`. The message is always followed by: 1) the "value" of the transition rate (either of: "`+infinity`", "`-infinity`", "`not a number`"), 2) the index of the response, 3) the index of the action, and 4) the values of the state variables for the state in which the action-response pair is enabled.

`Two or more instantaneous actions are enabled in state.`

The rewarded CTMC includes a state different from the one specified by means of the `start_state` construct in which two or more instantaneous actions are enabled. The message is always followed by the values of the state variables for the state and the indices of the instantaneous actions that are enabled in it.

`Wrong input.`

Some input provided by the user during the interaction is not correct. The message is always followed by a brief description of the kind of input that was expected.

`Wrong state.`

The rewarded CTMC includes a state for which the function with predefined name and prototype `check_state` (Section 3, page 31) returns 0. The message is always followed by: 1) the values of the state variables for the wrong state, 2) the values of the state variables for the state from which the wrong one is reached,[4] and 3) the index of the action or instantaneous action and the index of the response of the corresponding transition.

D.2 Warning Messages

Warning messages are intended to inform the user of situations that are potentially dangerous as, for instance, the fact that a model specification file included an action that did not get enabled in any state. Warning messages are printed to the standard output stream and to the end of the file *name*`.log`. A warning message begins with the header `[metfac2 warning ...]` (the '`...`' stands for the warning number) and is followed by one or more lines of text giving the user a brief description of the warning. We list next in alphabetical order the messages associated with warnings (excluding the header) together with their possible causes.

```
Accelerated Adaptive Successive Overrelaxation: All states of
transient class of states have been defined as pivot states.
```

In the course of solving one of the non-singular linear systems of equations involved in the computation of the measures ESSRR or ECRTE using the method Accelerated Adaptive Successive Overrelaxation, it has turned out that in one of the transient classes of states of the

[4] Unless, of course, when the wrong state is the one specified by means of the `start_state` construct.

rewarded CTMC with vanishing states deleted the number of states that have been defined as pivot states using the function with predefined name and prototype `pivot` (Section 4.2, page 49) is equal to the number of states of the class.

```
Accelerated Gauss-Seidel: All states of transient class of states
have been defined as pivot states.
```

In the course of solving one of the non-singular linear systems of equations involved in the computation of the measures ESSRR or ECRTE using the method Accelerated Gauss-Seidel, it has turned out that in one of the transient classes of states of the rewarded CTMC with vanishing states deleted the number of states that have been defined as pivot states using the function with predefined name and prototype `pivot` (Section 4.2, page 49) is equal to the number of states of the class.

```
Action did not get enabled.
```

The model specification includes a (non instantaneous) action that did not get enabled in any state of the rewarded CTMC or, if the selected task was to estimate a measure using simulation, in any state of the realizations or regenerative cycles that have been generated. The message is always followed by the index of the action.

```
Block Gauss-Seidel: In recurrent class of states there are as many blocks
as states.
```

In the course of solving one of the singular linear systems of equations involved in the computation of the measures ESSRR and ECRTE using the method Block Gauss-Seidel, it has turned out that in one of the recurrent classes of states of the rewarded CTMC with vanishing states deleted the number of blocks of states defined by means of the function with predefined name and prototype `block` (Section 4.2, page 47) is equal to the number of states of the class.

```
Block Gauss-Seidel: In recurrent class of states there is only one block.
```

In the course of solving one of the singular linear systems of equations involved in the computation of the measures ESSRR and ECRTE using the method Block Gauss-Seidel, it has turned out that in one of the recurrent classes of states of the rewarded CTMC with vanishing states deleted the number of blocks of states defined by means of the function with predefined name and prototype `block` (Section 4.2, page 47) is just one.

```
Block Gauss-Seidel: In transient class of states there are as many blocks
as states.
```

In the course of solving one of the non-singular linear systems of equations involved in the computation of the measures ESSRR and ECRTE using the method Block Gauss-Seidel, it has turned out that in one of the transient classes of states of the rewarded CTMC with vanishing states deleted the number of blocks of states defined by means of the function with predefined name and prototype `block` (Section 4.2, page 47) is equal to the number of states of the class.

`Block Gauss-Seidel: In transient class of states there is only one block.`

In the course of solving one of the non-singular linear systems of equations involved in the computation of the measures ESSRR and ECRTE using the method Block Gauss-Seidel, it has turned out that in one of the transient classes of states of the rewarded CTMC with vanishing states deleted the number of blocks of states defined by means of the function with predefined name and prototype `block` (Section 4.2, page 47) is just one.

`Enabled action does not have any response enabled in state.`

The rewarded CTMC includes a state such that for a (non instantaneous) action that is enabled in the state, no response of that action is enabled in the state. The message is always followed by the index of the action and the values of the state variables for the state.

`Instantaneous action did not get enabled.`

The model specification includes an instantaneous action that did not get enabled in any state of the rewarded CTMC or, if the selected task was to estimate a measure using simulation, in any state of the realizations or regenerative cycles that have been generated. The message is always followed by the index of the instantaneous action.

`Response did not get enabled.`

The model specification includes a response that did not get enabled in any state of the rewarded CTMC or, if the selected task was to estimate a measure using simulation, in any state of the realizations or regenerative cycles that have been generated. The message is always followed by the index of the response and the index of the corresponding action.

`Reward rate of state is negative.`

The rewarded CTMC includes a state whose reward rate is negative. The message is always followed by the values of the state variables for the state and the state's reward rate and can only be issued when the chosen task is to generate a verbose description of the rewarded CTMC.

`Sum of the initial probabilities of the states is not one.`

The difference between 1 and the sum of the initial probabilities of the states of the rewarded CTMC with vanishing states is larger, in absolute value, than 50,000 times the "epsilon" constant of the underlying hardware.[5] The message is always followed by the value of the sum of the initial probabilities.

`Transient states with null reward rate have been deleted.`

The rewarded CTMC included one or more non vanishing, transient states whose reward rate was 0 and those states have been deleted. The message is always followed by the number of such states and can only be issued when the chosen task is to compute the measure "Cumulative Reward Distribution Till Exit of a subset of states".

[5] The "epsilon" constant is the difference between the smallest exactly representable number greater than 1 and 1.

`Vanishing states have been deleted.`

The rewarded CTMC included one or more vanishing states and those states have been deleted. The message is always followed by the number of such states.

```
Sum of probabilities of enabled responses of enabled action
is larger than one in state.
```

The rewarded CTMC includes a state such that for one of the (non instantaneous) actions enabled in the state, the sum of the probabilities of the responses of that action that are enabled in the state is larger than 1 by more than 1,000 times the "epsilon" constant of the underlying hardware. The message is always followed by: 1) the value of the sum of the probabilities, 2) the index of the action, 3) the indices of the enabled responses of the action, and 4) the values of the state variables for the state.

`The ‘‘start state’’ is absorbing.`

The state specified by means of the `start_state` construct is absorbing. The message is always followed by the values of the state variables for the state.

Appendix E

Mathematical Justifications

In this appendix, we justify: 1) the formalization of the computation of the measures ESSRR (Section 4.2, page 46) and ECRTE (Section 4.6, page 69) used in the tool, and 2) the extension to impulse rewards of the measures $\mathrm{ETRR}(t)$ (Section 4.1 on page 40), $\mathrm{EARR}(t)$ (Section 4.3 on page 54), and ECRTE. Throughout the appendix, we will use the same notation as in Section 4.

E.1 Formalization of the Computation of Some Measures

We start by justifying the formulation of the computation of the ESSRR measure in terms of the solution of some linear systems. The measure can be expressed as:

$$\mathrm{ESSRR} = \sum_{i \in \Omega} r_i p_i$$

where $p_i = \lim_{t \to \infty} p_i(t)$ is the steady-state probability of state i. Let S be the subset of transient states of X and let $C_1, C_2, \ldots, C_m$ be the recurrent classes of states of X. If S is empty, then states in different recurrent classes are inaccessible, and it is trivial that, for $1 \leq k \leq m$, $p_i = \alpha_{C_k} p_i^k$, $i \in C_k$, where $\alpha_{C_k} = \sum_{i \in C_k} \alpha_i$ and the column vectors $\mathbf{p}^k = (p_i^k)_{i \in C_k}$ are the 1-normalized ($\left\|\mathbf{p}^k\right\|_1 = 1$) solutions of the singular linear systems

$$(\mathbf{p}^k)^{\mathrm{T}} \mathbf{A}^{C_k, C_k} = \mathbf{0}^{\mathrm{T}}, \tag{E.1}$$

If S is not empty, then, it has been shown in [37] that $p_i = 0$, $i \in S$ and that for $1 \leq k \leq m$, $p_i = \gamma_k p_i^k$, $i \in C_k$, where γ_k is the probability that X will end up for $t \to \infty$ in C_k and p_i^k, $i \in C_k$, $1 \leq k \leq m$ can be computed as just described. If $m = 1$, X will end up in the single recurrent class C_1 with probability 1 and $\gamma_1 = 1$. If $m \geq 2$, then $\gamma_k = \alpha_{C_k} + \beta_k$, where β_k is the probability that X will exit S through C_k. Computation of the probabilities β_k, $1 \leq k \leq m$ can be formalized in terms of the solution of a non-singular linear system as follows. For $i, j \in S$, define $\tau_{i,j} = E[\int_0^\infty \mathbf{1}_{X(t)=j} \, \mathrm{d}t \mid X(0) = i]$, $i, j \in S$, i.e. $\tau_{i,j}$ is the expected time of X in j conditioned on the initial state being i. A standard one-step analysis using the interpretation of X in terms of the embedded discrete-time Markov chain (see Appendix F) allows us to write

$$\tau_{i,j} = \frac{\delta_{i,j}}{\lambda_i} + \sum_{\substack{l \in S \\ l \neq i}} \frac{\lambda_{i,l}}{\lambda_i} \tau_{l,j}, \quad i, j \in S,$$

where $\delta_{i,j} = 1$ for $i = j$ and $\delta_{i,j} = 0$ otherwise. We have, then,

$$\sum_{\substack{l \in S \\ l \neq i}} \lambda_{i,l} \tau_{l.j} - \lambda_i \tau_{i,j} = -\delta_{i,j}\,, \quad i, j \in S\,,$$

which, defining the matrix $\mathbf{T} = (\tau_{i,j})_{i,j \in S}$, can be written as

$$\mathbf{A}^{S,S}\mathbf{T} = -\mathbf{I}\,.$$

Now, we argue that $\mathbf{A}^{S,S}$ is non-singular. One way is to consider the transition matrix $\mathbf{P}$ of the randomized DTMC of X with a randomization rate $\Lambda \geq \max_{i \in \Omega} \lambda_i$ (see Appendix F). Classification of the states in the randomized DTMC is as in X. Denoting by $\mathbf{P}^{S,S}$ the restriction of $\mathbf{P}$ to the block $S \times S$, we have $\mathbf{A}^{S,S} = \Lambda(\mathbf{P}^{S,S} - \mathbf{I})$. But, the spectral radius of $\mathbf{P}^{S,S}$ satisfies $\rho(\mathbf{P}^{S,S}) < 1$ (see, for instance, [38, Chapter 8, Lemma 3.20]), and this implies that all eigenvalues of $\mathbf{A}^{S,S}$ have strictly negative real part, implying that $\mathbf{A}^{S,S}$ is non-singular. Being $\mathbf{A}^{S,S}$ non-singular, we have

$$\mathbf{T} = -(\mathbf{A}^{S,S})^{-1}\,.$$

Now, define $\tau_i = E[\int_0^\infty \mathbf{1}_{X(t)=i}\,\mathrm{d}t]$, $i \in S$, i.e. τ_i is the expected time of X in state i, and the column vector $\boldsymbol{\tau} = (\tau_i)_{i \in S}$. Clearly, $\boldsymbol{\tau}^{\mathrm{T}} = (\boldsymbol{\alpha}^S)^{\mathrm{T}}\mathbf{T}$, so that $\boldsymbol{\tau}^{\mathrm{T}} = -(\boldsymbol{\alpha}^S)^{\mathrm{T}}(\mathbf{A}^{S,S})^{-1}$ and $\boldsymbol{\tau}$ is the solution of the non-singular linear system

$$\boldsymbol{\tau}^{\mathrm{T}}\mathbf{A}^{S,S} = -(\boldsymbol{\alpha}^S)^{\mathrm{T}}\,. \tag{E.2}$$

It is easy to justify that $\tau_i = \int_0^\infty p_i(t)\,\mathrm{d}t$. To that end consider that

$$E\Big[\int_t^{t+\Delta t} \mathbf{1}_{X(t)=i}\,\mathrm{d}t\Big] = P[X(t) = i] \\ \times P[X \text{ has not made any transition in } [t, t+\Delta t] \mid X(t) = i](\Delta t + T)\,,$$

where

$$0 \leq T \leq \sum_{j \in \Omega} P[X(t) = j] P[X \text{ has made some transition in } [t, t+\Delta t] \mid X(t) = j]\Delta t\,.$$

But,

$$P[X \text{ has made some transition in } [t, t+\Delta t] \mid X(t) = j] = \int_0^{\Delta t} \lambda_j \mathrm{e}^{-\lambda_j \tau}\,\mathrm{d}\tau = o(1)\,,$$

where $o(f(\Delta t))$ denotes a function $h(\Delta t)$ with $\lim_{t \to 0} h(\Delta t)/f(\Delta t) = 0$. Then, we have

$$E\Big[\int_t^{t+\Delta t} \mathbf{1}_{X(t)=i}\,\mathrm{d}t\Big] = p_i(t)\mathrm{e}^{-\lambda_i \Delta t}\Delta t + o(\Delta t) = p_i(t)\Delta t + o(\Delta t)\,,$$

which implies $\tau_i = \int_0^\infty p_i(t)\,\mathrm{d}t$. Now, it is easy to see that $\beta_k = \boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\Lambda}^k$, $1 \leq k \leq m$, where $\boldsymbol{\Lambda}^k$ is the column vector $\boldsymbol{\Lambda}^k = (\lambda_{i,C_k})_{i \in S}$. This is equivalent to $\beta_k = \sum_{i \in S} \tau_i \lambda_{i,C_k} = \sum_{i \in S} \int_0^\infty p_i(t)\lambda_{i,C_k}\,\mathrm{d}t$, $1 \leq k \leq m$. But,

$$\begin{aligned} &P[X \text{ has made some transition from } S \text{ to } C_k \text{ in } [t, t+\Delta t]] \\ &\quad = \sum_{i \in S} P[X(t) = i] P[X \text{ has made some transition from } S \text{ to } C_k \text{ in } [t, t+\Delta t] \mid X(t) = i] \\ &\quad + T_i\,, \end{aligned}$$

where

$$\begin{aligned}0 \le T_i &\le P[X \text{ has made more than one transition in } [t, t+\Delta t] \mid X(t) = i] \\ &= \sum_{\substack{j\in\Omega \\ j\neq i}} \frac{\lambda_{i,j}}{\lambda_i} \int_0^{\Delta t} \lambda_i \mathrm{e}^{-\lambda_i \tau}(1 - \mathrm{e}^{-\lambda_j(\Delta t - \tau)})\,\mathrm{d}\tau = o(\Delta t)\,,\end{aligned}$$

and

$$\begin{aligned}&P[X \text{ has made some transition from } S \text{ to } C_k \text{ in } [t, t+\Delta t]] \\ &\quad = \sum_{i\in S} p_i(t) \frac{\lambda_{i,C_k}}{\lambda_i}(1 - \mathrm{e}^{-\lambda_i \Delta t}) + o(\Delta t) \\ &\quad = \sum_{i\in S} p_i(t)\lambda_{i,C_k}\Delta t + o(\Delta t)\,,\end{aligned}$$

implying $\beta_k = \sum_{i\in S}\int_0^\infty p_i(t)\lambda_{i,C_k}\,\mathrm{d}t$. Summarizing, we have justified that computing the ESSRR measure involves obtaining the 1-normalized solution of the singular linear systems (E.1) and, if X has transient states and more than one recurrent class of states, solving the non-singular linear system (E.2).

The formulation of the computation of the ECRTE measure in terms of the solution of a non-singular linear system can be justified as follows. For $i, j \in B$, define $\tau_{i,j} = E[\int_0^\infty \mathbf{1}_{X(t)=j}\,\mathrm{d}t \mid X(0) = i]$, i.e. $\tau_{i,j}$ is the expected time of X in j conditioned on the initial state being i. A standard one-step analysis using the interpretation of X in terms of the embedded discrete-time Markov chain (see Appendix F) allows us to write

$$\tau_{i,j} = \frac{\delta_{i,j}}{\lambda_i} + \sum_{\substack{l\in B \\ l\neq i}} \frac{\lambda_{i,l}}{\lambda_i}\tau_{l,j}\,, \quad i, j \in B\,,$$

where $\delta_{i,j} = 1$ for $i = j$ and $\delta_{i,j} = 0$ otherwise. We have, then,

$$\sum_{\substack{l\in B \\ l\neq i}} \lambda_{i,l}\tau_{l.j} - \lambda_i\tau_{i,j} = -\delta_{i,j}\,, \quad i, j \in B\,,$$

which, defining the matrix $\mathbf{T} = (\tau_{i,j})_{i,j\in B}$, can be written as

$$\mathbf{A}^{B,B}\mathbf{T} = -\mathbf{I}\,.$$

That $\mathbf{A}^{B,B}$ is non-singular can be argued from the fact that the states in B are transient as it was argued that matrix $\mathbf{A}^{S,S}$ was non-singular when discussing the formulation of the ESSRR measure above. Being $\mathbf{A}^{B,B}$ non-singular, we have

$$\mathbf{T} = -(\mathbf{A}^{B,B})^{-1}\,.$$

Now, define $\tau_i = E[\int_0^\infty \mathbf{1}_{X(t)=i}\,\mathrm{d}t]$, $i \in B$, i.e. τ_i is the expected time of X in state i, and the column vector $\boldsymbol{\tau} = (\tau_i)_{i\in S}$. Clearly, $\boldsymbol{\tau}^{\mathrm{T}} = (\boldsymbol{\alpha}^B)^{\mathrm{T}}\mathbf{T}$, so that $\boldsymbol{\tau}^{\mathrm{T}} = -(\boldsymbol{\alpha}^B)^{\mathrm{T}}(\mathbf{A}^{B,B})^{-1}$ and $\boldsymbol{\tau}$ is the solution of the non-singular linear system

$$\boldsymbol{\tau}^{\mathrm{T}}\mathbf{A}^{B,B} = -(\boldsymbol{\alpha}^B)^{\mathrm{T}}\,. \tag{E.3}$$

But, from the definitions of ECRTE and τ_i, $i \in B$, using the fact that a is absorbing and Fubini's theorem:

$$\begin{aligned}
\text{ECRTE} &= E\left[\int_0^T r_{X(t)}\,\mathrm{d}t\right] = E\left[\sum_{i\in B} r_i \int_0^T \mathbf{1}_{X(t)=i}\,\mathrm{d}t\right] = \sum_{i\in B} r_i E\left[\int_0^T \mathbf{1}_{X(t)=i}\,\mathrm{d}t\right] \\
&= \sum_{i\in B} r_i E\left[\int_0^\infty \mathbf{1}_{X(t)=i}\,\mathrm{d}t\right] = \sum_{i\in B} r_i \tau_i\,.
\end{aligned}$$

We have thus justified that computing the ECRTE measure involves solving the non-singular linear system (E.3).

E.2 Extension to Impulse Rewards of Some Measures

We start by justifying the extension to impulse rewards of the $\mathrm{ETRR}(t)$ measure. The justification is based on the facts that, for $j \neq i$,

$$\begin{aligned}
&P[X \text{ has made a single transition to } j \text{ in } [t, t+\Delta t] \mid X(t) = i] \\
&\quad = \frac{\lambda_{i,j}}{\lambda_i} \int_0^{\Delta t} \lambda_i \mathrm{e}^{-\lambda_i \tau} \mathrm{e}^{-\lambda_j(\Delta t - \tau)}\,\mathrm{d}\tau = \lambda_{i,j}\Delta t + o(\Delta t)\,,
\end{aligned}$$

where $o(f(\Delta t))$ denotes a function $h(\Delta t)$ with $\lim_{\Delta t \to 0} h(\Delta t)/f(\Delta t) = 0$, and

$$\begin{aligned}
&P[X \text{ has made } k \text{ transitions in } [t, t+\Delta t] \mid X(t) = i] \\
&= \sum_{\substack{j_1\in\Omega \\ j_1\neq i}} \sum_{\substack{j_2\in\Omega \\ j_2\neq j_1}} \cdots \sum_{\substack{j_k\in\Omega \\ j_k\neq j_{k-1}}} \frac{\lambda_{i,j_1}}{\lambda_i} \frac{\lambda_{j_1,j_2}}{\lambda_{j_1}} \cdots \frac{\lambda_{j_{k-1},j_k}}{\lambda_{j_{k-1}}} \\
&\qquad \times \int_0^{\Delta t} \int_{\tau_1}^{\Delta t} \cdots \int_{\tau_1+\cdots+\tau_{k-1}}^{\Delta t} \lambda_i \mathrm{e}^{-\lambda_i \tau_1} \lambda_{j_1} \mathrm{e}^{-\lambda_{j_1}\tau_2} \cdots \lambda_{j_{k-1}} \mathrm{e}^{-\lambda_{j_{k-1}}\tau_k} \mathrm{e}^{-\lambda_{j_k}(\Delta t - \sum_{l=1}^k \tau_l)} \\
&\qquad\qquad \mathrm{d}\tau_k \mathrm{d}\tau_{k-1} \cdots \mathrm{d}\tau_1 \\
&\leq (\max_{l\in\Omega} \lambda_l \Delta t)^k\,.
\end{aligned}$$

The justification is as follows. First, we can write

$$\begin{aligned}
&E[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]] \\
&= \sum_{i\in\Omega} P[X(t) = i] P[X \text{ has not left } i \text{ in } [t, t+\Delta t] \mid X(t) = i] r_i \Delta t \\
&+ \sum_{i\in\Omega} P[X(t) = i] \\
&\qquad \times \sum_{\substack{j\in\Omega \\ j\neq i}} P[X \text{ has made a single transition to } j \text{ in } [t, t+\Delta t] \mid X(t) = i](r_{i,j} + o(1)) \\
&+ \sum_{k=2}^{\infty} \sum_{i\in\Omega} P[X(t) = i] P[X \text{ has made } k \text{ transitions in } [t, t+\Delta t] \mid X(t) = i] \\
&\qquad \times \big(\text{reward accumulated by } X \text{ in } [t, t+\Delta t] \text{ given } X(t) = i \text{ and that } X \text{ has made} \\
&\qquad\quad k \text{ transitions in } [t, t+\Delta t]\big)\,.
\end{aligned}$$

But, the reward accumulated by X in $[t, t+\Delta t]$ given $X(t) = i$ and that X has made k transitions in $[t, t+\Delta t]$ is ≥ 0 and $\leq r_{\max}\Delta t + kr'_{\max}$, with $r_{\max} = \max_{i\in\Omega} r_i$ and $r'_{\max} = \max_{\substack{i,j\in\Omega\\ j\neq i}} r_{i,j}$, and the last term is ≥ 0 and non larger than

$$\sum_{i\in\Omega} P[X(t) = i] \sum_{k=2}^{\infty} (\max_{l\in\Omega} \lambda_l \Delta t)^k (r_{\max}\Delta t + kr'_{\max}) \,,$$

which is $o(\Delta t)$. Then, we have

$$\begin{aligned} &E[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]] \\ &\quad = \sum_{i\in\Omega} p_i(t) \mathrm{e}^{-\lambda_i \Delta t} r_i \Delta t + \sum_{i\in\Omega} p_i(t) \sum_{\substack{j\in\Omega\\ j\neq i}} (\lambda_{i,j}\Delta t + o(\Delta t))(r_{i,j} + o(1)) + o(\Delta t) \\ &\quad = \sum_{i\in\Omega} \Big(r_i + \sum_{\substack{j\in\Omega\\ j\neq i}} \lambda_{i,j} r_{i,j}\Big) p_i(t)\Delta t + o(\Delta t) \,, \end{aligned}$$

showing that

$$\lim_{\Delta t\to 0^+} \frac{E[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]]}{\Delta t} = \sum_{i\in\Omega} \Big(r_i + \sum_{\substack{j\in\Omega\\ j\neq i}} \lambda_{i,j} r_{i,j}\Big) p_i(t) \,, \tag{E.4}$$

which is $\mathrm{ETRR}(t)$ for the rewarded CTMC with added contributions to the reward rates of the states.

The extension to impulse rewards of the $\mathrm{EARR}(t)$ measure can be justified as follows. We have

$$E\Big[\frac{\text{reward accumulated by } X \text{ in } [0,t]}{t}\Big] = \frac{1}{t} E[\text{reward accumulated by } X \text{ in } [0,t]]$$

and, since (E.4), it follows that

$$\begin{aligned} &\frac{1}{t} E[\text{reward accumulated by } X \text{ in } [0,t]] \\ &\quad = \frac{1}{t}\int_0^t \sum_{i\in\Omega} \Big(r_i + \sum_{\substack{j\in\Omega\\ j\neq i}} \lambda_{i,j} r_{i,j}\Big) p_i(\tau)\,\mathrm{d}\tau = \frac{1}{t}\int_0^t E[r''_{X(\tau)}]\,\mathrm{d}\tau = E\Big[\frac{\int_0^t r''_{X(\tau)}\,\mathrm{d}\tau}{t}\Big] \,, \end{aligned}$$

where $r''_i = r_i + \sum_{\substack{j\in\Omega\\ j\neq i}} \lambda_{i,j} r_{i,j}$, which is $\mathrm{EARR}(t)$ for the rewarded CTMC with added contributions to the reward rates of the states.

Finally, we justify the extension to impulse rewards of the ECRTE measure. Setting conventionally $r_a = 0$, we have

$$\begin{aligned} \mathrm{ECRTE} &= E\,[\text{reward accumulated by } X \text{ in } [0,\infty]] \\ &= \int_0^\infty \lim_{\Delta t\to 0^+} \frac{E\,[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]]}{\Delta t}\,\mathrm{d}t \,. \end{aligned}$$

But, noting that $\Omega = B \cup \{a\}$, (E.4) with $r_a = 0$ yields

$$\lim_{\Delta t\to 0^+} \frac{E[\text{reward accumulated by } X \text{ in } [t, t+\Delta t]]}{\Delta t} = \sum_{i\in B} \left(r_i + \sum_{\substack{j\in B\\ j\neq i}} r_{i,j}\lambda_{i,j}\right) p_i(t) \,,$$

and, thus, using, by Fubini's theorem, $\tau_i = E[\int_0^\infty \mathbf{1}_{X(t)=i}\,\mathrm{d}t] = \int_0^\infty p_i(t)\,\mathrm{d}t$,

$$\begin{aligned}\mathrm{ECRTE} &= \int_0^\infty \sum_{i\in B}\left(r_i + \sum_{\substack{j\in B\\ j\neq i}} r_{i,j}\lambda_{i,j}\right)p_i(t)\,\mathrm{d}t = \sum_{i\in B}\left(r_i + \sum_{\substack{j\in B\\ j\neq i}} r_{i,j}\lambda_{i,j}\right)\int_0^\infty p_i(t)\,\mathrm{d}t \\ &= \sum_{i\in B}\left(r_i + \sum_{\substack{j\in B\\ j\neq i}} r_{i,j}\lambda_{i,j}\right)\tau_i\,,\end{aligned}$$

which, as seen, is ECRTE for the rewarded CTMC with added contributions to the reward rates of the states.

Appendix F

A Tutorial on Rewarded Finite CTMCs

This appendix provides a brief tutorial on rewarded finite CTMCs.

F.1 Finite CTMCs

An (homogeneous) continuous-time Markov chain (CTMC) with state space Ω is a stochastic process $X = \{X(t); t \geq 0\}$ with continuous parameter t in which each random variable $X(t)$, $t \geq 0$ takes values in Ω, satisfying the Markov property and being time invariant. Letting $s \geq 0$, $t > 0$, the Markov property can be stated as:

$$P[X(t+s) = j \mid X(s) = i, X(u) = x(u), 0 \leq u \leq s] = P[X(t+s) = j \mid X(s) = i]\,,$$

where $x(u)$, $0 \leq u \leq s$ is the realization of X (set of states visited by X in the time interval $[0, s]$), and paraphrased by saying that "the conditional distribution of the future state, given the present state and all past states, depends only on the present state and is independent of the past." Time-invariance makes reference to the fact that

$$P[X(t+s) = j \mid X(s) = i] = P[X(t) = j \mid X(0) = i]\,.$$

We will assume Ω finite.

Let $Q_{i,j}(t) = P[X(t) = j \mid X(0) = i]$. The matrix $\mathbf{Q}(t) = (Q_{i,j}(t))_{i,j\in\Omega}$ is called the transition probability matrix of X. Obviously, by definition, we have $\mathbf{Q}(0) = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. A transition matrix is called standard if $\mathbf{Q}(0^+) = \mathbf{I}$, where $\mathbf{Q}(0^+)$ denotes the limit from the right at $t = 0$. We will assume that the CTMC has a standard transition matrix unless explicitly stated otherwise. Intuitively, this is equivalent to assume that the CTMC does not have vanishing (also called instantaneous) states. In that case (see [39]), the derivative of $\mathbf{Q}(t)$ at $t = 0$ from the right, $\mathbf{Q}'(0^+)$, exists and is called the infinitesimal generator (or transition rate matrix) of the CTMC. Let $\mathbf{A} = (a_{i,j})_{i,j\in\Omega}$ be the infinitesimal generator of X. It has elements $a_{i,j} = \lambda_{i,j}$, $i, j \in \Omega$, $j \neq i$ and $a_{i,i} = -\lambda_i = -\sum_{j\in\Omega, j\neq i} \lambda_{i,j}$, $i \in \Omega$. The element $\lambda_{i,j}$, $i, j \in \Omega$, $j \neq i$ is called the transition rate of X from state i to state j. The element λ_i, $i \in \Omega$ is called the output rate of X from state i. The meaning of $\lambda_{i,j}$, $i, j \in \Omega$, $j \neq i$ is clear from its definition:

$$\lambda_{i,j} = \lim_{h\to 0^+} \frac{Q_{i,j}(h)}{h} = \lim_{h\to 0^+} \frac{P[X(h) = j \mid X(0) = i]}{h} = \lim_{h\to 0^+} \frac{P[X(t+h) = j \mid X(t) = i]}{h}\,,$$

and $\lambda_{i,j}$ is the "rate" at which X makes a transition from state i to state j. The meaning of λ_i, $i \in \Omega$ is also clear from its definition:

$$\lambda_i = \lim_{h \to 0^+} \frac{1 - Q_{i,i}(h)}{h} = \lim_{h \to 0^+} \frac{P[X(h) \neq i \mid X(0) = i]}{h} = \lim_{h \to 0^+} \frac{P[X(t+h) \neq i \mid X(t) = i]}{h},$$

and λ_i is the "rate" at which X leaves state i.

A finite CTMC X can be interpreted in terms of its embedded (homogeneous) discrete-time Markov chain (DTMC). The embedded DTMC, $\Pi = \{\Pi_n; n = 0, 1, 2, \ldots\}$, of X is a DTMC having same initial probability distribution as X and transition probabilities $p_{i,j} = P[\Pi_{n+1} = j \mid \Pi_n = i]$, $i, j \in \Omega$ defined as

$$p_{i,j} = \begin{cases} \lambda_{i,j}/\lambda_i & \text{for } j \neq i,\ \lambda_i > 0 \\ 0 & \text{for } j = i,\ \lambda_i > 0 \\ 0 & \text{for } j \neq i,\ \lambda_i = 0 \\ 1 & \text{for } j = i,\ \lambda_i = 0 \end{cases}.$$

The interpretation of X in terms of Π is as follows: Π gives the sequence of states visited by X, and each visit to state i has a duration, called the sojourn time, which is exponentially distributed with parameter λ_i (for $\lambda_i = 0$, the sojourn time would be infinite with probability 1).

Another interpretation of a finite CTMC X in terms of a DTMC which is often useful and which is the basis of many of the numerical methods supported by METFAC-2.1 is based on the randomization (also called uniformization) construct. In that construct (see, for instance, [39]), the CTMC X is interpreted in terms of a Poisson process $N = \{N(t); t \geq 0\}$ with rate $\Lambda \geq \max_{i \in \Omega} \lambda_i$ and an independent randomized DTMC $\widehat{X} = \{\widehat{X}_n; n = 0, 1, 2, \ldots\}$ with randomization rate Λ. The Poisson process is a birth CTMC with state space $\{0, 1, 2, \ldots\}$ with birth rate Λ at every state. Its transient state probabilities are given by

$$P[N(t) = n] = \frac{(\Lambda t)^n}{n!} \mathrm{e}^{-\Lambda t}.$$

The randomized DTMC $\widehat{X}$ is the DTMC with same state space and initial probability distribution as X and transition matrix $\mathbf{P} = (P_{i,j})_{i,j \in \Omega} = \mathbf{I} + \mathbf{A}/\Lambda$, i.e., $P_{i,i} = 1 - \lambda_i/\Lambda$, $i \in \Omega$ and $P_{i,j} = \lambda_{i,j}/\Lambda$, $i, j \in \Omega$, $j \neq i$. The interpretation of X in terms of N and $\widehat{X}$ yields the important result that $X = \{X(t); t \geq 0\}$ and $\{\widehat{X}_{N(t)}; t \geq 0\}$ are probabilistically identical [39, Theorem 4.19], a fact which is used in many of the numerical methods supported by METFAC-2.1.

The interpretation of a CTMC without vanishing states in terms of its embedded DTMC can be extended as follows to a CTMC with vanishing states of the kind METFAC-2.1 allows to specify. In those CTMCs, there is at least one non vanishing state (the one specified by means of the `start_state` construct), for each vanishing state i there is one and only one state j such that $\lambda_{i,j} = \infty$, and there are not cycles of vanishing states, i.e, sequences of states $\{i(k); 1 \leq k \leq n+1\}$, $n \geq 1$, with $i(1) = i(n)$ and $i(k) \neq i(k+1)$, $1 \leq k \leq n-1$, such that $\lambda_{i(k),i(k+1)} = \infty$ for $k = 1, \ldots, n$. In the extended interpretation, the transition probabilities of the embedded DTMC

Π of X are defined as

$$p_{i,j} = \begin{cases} \lambda_{i,j}/\lambda_i & \text{for } j \neq i,\, \lambda_i > 0,\, \lambda_i < \infty \\ 1 & \text{for } j \neq i,\, \lambda_i = \lambda_{i,j} = \infty \\ 0 & \text{for } j \neq i,\, \lambda_i = \infty,\, \lambda_{i,j} < \infty \\ 0 & \text{for } j = i,\, \lambda_i > 0 \\ 0 & \text{for } j \neq i,\, \lambda_i = 0 \\ 1 & \text{for } j = i,\, \lambda_i = 0 \end{cases}.$$

Then, the interpretation of X in terms of Π is as follows: Π gives the sequence of states visited by X, each visit to a non vanishing state i has a sojourn time that is exponentially distributed with parameter λ_i (for $\lambda_i = 0$, the sojourn time would be infinite with probability 1), and each visit to a vanishing state i has a sojourn time τ_i with distribution function $P[\tau_i \leq t] = 1, t \geq 0$.

We continue our review of finite CTMCs that have a standard transition matrix with the classification of the states of the CTMC. That subject is covered in [37]. It turns out that the classification of the states of a finite CTMC X without vanishing states can be performed algorithmically by analyzing the transition diagram of X. The transition diagram of X is a digraph having as set of nodes the states of the CTMC and an arc from state i to state j labeled with the value $\lambda_{i,j}$ if and only if $\lambda_{i,j} > 0$. A state j (not necessarily different from i) is said to be *accessible* from a state i (denoted by $i \to j$) if either $j = i$ or there is path in the transition diagram from i to j. Two states $i, j \in \Omega$ are said to *communicate* (denoted by $i \leftrightarrow j$) if both $i \to j$ and $j \to i$. Thus $i \leftrightarrow i$, $i \in \Omega$ and for $i, j \in \Omega$, $i \neq j$, $i \leftrightarrow j$ if and only if there are paths in the transition diagram both from i to j and from j to i. State communication is an equivalence relation. The resulting classes are called the *classes* of states of X, i.e., a class of states of X is a maximal subset of communicating states. In terms of the transition diagram of X, the classes of X are the strongly connected components of its transition diagram (see, for instance, [40]). Determination of which classes of states of a finite CTMC X are transient and which are recurrent can be done by analyzing the *digraph of state classes* of X. The *digraph of state classes* of a finite CTMC X is the acyclic digraph having a node for each class of states of X and an arc from state class C to state class C' if and only if the transition diagram of X has an arc from some state in C to some state in C'. To illustrate the concepts defined so far, Figure F.1 gives the transition diagram of a small finite CTMC and the corresponding digraph of state classes. Then, the recurrent classes of states are the classes which in the digraph of state classes do not have outgoing arcs, implying that once the CTMC enters such a class it will not leave it, and the transient classes are the classes which in the digraph of state classes have some outgoing arc, implying that the CTMC will leave the class with probability 1 and that, once the CTMC leaves the class it will never reenter it. For the example given in Figure F.1, state classes C_1, C_2 and C_3 would be transient and state classes C_4 and C_5 would be recurrent. A finite CTMC having a single (recurrent) class of states is said to be *irreducible*. A state $i \in \Omega$ is said to be absorbing if and only if $\lambda_i = 0$. This is equivalent to the state i making up a recurrent class of states. For the example given in Figure F.1, state 9 would be absorbing.

We end this brief review of finite CTMCs by discussing the concept of reachability in finite CTMCs without vanishing states. Let $p_i(t) = P[X(t) = i]$, $i \in \Omega$ be the transient probability of state i. Then, a state i is said to be reachable if and only if $p_i(t) > 0$ for some $t > 0$, which

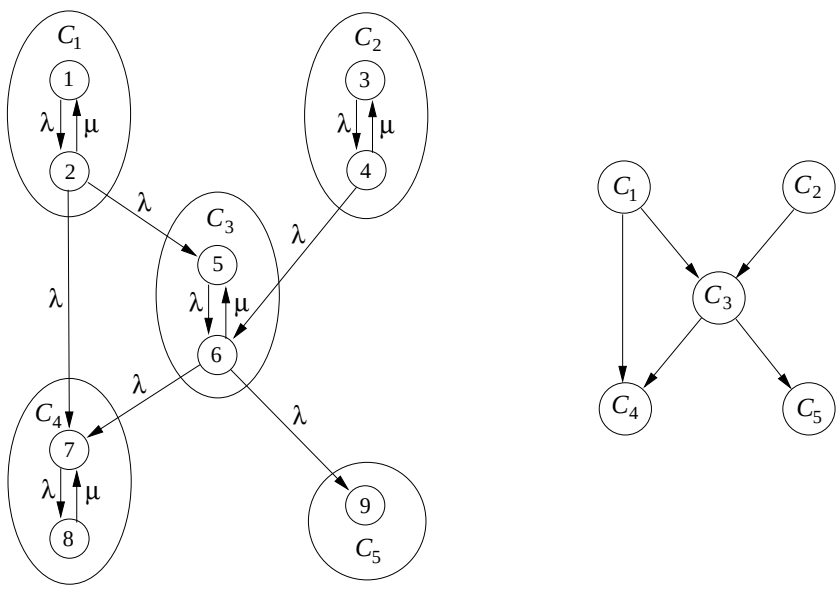

Figure F.1: Transition diagram of a small finite CTMC (left) and the corresponding digraph of state classes (right).

is equivalent to $p_i(t) > 0$ for all $t > 0$. It turns out that reachability is a class property, i.e., all states of a given class of states C are either reachable or unreachable. Thus, we can properly talk about reachable and unreachable state classes. Unreachable states are never visited and can be suppressed from the state space of the CTMC without modifying the feasible realizations of the CTMC. Determination of which classes of states are reachable and which are unreachable can be made by analyzing the digraph of state classes together with the initial probability distribution vector of the CTMC, $\boldsymbol{\alpha} = (\alpha_i)_{i\in\Omega}$, $\alpha_i = P[X(0) = i]$. For every class C of states of the CTMC, define $\alpha_C = \sum_{i\in C} \alpha_i$, i.e., α_C is the probability that initially the CTMC is in some state of class C. Then, a class C is reachable if either $\alpha_C > 0$, or there exists some other class of states C' with $\alpha_{C'} > 0$ such that there is some path in the digraph of state classes from C' to C. For the example in Figure F.1, assuming $\alpha_{C_1} > 0$, $\alpha_{C_2} = 0$, $\alpha_{C_3} > 0$, $\alpha_{C_4} > 0$, and $\alpha_{C_5} = 0$, the reachable state classes would be the classes C_1, C_3, C_4, and C_5, and the state class C_2 would be unreachable. METFAC-2.1 aborts the execution of a model if the corresponding CTMC with vanishing states deleted includes one or more unreachable states.

F.2 Rewarded Finite CTMCs

A rewarded finite CTMC is a finite CTMC with a reward structure imposed over it. In general, the reward structure may include reward rates r_i associated with states and impulse rewards $r_{i,j}$ associated with transitions. The quantity r_i has the meaning of "rate at which reward is earned while X is in state i." The quantity $r_{i,j}$ has the meaning of "reward which is earned each time X makes a transition from state i to state j." METFAC-2.1 allows the specification and solution of rewarded finite CTMCs with a reward structure including only reward rates. However, for the purpose of computing many reward measures, rewarded finite CTMCs with both reward rates and impulse rewards can be mapped into rewarded finite CTMCs with only reward rates (see Section 4

and Appendix E).

Appendix G

Some Sizable Examples

In this section we will illustrate the capabilities of METFAC-2.1 using three sizable modeling examples of increasing complexity.

G.1 A Reliability Model of a 5-level RAID Storage Subsystem

RAID (Redundant Array of Inexpensive Disks) architectures may provide reliable and high capacity storage at a moderate cost. Level 5 is one of the most popular RAID architectures. In a level-5 RAID architecture with M disks, data is organized into groups of $M-1$ bits and a parity bit is computed for each group. Data blocks are organized into M subblocks of which one subblock contains the parity bits of the corresponding $M-1$ bit groups residing in the other $M-1$ subblocks. Each subblock is stored in a different disk. The disk in which the parity subblock is stored is rotated among the disks comprising the RAID. This balances the load of the disks. The architecture tolerates the failure of one disk without losing data. When a disk fails, the failed disk is first repaired. After that, a reconstruction process writes into the repaired disk the data which that disk has to hold to be consistent with the data of the remaining disks. In this section we will analyze the reliability of a 5-level RAID storage subsystem comprising eight disks, two redundant disk controllers and two redundant power supplies (see Figure G.1). The power supplies work in cold standby redundancy. The RAID subsystem is up if, ignoring coverage faults, at least one controller is unfailed, at least one power supply is unfailed and at least seven disks have updated data. Disks fail with rate λ_{D} if no disk is under reconstruction and with rate λ_{DR} if one disk is under reconstruction; controllers fail with rate λ_{C2} if the subsystem has two unfailed controllers and with rate λ_{C1} if the subsystem has one unfailed controller; the active power supply fails with rate λ_{P}; the coverage to controller failures is C_{C}; and the coverage to power supply failures is C_{P}. Disks are reconstructed with rate μ_{DR}. It is assumed the availability of an unlimited number of repairmen to repair failed components. The repair rate is μ_{R} for all components. The measure of interest is the unreliability at time t (probability that the system has failed in the time interval $[0,t]$), $\mathrm{ur}(t)$. It is assumed that initially the system is in the state without failed components and all disks with consistent data.

We will call `raid` the unreliability model of the 5-level RAID subsystem. The model will be specified using an enumeration style in which the states and transition rates of the rewarded CTMC model are specified directly. The specification will make use of 13 state variables identifying the

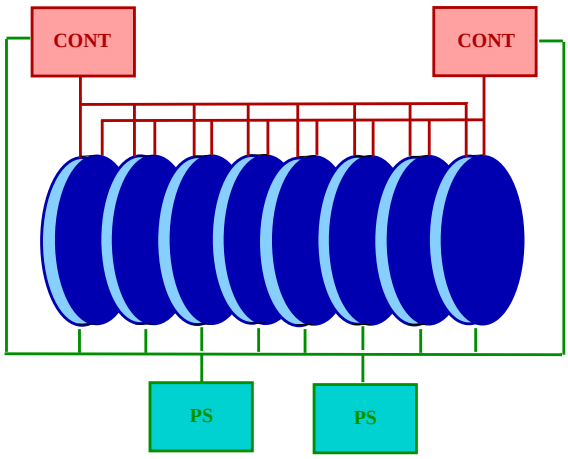

Figure G.1: Architecture of the RAID subsystem.

state of the RAID subsystem: `FOP` (fully operational, i.e., without failed components and with all disks with consistent data), `C` (with one controller failed), `D` (with one disk failed), `P` (with one power supply failed), `R` (with one disk under reconstruction), `CD` (with one controller and one disk failed), `CP` (with one controller and one power supply failed), `CR` (with one controller failed and one disk under reconstruction), `DP` (with one disk and one power supply failed), `PR` (with one power supply failed and one disk under reconstruction), `CDP` (with one controller, one disk, and one power supply failed), `CPR` (with one controller and one power supply failed and one disk under reconstruction), and `DOWN` (down). We give next a model specification file `raid.spec` that is appropriate for computing the unreliability $\mathrm{ur}(t)$ using the measure $\mathrm{CRDTE}(s)$ with $s = t$. Since the initial probability distribution is concentrated in the "start state" of the model specification, it is not necessary to use the construct `initial_probability`, and since no model-specific functions will be used, the model specification will not include a file `raid.c`.

raid.spec

```
parameters

   double
   LD,  /* disk failure rate when the RAID has no disk under reconstruction */
   LDR, /* disk failure rate when the RAID has one disk under reconstruction */
   LC2, /* controller failure rate when both controllers are failed */
   LC1, /* controller failure rate when only one controller is failed */
   LP,  /* power supply failure rate */
   CVC, /* coverage to controller failures */
   CVP, /* coverage to power supply failures */
   MDR, /* disk reconstruction rate */
   MR   /* repair rate */

state_variables

   FOP, C, D, P, R, CD, CP, CR, DP, PR, CDP, CPR, DOWN

start_state
```

```
   FOP=yes, C=no, D=no, P=no, R=no, CD=no, CP=no, CR=no, DP=no,
   PR=no, CDP=no, CPR=no, DOWN=no

reward_rate

  (double)(DOWN==no)

production_rules

if FOP action FOP_C with_rate 2*LC2*CVC
next_state  FOP=no, C=yes

if FOP action FOP_D with_rate 8*LD
next_state  FOP=no, D=yes

if FOP action FOP_P with_rate LP*CVP
next_state  FOP=no, P=yes

if FOP action FOP_DOWN with_rate 2*LC2*(1-CVC)+LP*(1-CVP)
next_state  FOP=no, DOWN=yes

if C action C_FOP with_rate MR
next_state  C=no, FOP=yes

if C action C_CD with_rate 8*LD
next_state  C=no, CD=yes

if C action C_CP with_rate LP*CVP
next_state  C=no, CP=yes

if C action C_DOWN with_rate LC1+LP*(1-CVP)
next_state  C=no, DOWN=yes

if D action D_R with_rate MR
next_state  D=no, R=yes

if D action D_CD with_rate 2*LC2*CVC
next_state  D=no, CD=yes

if D action D_DP with_rate LP*CVP
next_state  D=no, DP=yes

if D action D_DOWN with_rate 7*LD+2*LC2*(1-CVC)+LP*(1-CVP)
next_state  D=no, DOWN=yes

if P action P_FOP with_rate MR
next_state  P=no, FOP=yes

if P action P_CP with_rate 2*LC2*CVC
next_state  P=no, CP=yes

if P action P_DP with_rate 8*LD
next_state  P=no, DP=yes
```

```
if P action P_DOWN with_rate 2*LC2*(1-CVC)+LP
next_state  P=no, DOWN=yes

if R action R_FOP with_rate MDR
next_state  R=no, FOP=yes

if R action R_D with_rate LDR
next_state  R=no, D=yes

if R action R_CR with_rate 2*LC2*CVC
next_state  R=no, CR=yes

if R action R_PR with_rate LP*CVP
next_state  R=no, PR=yes

if R action R_DOWN with_rate 7*LDR+2*LC2*(1-CVC)+LP*(1-CVP)
next_state  R=no, DOWN=yes

if CD action CD_D with_rate MR
next_state  CD=no, D=yes

if CD action CD_CR with_rate MR
next_state  CD=no, CR=yes

if CD action CD_CDP with_rate LP*CVP
next_state  CD=no, CDP=yes

if CD action CD_DOWN with_rate 7*LD+LC1+LP*(1-CVP)
next_state  CD=no, DOWN=yes

if CP action CP_C with_rate MR
next_state  CP=no, C=yes

if CP action CP_P with_rate MR
next_state  CP=no, P=yes

if CP action CP_CDP with_rate 8*LD
next_state  CP=no, CDP=yes

if CP action CP_DOWN with_rate LC1+LP
next_state  CP=no, DOWN=yes

if CR action CR_C with_rate MDR
next_state  CR=no, C=yes

if CR action CR_R with_rate MR
next_state  CR=no, R=yes

if CR action CR_CD with_rate LDR
next_state  CR=no, CD=yes

if CR action CR_CPR with_rate LP*CVP
next_state  CR=no, CPR=yes
```

```
if CR action CR_DOWN with_rate 7*LDR+LC1+LP*(1-CVP)
next_state  CR=no, DOWN=yes

if DP action DP_D with_rate MR
next_state  DP=no, D=yes

if DP action DP_PR with_rate MR
next_state  DP=no, PR=yes

if DP action DP_CDP with_rate 2*LC2*CVC
next_state  DP=no, CDP=yes

if DP action DP_DOWN with_rate 7*LD+2*LC2*(1-CVC)+LP
next_state  DP=no, DOWN=yes

if PR action PR_P with_rate MDR
next_state  PR=no, P=yes

if PR action PR_R with_rate MR
next_state  PR=no, R=yes

if PR action PR_DP with_rate LDR
next_state  PR=no, DP=yes

if PR action PR_CPR with_rate 2*LC2*CVC
next_state  PR=no, CPR=yes

if PR action PR_DOWN with_rate 7*LDR+2*LC2*(1-CVC)+LP
next_state  PR=no, DOWN=yes

if CDP action CDP_CD with_rate MR
next_state  CDP=no, CD=yes

if CDP action CDP_DP with_rate MR
next_state  CDP=no, DP=yes

if CDP action CDP_CPR with_rate MR
next_state  CDP=no, CPR=yes

if CDP action CDP_DOWN with_rate 7*LD+LC1+LP
next_state  CDP=no, DOWN=yes

if CPR action CPR_CP with_rate MDR
next_state  CPR=no, CP=yes

if CPR action CPR_CR with_rate MR
next_state  CPR=no, CR=yes

if CPR action CPR_PR with_rate MR
next_state  CPR=no, PR=yes

if CPR action CPR_CDP with_rate LDR
next_state  CPR=no, CDP=yes
```

```
if CPR action CPR_DOWN with_rate 7*LDR+LC1+LP
next_state  CPR=no, DOWN=yes
```

Assume that it is desired to analyze how the unreliability $\mathrm{ur}(t)$ at $t = 1$ year depends on the coverages to controller and power supply failures for $\lambda_{\mathrm{D}} = 4 \times 10^{-6}\ \mathrm{h}^{-1}$, $\lambda_{\mathrm{DR}} = 6 \times 10^{-6}\ \mathrm{h}^{-1}$, $\lambda_{\mathrm{C2}} = 2 \times 10^{-5}\ \mathrm{h}^{-1}$, $\lambda_{\mathrm{C1}} = 3 \times 10^{-5}\ \mathrm{h}^{-1}$, $\lambda_{\mathrm{P}} = 2 \times 10^{-5}\ \mathrm{h}^{-1}$, $\mu_{\mathrm{DR}} = 0.125\ \mathrm{h}^{-1}$, and $\mu_{\mathrm{R}} = 0.05\ \mathrm{h}^{-1}$. We may choose to give to C_{C} the values 0.9, 0.95 and 0.99 and to give to C_{P} the values 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, and 0.99. This will require running the executable file `raid.exe` 30 times and collecting the obtained results. Doing that manually is cumbersome and error prone. A better idea is to generate automatically the input required by the tool for each run and launch the runs using two C-shell (`csh`) scripts. Assuming that the measure $\mathrm{CRDTE}(s)$ (equal, we recall, to $\mathrm{ur}(t)$ for $t = s$) is computed using the method "Standard Randomization" with an error requirement $\varepsilon = 10^{-8}$ and that the full path to the C-shell is `/bin/csh`, two appropriate C-shell scripts would be the scripts "`input.csh`" and "`schedule.csh`" given next. The script "`input.csh`" builds in the file "`raid.inp`" the input required by `raid.exe` and has as first parameter the value of C_{C} and as a second parameter the value of C_{P}; the script "`schedule.csh`" schedules the executions of `raid.exe`, collecting the results in the file `raid.res`.

input.csh

```
#!/bin/csh

set CVC = ($1)
set CVP = ($2)

set LD = (4e-6)
set LDR = (6e-6)
set LC2 = (2e-5)
set LC1 = (3e-5)
set LP = (2e-5)
set MDR = (0.125)
set MR = (0.05)

echo $LD >! raid.inp
foreach val ($LDR $LC2 $LC1 $LP $CVC $CVP $MDR $MR)
   echo $val >> raid.inp
end
echo 1 >> raid.inp
echo 7 >> raid.inp
echo 1 >> raid.inp
echo 1e-8 >> raid.inp
echo 100000 >> raid.inp
echo 1 >> raid.inp
echo 8760 >> raid.inp
echo y >> raid.inp
#
```

schedule.csh

```
#!/bin/csh
```

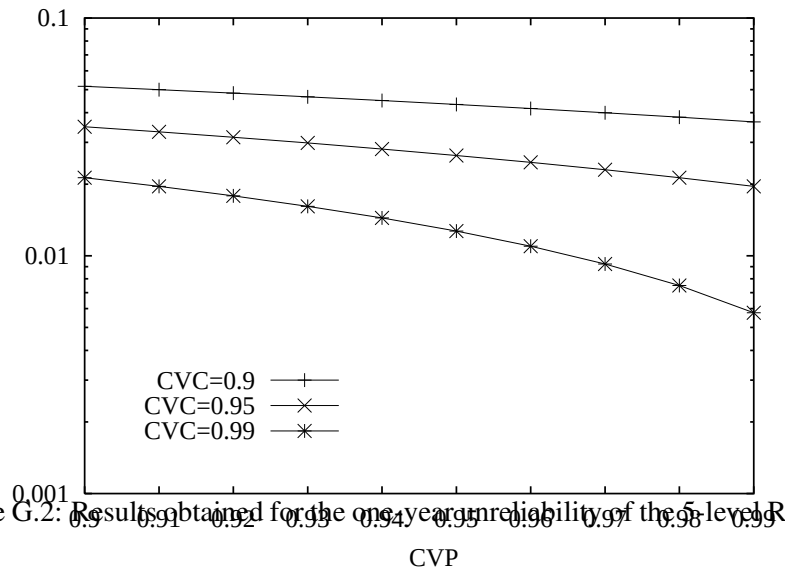

Figure G.2: Results obtained for the one-year unreliability of the 5-level RAID storage subsystem.

```
echo "" >! raid.res
foreach CVC (0.9 0.95 0.99)
   foreach CVP (0.9 0.91 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99)
      input $CVC $CVP
      raid.exe < raid.inp > /dev/null
      echo ">" >> raid.res
      echo ">" >> raid.res
      cat raid.res raid.log >! tmp.out
      \mv -f tmp.out raid.res
   end
end
\rm -f raid.inp
#
```

Figure G.2 plots the obtained results. We can note that the relative improvement on $\mathrm{ur}(t)$ due to increasing C_{P} (CVP in the figure) gets more noticeable as C_{C} (CVC in the figure) gets closer to 1.

G.2 A Reliability Model of a Storage System

The second example is a storage system comprising N 5-level RAID storage subsystems such as those described in Section G.1. The storage system is up if every RAID subsystem is up. The measure of interest is the unreliability of the system at time t, $\mathrm{ur}(t)$. An appropriate model specification can be obtained by defining state variables counting the number of RAID subsystems in each operational state and one state variable indicating that the system has failed. Such a model specification takes advantage of the fact that all RAID subsystems have identical behavior to reduce the size of the generated CTMC. However, the resulting CTMCs can be large for large values of N. The size of the CTMCs can be kept very small, however, using bounding methods.

A CTMC, X_{b}, yielding bounds for $\mathrm{ur}(t)$ would have a state space $S \cup \{f, a\}$, where f and a are absorbing states and S includes operational states with no more than M failed components or disks under reconstruction. Transitions in the exact model from S to the "system down" state would be directed to state f; transitions in the exact model from states in S to operational states with more than M failed components or disks under reconstruction would be directed to state a. Then, a lower bound for $\mathrm{ur}(t)$ would be $\mathrm{ur}_{\mathrm{lb}}(t) = P[X_{\mathrm{b}}(t) = f]$ and an upper bound for $\mathrm{ur}(t)$ would be $\mathrm{ur}_{\mathrm{ub}}(t) = P[X_{\mathrm{b}}(t) = f] + P[X_{\mathrm{b}}(t) = a]$. Small values of M are enough to obtain tight bounds for $\mathrm{ur}(t)$. A suitable model specification for computing both bounds using the $\mathrm{ETRR}(t)$ measure follows. The specification is easily derived from the model specification of the RAID subsystem given in Section G.1 and includes a model specification file `raidsys.spec` and a C file `raidsys.c` defining a function counting the number of failed components and disks under reconstruction in an operational state. State variables `F` and `A` identify the absorbing states f and a, respectively. An `int` parameter `UB` is used to indicate whether the lower bound (`UB` $= 0$) or the upper bound (`UB` $\neq 0$) is to be computed. Note that when X_{b} has to enter either the absorbing state f or the absorbing state a, the counting variables are reset. If they were not, multiple absorbing states f and a would be generated and the resulting CTMC could be substantially larger.

raidsys.spec

```
parameters

int
   N,   /* number of RAID susbsystems */
   M,   /* operational states with up to M failed components or
           disks under reconstruction are generated */
   UB,  /* when not 0 compute upper bound; when 0 compute
           lower bound */
double
   LD,  /* disk failure rate when the RAID has no disk under
           reconstruction */
   LDR, /* disk failure rate when the RAID has one disk under
           reconstruction */
   LC2, /* controller failure rate when both controllers are failed */
   LC1, /* controller failure rate when only one controller is failed */
   LP,  /* power supply failure rate */
   CVC, /* coverage to controller failures */
   CVP, /* coverage to power supply failures */
   MDR, /* disk reconstruction rate */
   MR   /* repair rate */

state_variables

   NFOP, NC, ND, NP, NR, NCD, NCP, NCR, NDP, NPR, NCDP, NCPR, F, A

external

int NFCDR(int, int, int, int, int, int, int, int, int, int, int)


start_state
```

```
NFOP=N, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
NPR=0, NCDP=0, NCPR=0, F=no, A=no

reward_rate

(double)(F==1 || (A==1 && UB))

production_rules

if NFOP>0 action FOP_C with_rate NFOP*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NFOP--, NC++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end

if NFOP>0 action FOP_ND with_rate NFOP*8*LD

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NFOP--, ND++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end


if NFOP>0 action FOP_P with_rate NFOP*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NFOP--, NP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end

if NFOP>0 action FOP_DOWN with_rate NFOP*(2*LC2*(1-CVC)+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NC>0 action C_FOP with_rate NC*MR
next_state  NC--, NFOP++

if NC>0 action C_CD with_rate NC*8*LD

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NC--, NCD++
```

```
   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NC>0 action C_CP with_rate NC*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NC--, NCP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NC>0 action C_DOWN with_rate NC*(LC1+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if ND>0 action D_R with_rate ND*MR
next_state  ND--, NR++

if ND>0 action D_CD with_rate ND*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  ND--, NCD++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if ND>0 action D_DP with_rate ND*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  ND--, NDP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if ND>0 action D_DOWN with_rate ND*(7*LD+2*LC2*(1-CVC)+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NP>0 action P_FOP with_rate NP*MR
next_state  NP--, NFOP++
```

```
if NP>0 action P_CP with_rate NP*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NP--, NCP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end

if NP>0 action P_DP with_rate NP*8*LD

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NP--, NDP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end

if NP>0 action P_DOWN with_rate NP*(2*LC2*(1-CVC)+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NR>0 action R_FOP with_rate NR*MDR
next_state  NR--, NFOP++

if NR>0 action R_D with_rate NR*LDR
next_state  NR--, ND++

if NR>0 action R_CR with_rate NR*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NR--, NCR++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end

if NR>0 action R_PR with_rate NR*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NR--, NPR++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
               NPR=0, NCDP=0, NCPR=0, A=yes

end
```

```
if NR>0 action R_DOWN with_rate NR*(7*LDR+2*LC2*(1-CVC)+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NCD>0 action CD_D with_rate NCD*MR
next_state  NCD--, ND++

if NCD>0 action CD_CR with_rate NCD*MR
next_state  NCD--, NCR++

if NCD>0 action CD_CDP with_rate NCD*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NCD--, NCDP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NCD>0 action CD_DOWN with_rate NCD*(7*LD+LC1+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NCP>0 action CP_C with_rate NCP*MR
next_state  NCP--, NC++

if NCP>0 action CP_P with_rate NCP*MR
next_state  NCP--, NP++

if NCP>0 action CP_CDP with_rate NCP*8*LD

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NCP--, NCDP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NCP>0 action CP_DOWN with_rate NCP*(LC1+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NCR>0 action CR_C with_rate NCR*MDR
next_state  NCR--, NC++

if NCR>0 action CR_R with_rate NCR*MR
next_state  NCR--, NR++

if NCR>0 action CR_CD with_rate NCR*LDR
next_state  NCR--, NCD++
```

```
if NCR>0 action CR_CPR with_rate NCR*LP*CVP

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NCR--, NCPR++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NCR>0 action CR_DOWN with_rate NCR*(7*LDR+LC1+LP*(1-CVP))
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NDP>0 action DP_D with_rate NDP*MR
next_state  NDP--, ND++

if NDP>0 action DP_PR with_rate NDP*MR
next_state  NDP--, NPR++

if NDP>0 action DP_CDP with_rate NDP*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NDP--, NCDP++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes

end

if NDP>0 action DP_DOWN with_rate NDP*(7*LD+2*LC2*(1-CVC)+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NPR>0 action PR_P with_rate NPR*MDR
next_state  NPR--, NP++

if NPR>0 action PR_R with_rate NPR*MR
next_state  NPR--, NR++

if NPR>0 action PR_DP with_rate NPR*LDR
next_state  NPR--, NDP++

if NPR>0 action PR_CPR with_rate NPR*2*LC2*CVC

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)<M response GO
   next_state  NPR--, NCPR++

   if NFCDR(NC,ND,NP,NR,NCD,NCP,NCR,NDP,NPR,NCDP,NCPR)==M response A
   next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
                NPR=0, NCDP=0, NCPR=0, A=yes
```

```
end

if NPR>0 action PR_DOWN with_rate NPR*(7*LDR+2*LC2*(1-CVC)+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NCDP>0 action CDP_CD with_rate NCDP*MR
next_state  NCDP--, NCD++

if NCDP>0 action CDP_DP with_rate NCDP*MR
next_state  NCDP--, NDP++

if NCDP>0 action CDP_CPR with_rate NCDP*MR
next_state  NCDP--, NCPR++

if NCDP>0 action CDP_DOWN with_rate NCDP*(7*LD+LC1+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes

if NCPR>0 action CPR_CP with_rate NCPR*MDR
next_state  NCPR--, NCP++

if NCPR>0 action CPR_CR with_rate NCPR*MR
next_state  NCPR--, NCR++

if NCPR>0 action CPR_PR with_rate NCPR*MR
next_state  NCPR--, NPR++

if NCPR>0 action CPR_CDP with_rate NCPR*LDR
next_state  NCPR--, NCDP++

if NCPR>0 action CPR_DOWN with_rate NCPR*(7*LDR+LC1+LP)
next_state  NFOP=0, NC=0, ND=0, NP=0, NR=0, NCD=0, NCP=0, NCR=0, NDP=0,
            NPR=0, NCDP=0, NCPR=0, F=yes
```

raidsys.c

```
#include "raidsys.h"

int NFCDR(int NC, int ND, int NP, int NR, int NCD, int NCP,
          int NCR, int NDP, int NPR, int NCDP, int NCPR)
{
  return 3*(NCDP+NCPR)+2*(NCD+NCP+NCR+NDP+NPR)+NC+ND+NP+NR;
}
```

The size of X_{b} is independent on N for $N \geq M$. As previously commented, small values of M are enough to obtain tight bounds. To illustrate, Table G.1 gives the number of states and transitions of X_{b} and the bounds for the one-year unreliability for increasing values of M and $N = 10$, $\lambda_{\mathrm{D}} = 4 \times 10^{-6}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{DR}} = 6 \times 10^{-6}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{C2}} = 2 \times 10^{-5}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{C1}} = 3 \times 10^{-5}\,\mathrm{h}^{-1}$, $\lambda_{\mathrm{P}} = 2 \times 10^{-5}\,\mathrm{h}^{-1}$, $C_{\mathrm{C}} = 0.98$, $C_{\mathrm{P}} = 0.99$, $\mu_{\mathrm{DR}} = 0.125\,\mathrm{h}^{-1}$, and $\mu_{\mathrm{R}} = 0.05\,\mathrm{h}^{-1}$. The bounds have been computed using the method "Standard Randomization" with an error requirement $\varepsilon = 10^{-10}$

Table G.1: Size of X_{b} and bounds obtained for the one-year unreliability for increasing values of M.

M	states	transitions	lower bound	upper bound
1	7	17	0.0821056001	0.2179152859
2	22	95	0.0885426759	0.0899597496
3	64	376	0.0886092875	0.0886180552
4	172	1,234	0.0886097050	0.0886097439
5	418	3,502	0.0886097068	0.0886097070

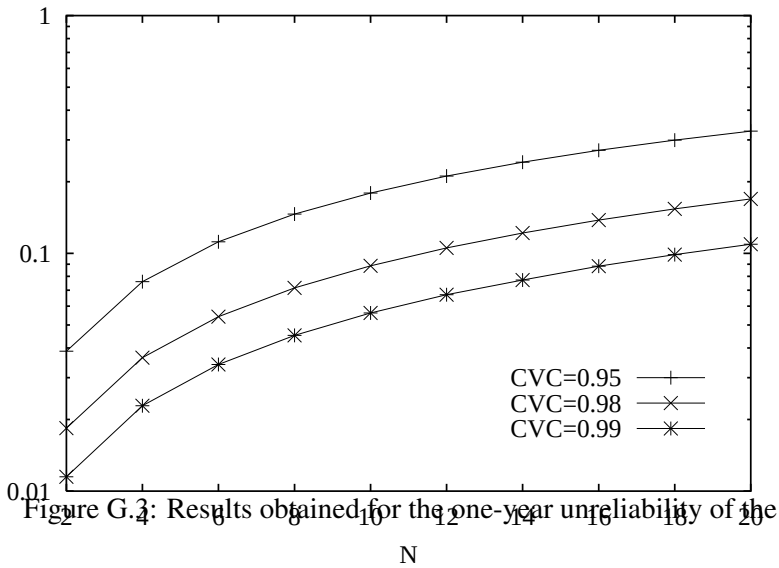

Figure G.3: Results obtained for the one-year unreliability of the storage system.

Assume that we want to investigate the impact of N and C_{C} on the one-year unreliability of the storage system with the remaining parameters with the previously given values. We can take for C_{C} the values 0.95, 0.98 and 0.99 and for N the values 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20. According to the results shown on Table G.1, $M = 5$ should yield very tight bounds. Figure G.3 plots the obtained results. (In the figure, C_{C} is denoted `CVC`.)

G.3 A Performability Model of a Multiprocessor System

The third and last example is a performability model of a multiprocessor system. The system includes 16 processors interconnected by an 8-node hypercube, as shown in Figure G.4. Processors fail with rate λ_{P}; nodes of the hypercube fail with rate λ_{N}; links of the hypercube fail with rate λ_{L}. A fault of a processor is covered with probability C_{P}; a fault of a node of the hypercube is covered with probability C_{N}. Coverage to link faults is assumed perfect. There is an unlimited number of repairmen to repair components in covered failure. The repair rate is μ_{P} for processors,

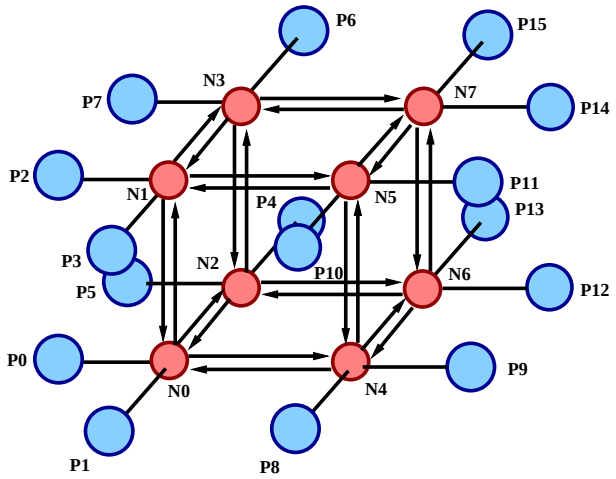

Figure G.4: Architecture of the fault-tolerant multiprocessor system.

μ_{N} for nodes, and μ_{L} for links. A completely down system because there was an uncovered fault is brought to a fully operational state without failed components at rate μ_{G}. It is assumed the availability of diagnosis and reconfiguration procedures to determine a subset of interconnected unfailed processors of maximal size and to reconfigure the multiprocessor so that it works using such a maximal subset. The measure of interest is $\mathrm{CRCD}(t, s)$ using as reward rates the speedup function of the number of connected processors in the healthy subset in which the system is configured described in Table G.2. This gives the probability that the normalized performance of the multiprocessor in the time interval $[0, t]$ is $> s$. This allows to take into account that the performance of the multiprocessor degrades as components fail. The system is initially in the state without failed components.

An exact rewarded CTMC of the multiprocessor has an unmanageable state space. Instead, we will use bounding models with state space $S \cup \{f\}$, where f is an absorbing state in which the bounding model enters when the exact model would exit subset S and S includes the states with up to M covered faults and the state in which the system is down due to an uncovered fault. By assigning to the absorbing state a reward rate $r_f = 0$ we obtain a lower bounding model; by assigning to the absorbing state a reward rate $r_f = 12$ we obtain an upper bounding model. A model specification including a model specification file `mp.spec` and a C file `mp.c` is given next (for the file `mp.spec` we illustrate its contents). The specification includes an `int` parameter `UB` such that when `UB` $= 0$ the lower bound is computed and when `UB` $\neq 0$ the upper bound is computed and another `int` parameter `MAXF` indicating the maximum number of covered faults for which states have to be included in S. An external `double` function is used to compute the reward rates. Programming that function is relatively complicated since it requires to determine the number of processors in a maximum healthy connected subset of processors. That programming is performed in the file `mp.c`. In file `mp.spec`, note the use of the state variable `NF` keeping track of the number of covered faults and that the construct `initial_probability` is not used because the initial probability distribution is concentrated in the "start state" of the model specification. Note also that when the bounding model has to enter either the absorbing state f or the state in which the system is down due to an uncovered fault, the state variables are reset not to have multiple absorbing states and multiple states in which the system is down due to an uncovered

Table G.2: Speedups in h^{-1} of the multiprocessor system as a function of the number of connected operational processors.

processors	speedup
1	1
2	1.96667
3	2.9
4	3.8
5	4.66667
6	5.5
7	6.3
8	7.06667
9	7.8
10	8.5
11	9.16667
12	9.8
13	10.4
14	10.96667
15	11.5
16	12

fault.

mp.spec

```
parameters

   int
   UB,    /* yes for upper bound; no for lower bound */
   MAXF,  /* maximum number of covered faults */

   double
   LDP,   /* processor failure rate */
   LDN,   /* node failure rate */
   LDL,   /* link failure rate */
   CP,    /* processor coverage */
   CN,    /* node coverage */
   MUP,   /* processor repair rate */
   MUN,   /* node repair rate */
   MUL,   /* link rate */
   MUG    /* global repair rate from state with uncovered fault */

state_variables

   P0,    /* processor 0 up */
   P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15,
   N0,    /* node 0 up */
   N1, N2, N3, N4, N5, N6, N7,
   L0_1,  /* link from node 0 to node 1 up */
   L0_2, L0_4, L1_0, L1_3, L1_5, L2_0, L2_3, L2_6, L3_1, L3_2, L3_7, L4_0,
```

```
   L4_5, L4_6, L5_1, L5_4, L5_7, L6_2, L6_4, L6_7, L7_3, L7_5, L7_6,
   NF,    /* number of covered faults */
   UNCOV, /* yes if uncovered fault */
   ABS    /* yes if absorbing state */

external

double
proc_rate(int, int, int, int, int, int, int, int, int, int, int, int,
          int, int, int, int, int, int, int, int, int, int, int, int,
          int, int, int, int, int, int, int, int, int, int, int, int,
          int, int, int, int, int, int, int, int, int, int, int, int,
          int, int, int)

start_state

P0=yes, P1=yes, P2=yes, P3=yes, P4=yes, P5=yes, P6=yes, P7=yes,
P8=yes, P9=yes, P10=yes, P11=yes, P12=yes, P13=yes, P14=yes,
P15=yes,
N0=yes, N1=yes, N2=yes, N3=yes, N4=yes, N5=yes, N6=yes, N7=yes,
L0_1=yes, L0_2=yes, L0_4=yes,
L1_0=yes, L1_3=yes, L1_5=yes,
L2_0=yes, L2_3=yes, L2_6=yes,
L3_1=yes, L3_2=yes, L3_7=yes,
L4_0=yes, L4_5=yes, L4_6=yes,
L5_1=yes, L5_4=yes, L5_7=yes,
L6_2=yes, L6_4=yes, L6_7=yes,
L7_3=yes, L7_5=yes, L7_6=yes,
NF=0, UNCOV=no, ABS=no

reward_rate

proc_rate(UB,
          P0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,
          N0,N1,N2,N3,N4,N5,N6,N7,
          L0_1,L0_2,L0_4,
          L1_0,L1_3,L1_5,
          L2_0,L2_3,L2_6,
          L3_1,L3_2,L3_7,
          L4_0,L4_5,L4_6,
          L5_1,L5_4,L5_7,
          L6_2,L6_4,L6_7,
          L7_3,L7_5,L7_6,
          UNCOV,ABS)

production_rules

/* failure of processor 0 */
if P0 action P0_F with_rate LDP

   if NF<=MAXF-1 response COV with_prob CP
   next_state P0=no, NF++

   if NF==MAXF response COV with_prob CP
```

```
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

   response UNCOV with_prob 1-CP
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=yes, ABS=no

end

...

/* failure of processor 15 */
if P15 action P15_F with_rate LDP

   if NF<=MAXF-1 response COV with_prob CP
   next_state P15=no, NF++

   if NF==MAXF response COV with_prob CP
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

   response UNCOV with_prob 1-CP
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=yes, ABS=no

end

/* failure of node 0 */
if N0 action N0_F with_rate LDN

   if NF<=MAXF-1 response COV with_prob CN
   next_state N0=no, NF++

   if NF==MAXF response COV with_prob CN
```

```
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

   response UNCOV with_prob 1-CN
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=yes, ABS=no

end

...

/* failure of node 7 */
if N7 action N7_F with_rate LDN

   if NF<=MAXF-1 response COV with_prob CN
   next_state N7=no, NF++

   if NF==MAXF response COV with_prob CN
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

   response UNCOV with_prob 1-CN
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=yes, ABS=no

end

/* failure of link from node 0 to node 1 */
if L0_1 action L0_1_F with_rate LDL

   if NF<=MAXF-1 response NOABS
   next_state L0_1=no, NF++

   if NF==MAXF response ABS
```

```
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

end

...

/* failure of link from node 7 to node 6 */
if L7_6 action L7_6_F with_rate LDL

   if NF<=MAXF-1 response NOABS
   next_state L7_6=no, NF++

   if NF==MAXF response ABS
   next_state P0=no, P1=no, P2=no, P3=no, P4=no, P5=no, P6=no, P7=no, P8=no,
              P9=no, P10=no, P11=no, P12=no, P13=no, P14=no, P15=no, N0=no,
              N1=no, N2=no, N3=no, N4=no, N5=no, N6=no, N7=no, L0_1=no,
              L0_2=no, L0_4=no, L1_0=no, L1_3=no, L1_5=no, L2_0=no, L2_3=no,
              L2_6=no, L3_1=no, L3_2=no, L3_7=no, L4_0=no, L4_5=no, L4_6=no,
              L5_1=no, L5_4=no, L5_7=no, L6_2=no, L6_4=no, L6_7=no, L7_3=no,
              L7_5=no, L7_6=no, NF=0, UNCOV=no, ABS=yes

end

/* global repair from state with uncovered failure */
if UNCOV action UN_R with_rate MUG
next_state P0=yes, P1=yes, P2=yes, P3=yes, P4=yes, P5=yes, P6=yes, P7=yes,
           P8=yes, P9=yes, P10=yes, P11=yes, P12=yes, P13=yes, P14=yes,
           P15=yes, N0=yes, N1=yes, N2=yes, N3=yes, N4=yes, N5=yes, N6=yes,
           N7=yes, L0_1=yes, L0_2=yes, L0_4=yes, L1_0=yes, L1_3=yes, L1_5=yes,
           L2_0=yes, L2_3=yes, L2_6=yes, L3_1=yes, L3_2=yes, L3_7=yes,
           L4_0=yes, L4_5=yes, L4_6=yes, L5_1=yes, L5_4=yes, L5_7=yes,
           L6_2=yes, L6_4=yes, L6_7=yes, L7_3=yes, L7_5=yes, L7_6=yes,
           NF=0, UNCOV=no, ABS=no

/* repair of node 0 */
if !ABS && !UNCOV && !N0 action N0_R with_rate MUN
next_state N0=yes, NF--

...

/* repair of node 7 */
if !ABS && !UNCOV && !N7 action N7_R with_rate MUN
next_state N7=yes, NF--

/* repair of processor 0 */
if !ABS && !UNCOV && !P0 action P0_R with_rate MUP
next_state P0=yes, NF--
```

```
...

/* repair of processor 15 */
if !ABS && !UNCOV && !P15 action P15_R with_rate MUP
next_state P15=yes, NF--

/* repair of link from node 0 to node 1 */
if !ABS && !UNCOV && !L0_1 action L0_1_R with_rate MUL
next_state L0_1=yes, NF--

...

/* repair of link from node 7 to node 6 */
if !ABS && !UNCOV && !L7_6 action L7_6_R with_rate MUL
next_state L7_6=yes, NF--
```

mp.c

```
#include "mp.h"

#define YES 1
#define NO  0

int regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (NF==0 && !UNCOV && !ABS) return 1;
   else return 0;
}

static double speedup(int n)
{
   switch (n){
   case 0: return 0.0;
   case 1: return 1.0;
   case 2: return 1.96667;
   case 3: return 2.9;
   case 4: return 3.8;
   case 5: return 4.66667;
   case 6: return 5.5;
   case 7: return 6.3;
   case 8: return 7.06667;
   case 9: return 7.8;
   case 10: return 8.5;
   case 11: return 9.16667;
   case 12: return 9.8;
   case 13: return 10.4;
   case 14: return 10.96667;
   case 15: return 11.5;
   case 16: return 12.0;
   default: return 0.0;
   }
}
```

```
double proc_rate(int UB, int P0, int P1, int P2, int P3, int P4, int P5,
                 int P6, int P7, int P8, int P9, int P10, int P11, int P12,
                 int P13, int P14, int P15, int N0, int N1, int N2, int N3,
                 int N4, int N5, int N6, int N7, int L0_1, int L0_2, int L0_4,
                 int L1_0, int L1_3, int L1_5, int L2_0, int L2_3, int L2_6,
                 int L3_1, int L3_2, int L3_7, int L4_0, int L4_5, int L4_6,
                 int L5_1, int L5_4, int L5_7, int L6_2, int L6_4, int L6_7,
                 int L7_3, int L7_5, int L7_6, int UNCOV, int ABS)
{
   int p[16], n[8], con[8][8], node[9], max_proc, number, nnodes, clique,
       nproc, i, j, k, l;

   if (UNCOV) return speedup(0);
   if (ABS && UB) return speedup(16);
   if (ABS && !UB) return speedup(0);
   p[0] = P0; p[1] = P1; p[2] = P2; p[3] = P3; p[4] = P4; p[5] = P5; p[6] = P6;
   p[7] = P7; p[8] = P8; p[9] = P9; p[10] = P10; p[11] = P11; p[12] = P12;
   p[13] = P13; p[14] = P14; p[15] = P15; n[0] = N0; n[1] = N1; n[2] = N2;
   n[3] = N3; n[4] = N4; n[5] = N5; n[6] = N6; n[7] = N7;
   for (i = 0; i <= 7; i++) con[i][i] = n[i];
   for (i = 1; i <= 7; i++){
      if (L0_1)
         for (j = 0; j <= 7; j++)
            if (!con[0][j] && j != 0) con[0][j] = con[1][j];
      if (L0_2)
         for (j = 0; j <= 7; j++)
            if (!con[0][j] && j != 0) con[0][j] = con[2][j];
      if (L0_4)
         for (j = 0; j <= 7; j++)
            if (!con[0][j] && j != 0) con[0][j] = con[4][j];
      if (L1_0)
         for (j = 0; j <= 7; j++)
            if (!con[1][j] && j != 1) con[1][j] = con[0][j];
      if (L1_3)
         for (j = 0; j <= 7; j++)
            if (!con[1][j] && j != 1) con[1][j] = con[3][j];
      if (L1_5)
         for (j = 0; j <= 7; j++)
            if (!con[1][j] && j != 1) con[1][j] = con[5][j];
      if (L2_0)
         for (j = 0; j <= 7; j++)
            if (!con[2][j] && j != 2) con[2][j] = con[0][j];
      if (L2_3)
         for (j = 0; j <= 7; j++)
            if (!con[2][j] && j != 2) con[2][j] = con[3][j];
      if (L2_6)
         for (j = 0; j <= 7; j++)
            if (!con[2][j] && j != 2) con[2][j] = con[6][j];
      if (L3_1)
         for (j = 0; j <= 7; j++)
            if (!con[3][j] && j != 3) con[3][j] = con[1][j];
      if (L3_2)
         for (j = 0; j <= 7; j++)
```

```
            if (!con[3][j] && j != 3) con[3][j] = con[2][j];
      if (L3_7)
         for (j = 0; j <= 7; j++)
            if (!con[3][j] && j != 3) con[3][j] = con[7][j];
      if (L4_0)
         for (j = 0; j <= 7; j++)
            if (!con[4][j] && j != 4) con[4][j] = con[0][j];
      if (L4_5)
         for (j = 0; j <= 7; j++)
            if (!con[4][j] && j != 4) con[4][j] = con[5][j];
      if (L4_6)
         for (j = 0; j <= 7; j++)
            if (!con[4][j] && j != 4) con[4][j] = con[6][j];
      if (L5_1)
         for (j = 0; j <= 7; j++)
            if (!con[5][j] && j != 5) con[5][j] = con[1][j];
      if (L5_4)
         for (j = 0; j <= 7; j++)
            if (!con[5][j] && j != 5) con[5][j] = con[4][j];
      if (L5_7)
         for (j = 0; j <= 7; j++)
            if (!con[5][j] && j != 5) con[5][j] = con[7][j];
      if (L6_2)
         for (j = 0; j <= 7; j++)
            if (!con[6][j] && j != 6) con[6][j] = con[2][j];
      if (L6_4)
         for (j = 0; j <= 7; j++)
            if (!con[6][j] && j != 6) con[6][j] = con[4][j];
      if (L6_7)
         for (j = 0; j <= 7; j++)
            if (!con[6][j] && j != 6) con[6][j] = con[7][j];
      if (L7_3)
         for (j = 0; j <= 7; j++)
            if (!con[7][j] && j != 7) con[7][j] = con[3][j];
      if (L7_5)
         for (j = 0; j <= 7; j++)
            if (!con[7][j] && j != 7) con[7][j] = con[5][j];
      if (L7_6)
         for (j = 0; j <= 7; j++)
            if (!con[7][j] && j != 7) con[7][j] = con[6][j];
   }
   max_proc = 0;
   for (i = 1; i <= 255; i++){
      number = i;
      nnodes = 0;
      for (j = 0; number > 0; j++){
         l = number % 2;
         if (l == 1){
            nnodes++;
            node[nnodes] = j;
         }
         number = number/2;
      }
      clique = YES;
```

```
        for (j = 1; j <= nnodes; j++)
           for (k = 1; k <= nnodes; k++)
              if (!con[node[j]][node[k]]){
                 clique = NO;
                 break;
              }
        if (clique){
           nproc = 0;
           for (j = 1; j <= nnodes; j++){
              if (p[2*node[j]]) nproc++;
              if (p[2*node[j]+1]) nproc++;
           }
           if (nproc > max_proc) max_proc = nproc;
        }
     }
     if (max_proc == 0 && (p[0] || p[1] || p[2] || p[3] || p[4] || p[5] || p[6]
         || p[7] || p[8] || p[9] || p[10] || p[11] || p[12] || p[13] || p[14]
         || p[13])) max_proc = 1;
     return speedup(max_proc);
  }
```

For $M = 4$, the bounding rewarded CTMC models have 213,055 states and 2,072,658 transitions and the obtained bounds are tight. As numerical solution method we will use the method Bounding Transformation/Bounding Regenerative Transformation (BT/BRT) with $D_C = 1$ and regenerative state the state in which no component is failed. That regenerative state is specified in the file `mp.c` by means of the function with predefined name and prototype `regstat` (see Section 4.1, page 41). We will use as a baseline the set of model parameter values $\lambda_{\mathrm{P}} = 2 \times 10^{-5}$ h^{-1}, $\lambda_{\mathrm{N}} = 10^{-5}$ h^{-1}, $\lambda_{\mathrm{L}} = 5 \times 10^{-6}$ h^{-1}, $C_{\mathrm{P}} = 0.99$, $C_{\mathrm{N}} = 0.995$, $\mu_{\mathrm{P}} = 0.1$ h^{-1}, $\mu_{\mathrm{N}} = 0.05$ h^{-1}, $\mu_{\mathrm{L}} = 0.05$ h^{-1}, and $\mu_{\mathrm{G}} = 0.2$ h^{-1}, and will investigate how the probability that the normalized performance of the multiprocessor in the time interval $[0, t]$ is $> s$ at $t = 2$ years is enhanced by improving its maintenance in three different ways: 1) faster repair of processors ($\mu_{\mathrm{P}} = 0.2$ h^{-1}), 2) faster repair of components of the hypercube ($\mu_{\mathrm{N}} = \mu_{\mathrm{L}} = 0.1$ h^{-1}), and 3) faster repair of down systems due to an uncovered fault ($\mu_{\mathrm{G}} = 0.4$ h^{-1}). For all the considered sets of model parameter values, the chosen regenerative state is the so-called "natural" regenerative state and with that selection, the rewarded CTMC models satisfy the conditions under which the bounds given by the BT/BRT method with $D_C = 1$ should be tight [29]. The obtained results are plotted in Figure G.5. We plot the average of the lower and upper bound. The bounds are tight enough to consider the measure well computed at the plot resolution level. We can note that the most efficient way of improving the probability that the normalized performance of the multiprocessor in the time interval $[0, t]$ is $> s$ depends on the required probability level. When the normalized performance has to be guaranteed with very high probability, improving the repair of down systems due to an uncovered fault is the most efficient alternative; when the accumulated performance has to be guaranteed with moderate probability, the most efficient alternative is to improve the repair of processors; for intermediate values of the probability with which the accumulated performance has to be guaranteed, the most efficient alternative is to improve the repair of components of the hypercube. Using the simpler $\mathrm{EARR}(t) = E[(1/t) \int_0^t r_{X(\tau)}\, d\tau]$ measure would have led to the conclusion that the most efficient alternative is to improve the repair of components of the hy-

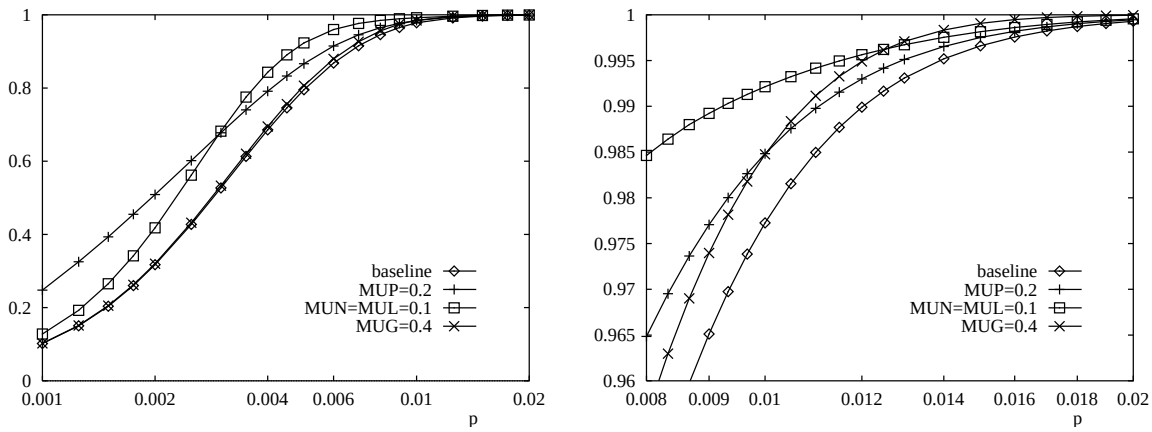

Figure G.5: $\mathrm{CRCD}(t, (12-p)t)$ as a function of p for $t = 2$ years and the baseline set of parameters and three alternatives for improving the maintenance.

Table G.3: $\mathrm{EARR}(t)$ measure for $t = 2$ years and the baseline repair rates, a set of repair rates with $\mu_{\mathrm{P}} = 0.2\ \mathrm{h}^{-1}$, a set of repair rates with $\mu_{\mathrm{N}} = \mu_{\mathrm{L}} = 0.1\ \mathrm{h}^{-1}$, and a set of repair rates with $\mu_{\mathrm{G}} = 0.4\ \mathrm{h}^{-1}$.

case	$\mathrm{EARR}(t)$
baseline	11.996558
$\mu_{\mathrm{P}} = 0.2\ \mathrm{h}^{-1}$	11.997350
$\mu_{\mathrm{N}} = \mu_{\mathrm{L}} = 0.1\ \mathrm{h}^{-1}$	11.997379
$\mu_{\mathrm{G}} = 0.4\ \mathrm{h}^{-1}$	11.996666

percube, as Table G.3 illustrates. Thus, use of the more detailed measure $\mathrm{CRCD}(t, s)$ provides interesting information to guide the maintenance of the fault-tolerant multiprocessor system.

References

[1] J. A. Carrasco. Computation of bounds for transient measures of large rewarded Markov models using regenerative randomization. *Computers and Operations Research*, 30(6):1005–1035, June 2003.

[2] V. Suñé and J. A. Carrasco. Efficient implementations of the randomization method with control of the relative error. *Computers and Operations Research*, 32:1089–1114, May 2005.

[3] B. Sericola. Availability analysis of reparairable computer systems and stationary regime detection. *IEEE Transactions on Computers*, 48(11):1166–1172, November 1999.

[4] J. A. Carrasco. Transient analysis of some rewarded Markov models using randomization with quasistationarity detection. *IEEE Trans. on Computers*, 53(9):1106–1120, September 2004.

[5] J. A. Carrasco. Transient analysis of large Markov models with absorbing states using regenerative randomization. *Communications in Statistics–Simulation and Computation*, 34(4):1037–1052, October-December 2005.

[6] J. A. Carrasco. Transient analysis of rewarded continuous time Markov models by regenerative randomization with Laplace transform inversion. *The Computer Journal*, 46(1):84–99, January 2003.

[7] J. A. Carrasco. Computationally efficient and numerically stable reliability bounds for repairable fault-tolerant systems. *IEEE Trans. on Computers*, 51(3):154–168, March 2002.

[8] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[9] P. J. Prince and J. R. Dormand. High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67–75, 1981.

[10] Byron L. Ehle. On Padé approximations to the exponential function and A-stable methods for the numerical solution of initial value problems. Technical Report CSRR 2010, Univ. of Waterloo, Dept. AACS, 1969.

[11] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, 2nd revised ed.* Springer, 1996.

[12] Jacques J. B. de Swart and Gustaf Söderlind. On the construction of error estimators for implicit Runge-Kutta methods. *J. of Comp. and App. Math.*, 86:347–358, 1997.

[13] R. S. Varga. On diagonal dominance arguments for bounding $||\boldsymbol{A}^{-1}||_\infty$. *Lin. Alg. App.*, 14:211–217, 1976.

[14] J. H. Ahlberg and E. N. Nilson. Convergence properties of the spline fit. *J. SIAM*, 11:95–104, 1963.

[15] J. M. Varah. A lower bound for the smallest singular value of a matrix. *Lin. Alg. App.*, 11:3–5, 1975.

[16] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. on Scientific Computing*, 13(2):631–644, 1992.

[17] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributerd uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[18] Winfried K. Grassmann, Michael I. Taksar, and Daniel P. Heyman. Regenerative analysis and steady state distributions for Markov chains. *Operations Research*, 33(5):1107–1116, 1985.

[19] Colm Art O'Cinneide. Relative-error bounds for the LU decomposition via the GTH algorithm. *Numerische Mathematik*, 73(4):507–519, 1996.

[20] William J. Stewart. *Computational Probability (W. Grassman, ed.)*, chapter Numerical methods for computing stationary distributions of finite irreducible Markov chains. Kluwer Academic Publishers, 2000.

[21] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, 1994.

[22] V. Suñé, J. L. Domingo, and J. A. Carrasco. Numerical iterative methods for Markovian dependability and performability models: New results and a comparison. *Performance Evaluation*, 39(1–4):99–125, February 2000.

[23] Maria Sosonkina, Layne T. Watson, Rakesh K. Kapania, and Homer F. Walker. A new adaptive GMRES algorithm for achieving high accuracy. *Num. Lin. Alg. App.*, 5(4):275–297, 1998.

[24] Y. Saad and M. H. Shultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[25] T. Dayar and N. Akar. Computing moments of first passage times to a subset of states in Markov chains. *SIAM Journal of Matrix Analysis and Applications*, 27(2):396–412, 2005.

[26] P. Heidelberger, J. K. Muppala, and K. S. Trivedi. Accelerating mean time to failure computations. *Performance Evaluation*, 27-28:627–645, October 1996.

[27] H. Nabli and B. Sericola. Performability analysis: A new algorithm. *IEEE Transactions on Computers*, 45(4):491–494, 1996.

[28] V. Suñé, J. A. Carrasco, H. Nabli, and B. Sericola. Comment on "performability analysis: A new algorithm". *IEEE Transactions on Computers*, 59(1):137–138, 2010.

[29] J. A. Carrasco. Two methods to compute bounds for the distribution of cumulative reward for large Markov models. *Performance Evaluation*, 63(12):1165–1195, December 2006.

[30] G. Rubino and B. Sericola. Interval availability analysis using denumerable Markov processes: Application to multiprocessor subject to breakdowns and repair. *IEEE Transactions on Computers*, 44(2):286–291, February 1995.

[31] C. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrica*, 26:404–413, 1934.

[32] Juan A: Carrasco. A new general-purpose method for the computation of the interval availability distribution. *INFORMS Journal on Computing*, 2012. To appear.

[33] J. A. Carrasco. An efficient and numerically stable method for computing bounds for the interval availability distribution. Technical Report DMSD_2005_1, Departament d'Enginyeria Electrònica, Universitat Politècnica de Catalunya, 2005. Available at `ftp://ftp-eel.upc.es/techreports`. To appear in *Journal of Computing*.

[34] J. A. Carrasco. Solving large interval availability models using a model transformation approach. *Computers and Operations Research*, 31(5):807–861, September 2004.

[35] J. A. Carrasco and V. Suñé. A numerical method for the evaluation of the distribution of cumulative reward till exit of a subset of transient states of a markov reward model. *IEEE Transactions on Dependable and Secure Computing*, 8(6):798–809, 2011.

[36] V. G. Kulkarni. A new class of multivariate phase type distributions. *Operations Research*, 37(1):151–158, January-February 1989.

[37] V. G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman and Hall, New York, 1995.

[38] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, Philadelphia, 1994.

[39] M. Kijima. *Markov Processes for Stochastic Modeling*. University Press, Cambridge, 1997.

[40] A. V. Aho, J. D. Ullman, and J. E. Hopcroft. *Data Structures and Algorithms*. Addison-Wesley, Boston, 1983.