Resum

Per a la realització d'aquest projecte ha estat necessari la creació d'un nombre elevat de fitxers i scripts en llenguatge C. En aquest Annex es presenta la primera part dels fitxers creats per a la validació del càlcul numèric de les mesures estocàstiques (Grup I).

Sumari

A. FITXERS I APLICACIONS DEL GRUP I _____ 5

A.1. Fitxers principals d'especificació de CMTC 5

A.2. Fitxers auxiliars d'especificació de CMTC 98

A.3. Fitxers de comandes per Maple..... 112

A.4. Fitxers d'execució de models..... 146

A. Fitxers i aplicacions del Grup I

A.1. Fitxers principals d'especificació de CMTC

```
/*
  ->Example description

  This is the nearly completely decomposable example of [1], p. 133.

  [1] W. J. Steward, "Introduction to the Numerical Solutionof Markov Chains,"
  Princeton University Press, 1994.

  ->CTMC description

  8; irreducible.

  ->What tests is the example intended for

  Testing of a particular case of the ESSRR measure using SOR.
*/

state_variables

S

production_rules

if S==1 action FROM_1 with_rate 1.0
  response TO_3 with_prob 0.149
  new_state S=3

  response TO_4 with_prob 9e-4
  new_state S=4

  response TO_6 with_prob 5e-5
  new_state S=6

  response TO_8 with_prob 5e-5
  new_state S=8
end

if S==2 action FROM_2 with_rate 1.0
  response TO_1 with_prob 0.1
  new_state S=1

  response TO_3 with_prob 0.249
  new_state S=3

  response TO_5 with_prob 9e-4
  new_state S=5

  response TO_6 with_prob 5e-5
  new_state S=6
  response TO_8 with_prob 5e-5
  new_state S=8
end

if S==3 action FROM_3 with_rate 1.0
  response TO_1 with_prob 0.1
  new_state S=1

  response TO_2 with_prob 0.8
  new_state S=2
```

```
  response TO_4 with_prob 3e-4
  new_state S=4

  response TO_7 with_prob 1e-4
  new_state S=7
end

if S==4 action FROM_4 with_rate 1.0
  response TO_2 with_prob 4e-4
  new_state S=2

  response TO_5 with_prob 0.2995
  new_state S=5

  response TO_7 with_prob 1e-4
  new_state S=7
end

if S==5 action FROM_5 with_rate 1.0
  response TO_1 with_prob 5e-4
  new_state S=1

  response TO_3 with_prob 4e-4
  new_state S=3

  response TO_4 with_prob 0.399
  new_state S=4

  response TO_6 with_prob 1e-4
  new_state S=6
end

if S==6 action FROM_6 with_rate 1.0
  response TO_2 with_prob 5e-5
  new_state S=2

  response TO_5 with_prob 5e-5
  new_state S=5

  response TO_7 with_prob 0.2499
  new_state S=7

  response TO_8 with_prob 0.15
  new_state S=8
end

if S==7 action FROM_7 with_rate 1.0
  response TO_1 with_prob 3e-5
  new_state S=1
  response TO_3 with_prob 3e-5
  new_state S=3

  response TO_4 with_prob 4e-5
  new_state S=4

  response TO_6 with_prob 0.1
  new_state S=6

  response TO_8 with_prob 0.0999
  new_state S=8
end

if S==8 action FROM_8 with_rate 1.0
  response TO_2 with_prob 5e-5
  new_state S=2

  response TO_5 with_prob 5e-5
```

```
  new_state S=5

  response TO_6 with_prob 0.1999
  new_state S=6

  response TO_7 with_prob 0.25
  new_state S=7
end


start_state
S=1

reward_rate (double)(S==1 || S==2 || S==3)
/*
   ->Example description

   Fictitious system.

   ->CTMC description

   4 states; reducible (one component is non-reachable).

   ->What tests is the example intended for

   Testing of the CCRD measure when all states have reward rate equal
   to either 1 or 0 and there are non-reachable states.

   ->Exact solution

   0<=s<t:

   CCRD(t,s)=CIAVD(t,p=s/t) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.

   s>=t:

   CCRD(t,s)=0.
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu   /* tiempo de reparacion de una replica */

state_variables

DEL, /* states with DEL=yes are not reachable */
N    /* fallo del sistema */


production_rules

if N==1  action FAIL with_rate la new_state N--

if N!=1  action  REPAIR with_rate mu new_state N++

if DEL action TO_REACH with_rate la new_state DEL=no


start_state

DEL=yes, N=1
```

```
reward_rate (double)(N==1)

initial_probability (double) (DEL==no && N==1)
/*
   ->Example description

   Example system described in [1]. The system includes NP processors
   and NM memories connected by a distributed crossbar
   network. Processors and memories fail with rates LDP and LDM,
   respectively; the crossbar network does not fail. The CTMC is
   described using two state variables: P (no. of fault-free
   processors) and M (no. of fault-free memories). The reward rate
   assigned to state (P,M) is the steady-state average number of
   active processors in the time interval [0,t],
   M*(1-(1-1/M)^P)/t. State (NP,NM) is assigned initial probability
   P0, 0<=P0<=1, states (NP-1,NM) and (NP,NM-1) are each assigned
   initial probability (1-P0)/2, and the remaining states have initial
   probability equal to zero.

   ->CTMC description

   Reducible.

   ->What tests is the example intended for

   Testing of the CCRD measure when there are more than two different
   reward rates.

   ->Exact solution

   For not too large values of NP and NM, the exact value of CCRD(t,s)
   can be computed using the recurrences described in [1]. (Maple's
   command file CCRD_3_maple.)
   [1] V. Grassi, L. Donatiello, and G. Iazeolla, "Performability Evaluation
       of Multicomponent Fault-Tolerant Systems," IEEE Trans. on Reliability,
       37(2): 216-222, June 1988.
*/

parameters
double
    T,   /* mission time */
    P0,  /* state (P,M) is assigned initial probability P0; states
          * (P-1,M) and (P,M-1) are both assigned initial probability (1-P0)/2 */
int
    NP,  /* no. of processors making up the system */
    NM,  /* initial no. of memories making up the system */
double
    LDP, /* processor failure rate in hours^{-1} */
    LDM  /* memory failure rate in hours^{-1} */


state_variables
P,       /* no. of fault-free processors */
M        /* no. of fault-free memories */


external
double REWARD(int,int,double),
double INITP(int,int,int,int,double)


production_rules

if P>0 action PROCESSOR_FAILURE with_rate P*LDP
    if P>1 response STILL_UP new_state P--
    if P==1 response SYSTEM_FAILURE new_state P--,M=0
end
```

```
if M>0 action MEMORY_FAILURE with_rate M*LDM
    if M>1 response STILL_UP new_state M--
    if M==1 response SYSTEM_FAILURE new_state M--,P=0
end


start_state
P=NP, M=NM


reward_rate REWARD(P,M,T)


initial_probability INITP(P,M,NP,NM,P0)


/*
   ->Example description

   The system includes NP processors and NM memories connected by a
   distributed crossbar network. Processors and memories fail with
   rates LDP and LDM, respectively; the crossbar network does not
   fail. The CTMC is described using two state variables: P (no. of
   fault-free processors) and M (no. of fault-free memories). The
   reward rate assigned to state (P,M) is 1 if it is an up state
   (i.e. both P and M are larger than 0) and 0 otherwise. State
   (NP,NM) is assigned initial probability P0, 0<=P0<=1, states
   (NP-1,NM) and (NP,NM-1) are each assigned initial probability
   (1-P0)/2, and the remaining states have initial probability equal
   to zero. The system is as described in [1] with the exception of
   the reward rates.

   ->CTMC description

   Reducible.

   ->What tests is the example intended for

   Testing of the CCRD measure when there are more than two different
   reward rates.

   ->Exact solution

   For not too large values of NP and NM, the exact value of CCRD(t,s)
   can be computed using the recurrences described in [1]. (Maple's
   command file CCRD_4_maple.)

   [1] V. Grassi, L. Donatiello, and G. Iazeolla, "Performability Evaluation
       of Multicomponent Fault-Tolerant Systems," IEEE Trans. on Reliability,
       37(2): 216-222, June 1988.
*/

parameters
double
    T,   /* mission time */
    P0,  /* state (P,M) is assigned initial probability P0; states
          * (P-1,M) and (P,M-1) are both assigned initial probability (1-P0)/2 */
int
    NP,  /* no. of processors making up the system */
    NM,  /* initial no. of memories making up the system */
double
    LDP, /* processor failure rate in hours^{-1} */
    LDM  /* memory failure rate in hours^{-1} */


state_variables
P,       /* no. of fault-free processors */
```

```
M        /* no. of fault-free memories */


external
double REWARD(int,int,double),
double INITP(int,int,int,int,double)


production_rules

if P>0 action PROCESSOR_FAILURE with_rate P*LDP
    if P>1 response STILL_UP new_state P--
    if P==1 response SYSTEM_FAILURE new_state P--,M=0
end

if M>0 action MEMORY_FAILURE with_rate M*LDM
    if M>1 response STILL_UP new_state M--
    if M==1 response SYSTEM_FAILURE new_state M--,P=0
end


start_state
P=NP, M=NM


reward_rate REWARD(P,M,T)


initial_probability INITP(P,M,NP,NM,P0)
/*
  ->Example description

  Non fault-tolerant system that fails with rate la and is repaired
  with rate mu.

  ->CTMC description

  2 states; irreducible

  ->What tests is the example intended for

  Testing of the CRCD measure using any of the methods offered by METFAC.

  ->Exact solution.

  0<=s<t:

   CCRD(t,s)=CIAVD(t,p=s/t) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.

   s>=t:

   CCRD(t,s)=0.

*/

parameters

double
la,  /* failure rate */
mu   /* repair rate */

state_variables
```

```
N    /* 1 if system operational and 0 otherwise */

production_rules

if N==1 action FAILURE with_rate la
new_state N--

if N==0 action REPAIR with_rate mu
new_state N++

start_state

N=1

reward_rate (double)(N==1)

initial_probability (double)(N==1)
/*
   ->Example description

   System consisting of a single component that fails with rate ld
   and is repaired with rate mu. Both the non-failed and failed states
   are "split" into several states making up a lumpable partition.

   ->CTMC description

   2*(ND0+1) states; irreducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using any of the numerical methods
   offered by METFAC2.1

  ->Exact solution

   CIAVD(t,p) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})                          (1)
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.
*/

parameters

int
stat_start,  /* 1 to start generating the CTMC from state (N=1,ND=ND0);
                <>1 to start from state (N=0,ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu    /* tiempo de reparacion de una replica */


state_variables

N,ND


production_rules

if N==1 action FAIL with_rate la new_state N=0,ND=ND0

if N==0  action  REPAIR with_rate mu new_state N=1,ND=ND0

if ND>0 action FAIL_LEFT with_rate la*ND new_state ND--
```

```
if ND<ND0 action REPAIR_RIGHT with_rate mu new_state ND++


start_state

N=(stat_start==1),ND=ND0

reward_rate (double)(N==1)

initial_probability (double)(N==1 && ND==ND0)
 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   2*(N_0+1)+1; reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method for the
   particular case in which there is an absorbing state and another trapping
   component and the U'_S set is not empty.

  ->Exact solution

   Maple worksheet CIAVD_10_maple.
*/

parameters

int
N_0,

double
la,  /* failure rate */
la_ph, /* phase change rate */
mu,  /* repair rate */
C,   /* coverage probability */
P0

state_variables

PH,N,ABS


production_rules

if PH==1 && N>0 && !ABS action FAIL with_rate N*la
  response COV with_prob C
  new_state N--

  response UNCOV with_prob 1-C

  new_state N=N_0,ABS=yes
end

if PH==1 && !ABS && N>0 action PHASE_CHANGE with_rate la_ph
new_state PH=0

if PH==0 && N>0 && !ABS action FAIL with_rate N*la
new_state N--

if N<N_0 && !ABS action REPAIR with_rate mu
new_state N++

if PH==0 && N==0 && !ABS action EXTRA_REPAIR with_rate mu
```

```
new_state N+=2


start_state

PH=1,N=N_0,ABS=no

reward_rate (double)(N>1 && !ABS)

initial_probability P0*(PH==1 && N==N_0 && !ABS)
                      +0.5*(1-P0)*(PH==0 && N==N_0 && !ABS)
                   +0.5*(1-P0)*(PH==0 && N==0 && !ABS)


 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   2*(N_0+1)+1; reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method for the
particular case in which there is an absorbing state, the U'_S set is empty and
there is a transition rate from the regenerative state to the absorbing state.

  ->Exact solution

   Maple worksheet CIAVD_11_maple.
*/

parameters

int
N_0,

double
la,  /* failure rate */
mu,  /* repair rate */
C,   /* coverage probability */
P0


state_variables

N,ABS

production_rules

if N>0 && !ABS action FAIL with_rate N*la
  response COV with_prob C
  new_state N--

  response UNCOV with_prob 1-C
  new_state N=N_0,ABS=yes
end

if N<N_0 && !ABS action REPAIR with_rate mu
new_state N++

start_state

N=N_0,ABS=no
```

```
reward_rate (double)(N==N_0 && !ABS)

initial_probability P0*(N==N_0 && !ABS)
                    +((1.0-P0)/2.0)*(N==N_0 && ABS)
                  +((1.0-P0)/2.0)*(N==N_0-1 && !ABS)

 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   2*(N_0+1)+1; reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method for the
particular case in which there is an absorbing state, the U'_S set is not empty,
the regenerative state is a down state which initial probability is smaller than
one, and there is a transition rate from some non-absorbing down state to the
absorbing state.

  ->Exact solution

   Maple worksheet CIAVD_11_maple.
*/

parameters
int
N_0,

double
la,  /* failure rate */
mu,  /* repair rate */
C,   /* coverage probability */
P0


state_variables

N,ABS


production_rules

if N>0 && !ABS action FAIL with_rate N*la
  response COV with_prob C
  new_state N--

  response UNCOV with_prob 1-C
  new_state N=N_0,ABS=yes
end

if N<N_0 && !ABS action REPAIR with_rate mu
new_state N++


start_state

N=N_0,ABS=no


reward_rate (double)(N==N_0 && !ABS)


initial_probability P0*(N==N_0 && !ABS)
                    +((1.0-P0)/2.0)*(N==N_0-1 && !ABS)
```

```
                +((1.0-P0)/2.0)*(N==N_0-2 && !ABS)

/*
   ->Example description

   System consisting of a single component that fails with rate ld
   and is repaired with rate mu.

   ->CTMC description

   2 states; irreducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using any of the numerical methods
   offered by METFAC2.1

  ->Exact solution

   CIAVD(t,p) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})                          (1)
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.
*/

parameters

int
stat_start,  /* 1 to start generating the CTMC from state (N=1,ND=ND0);
                <>1 to start from state (N=0,ND=ND0) */

double
la,  /* tasa de fallo de una replica */
mu    /* tiempo de reparacion de una replica */


state_variables

N


production_rules

if N==1 action FAIL with_rate la new_state N=0

if N==0  action  REPAIR with_rate mu new_state N=1


start_state

N=(stat_start==1)

reward_rate (double)(N==1)

initial_probability (double)(N==1)
/*
   ->Example description

   Fictitious system.

   ->CTMC description

   4 states; reducible (one component is non-reachable).

   ->What tests is the example intended for
```

```
   Testing of the CIAVD measure when there are non-reachable states
   using any of the numerical methods offered by METFAC2.1.

   ->Exact solution

   CIAVD(t,p) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})                          (1)
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu   /* tiempo de reparacion de una replica */

state_variables

DEL, /* states with DEL=yes are not reachable */
N    /* fallo del sistema */


production_rules

if N==1  action FAIL with_rate la new_state N--

if N!=1  action  REPAIR with_rate mu new_state N++

if DEL action TO_REACH with_rate la new_state DEL=no


start_state

DEL=yes, N=1

reward_rate (double)(N==1)

initial_probability (double) (DEL==no && N==1)
/*
   ->Example description

   The system includes NP processors and NM memories connected by a
   distributed crossbar network. Processors and memories fail with
   rates LDP and LDM, respectively; the crossbar network does not
   fail. The CTMC is described using two state variables: P (no. of
   fault-free processors) and M (no. of fault-free memories). The
   reward rate assigned to state (P,M) is 1 if it is an up state
   (i.e. both P and M are larger than 0) and 0 otherwise. State
   (NP,NM) is assigned initial probability P0, 0<=P0<=1, states
   (NP-1,NM) and (NP,NM-1) are each assigned initial probability
   (1-P0)/2, and the remaining states have initial probability equal
   to zero. The system is as described in [1] with the exception of
   the reward rates.

   ->CTMC description

   Reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure when there the initial probability is not
   concentrated on a single state.

   ->Exact solution
```

```
   For not too large values of NP and NM, the exact value of CIAVD(t,p)
   can be computed using the recurrences described in [1] with s=t*p.
   (Maple's command file CIAVD_3_maple.)

   [1] V. Grassi, L. Donatiello, and G. Iazeolla, "Performability Evaluation
       of Multicomponent Fault-Tolerant Systems," IEEE Trans. on Reliability,
       37(2): 216-222, June 1988.
*/

parameters
double
    P0,  /* state (P,M) is assigned initial probability P0; states
          * (P-1,M) and (P,M-1) are both assigned initial probability (1-P0)/2 */
int
    NP,  /* no. of processors making up the system */
    NM,  /* initial no. of memories making up the system */
double
    LDP, /* processor failure rate in hours^{-1} */
    LDM  /* memory failure rate in hours^{-1} */


state_variables
P,       /* no. of fault-free processors */
M        /* no. of fault-free memories */


external
double REWARD(int,int),
double INITP(int,int,int,int,double)


production_rules

if P>0 action PROCESSOR_FAILURE with_rate P*LDP
    if P>1 response STILL_UP new_state P--
    if P==1 response SYSTEM_FAILURE new_state P--,M=0
end

if M>0 action MEMORY_FAILURE with_rate M*LDM
    if M>1 response STILL_UP new_state M--
    if M==1 response SYSTEM_FAILURE new_state M--,P=0
end


start_state
P=NP, M=NM


reward_rate REWARD(P,M)


initial_probability INITP(P,M,NP,NM,P0)
/*
   ->Example description

   System consisting of a single component that fails with rate ld
   and is repaired with rate mu. Both the non-failed and failed states
   are "split" into several states making up a lumpable partition.

   ->CTMC description

   2*(ND0+1) states; irreducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using any of the numerical methods offered by
```

```
   METFAC2.1 for two particular cases: 1) all states have reward rate equal to 1,
and 2) all states have reward rate equal to 0.

  ->Exact solution

   CIAVD(t,p) = 1 in case 1), t>=0, p<1, and 0 in case 2), t>=0, p<1.
*/

parameters

int
stat_start,  /* 1 to start generating the CTMC from state (N=1,ND=ND0);
                <>1 to start from state (N=0,ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
rw   /* reward rate assigned to each state */


state_variables

N,ND

production_rules

if N==1 action FAIL with_rate la new_state N=0,ND=ND0
if N==0  action  REPAIR with_rate mu new_state N=1,ND=ND0
if ND>0 action FAIL_LEFT with_rate la*ND new_state ND--
if ND<ND0 action REPAIR_RIGHT with_rate mu new_state ND++

start_state

N=(stat_start==1),ND=ND0

reward_rate rw

initial_probability (double)(N==1 && ND==ND0)
/*
   ->Example description

   System consisting of a single component that fails with rate ld
   and is repaired with rate mu. Both the non-failed and failed states
   are "split" into several states making up a lumpable partition.

   ->CTMC description

   2*(ND0+1) states; irreducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method
   for the particular case in which the regenerative state is
   down.

  ->Exact solution

   CIAVD(t,p) = e^{-la*t*p}*(1+f*\sqrt{la*mu*p*t}),
   f=\int_0^{t(1-p)} (e^{-mu*y}/\sqrt{y})                          (1)
              I_1(2*\sqrt{la*mu*p*t*y}) dy,

   where I_1 is the modified Bessel function of the first kind and
   order equal to 1.
*/

parameters
```

```
int
stat_start,  /* 1 to start generating the CTMC from state (N=1,ND=ND0);
                <>1 to start from state (N=0,ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu    /* tiempo de reparacion de una replica */


state_variables

N,ND


production_rules

if N==1 action FAIL with_rate la new_state N=0,ND=ND0
if N==0  action  REPAIR with_rate mu new_state N=1,ND=ND0
if ND>0 action FAIL_LEFT with_rate la*ND new_state ND--
if ND<ND0 action REPAIR_RIGHT with_rate mu new_state ND++

start_state

N=(stat_start==1),ND=ND0

reward_rate (double)(N==1)

initial_probability (double)(N==1 && ND==ND0)
/*
   ->Example description

   System consisting of a single component that fails with rate ld
   and is repaired with rate mu. Both the non-failed and failed states
   are "split" into several states making up a lumpable partition.

   ->CTMC description

   2*(ND0+1) states; irreducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method
   for the particular case in which down states have non-zero
   initial probability

  ->Exact solution

   Maple worksheet CIAVD_6_maple.
*/

parameters
int
stat_start,  /* 1 to start generating the CTMC from state (N=1,ND=ND0);
                <>1 to start from state (N=0,ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
P0   /* init. prob. is P0 for state (N==1,ND=ND0) and 1-P0 for
        state (N==0,ND=ND0) */


state_variables

N,ND
```

```
production_rules

if N==1 action FAIL with_rate la new_state N=0,ND=ND0
if N==0  action  REPAIR with_rate mu new_state N=1,ND=ND0
if ND>0 action FAIL_LEFT with_rate la*ND new_state ND--
if ND<ND0 action REPAIR_RIGHT with_rate mu new_state ND++

/* transf-based method requires the existence of at least
   one transition rate from a down state with non-null
   initial probability to any up state different from the
   regenerative one => if eps is the absolute error and
   tmax the time epoch, add a tiny transition rate
   <=1E-10*eps/(2*tmax) */
if N==0 && ND==ND0 action SPECIAL with_rate 5E-21 new_state N=1,ND=0


start_state

N=(stat_start==1),ND=ND0

reward_rate (double)(N==1)

initial_probability P0*(N==1 && ND==ND0)+(1-P0)*(N==0 && ND==ND0)
 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   2*(ND0_INIT+1)+2*(ND1_INIT+1); reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method
   for the particular case in which there is an absorbing state and the initial
probability of the regenerative state is less than one.

  ->Exact solution

   Since once the system has entered the subset of down states it leaves such a
subset never again,

   CIAVD(t,p)=ETRR(pt). (Maple worksheet CIAVD_7_maple.)
*/

parameters

int
ND0_INIT,
ND1_INIT,

double
LA,
MU,
P0

state_variables

N,ND0,ND1,ABS

production_rules

if !ABS && N==1 && ND0==ND0_INIT action FAIL with_rate LA new_state N--

if !ABS && N==1 && ND0>0 action FAIL_UP with_rate LA new_state ND0--
```

```
if !ABS && N==1 && ND0<ND0_INIT action REPAIR_DOWN with_rate MU new_state ND0++

if !ABS && N==0 && ND1>0 action FAIL_UP with_rate LA new_state ND1--

if !ABS && N==0 && ND1<ND1_INIT action REPAIR_DOWN with_rate MU new_state ND1++

if !ABS && N==0 && ND1==ND1_INIT action ABS with_rate LA new_state ABS=yes

start_state

N=1,ND0=0,ND1=ND1_INIT,ABS=no

reward_rate (double)(N==1)

initial_probability P0*(!ABS && N==1 && ND0==ND0_INIT-1)+(1-P0)*(!ABS && N==0 &&
ND1==ND1_INIT)
 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   ND0+3; reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method for the
particular case in which there is an absorbing state, the U'_S set is empty and
the initial probability of the regenerative state is less than one.

  ->Exact solution

   Maple worksheet CIAVD_8_maple.
*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
P0   /* init. prob. is P0 for the state N==1,ND=ND0-1,ABS=no and 1-P0 for
        the state N==0,ND=ND0,ABS=no */


state_variables

N,ND,ABS


production_rules

if N==1 && !ABS action FAIL with_rate la new_state N=0,ND=ND0
if N==0 && ND==ND0 && !ABS action  REPAIR with_rate mu new_state N=1,ND=ND0
if N==0 && ND>0 && !ABS action FAIL_LEFT with_rate la*ND new_state ND--
if N==0 && ND<ND0 && !ABS action REPAIR_RIGHT with_rate mu new_state ND++
if N==0 && ND==ND0 && !ABS action ABS with_rate la new_state ABS=yes

start_state

N=1,ND=ND0,ABS=no

reward_rate (double)(N==1 && ND==ND0 && !ABS)
```

```
initial_probability P0*(N==1 && ND==ND0 && !ABS)+(1-P0)*(N==0 && ND==ND0 &&
!ABS)/2.0
                    +(1-P0)*(N==0 && ND==ND0 && ABS)/2.0
 /*
   ->Example description

   Fictitious system.

   ->CTMC description

   N_0+2; reducible.

   ->What tests is the example intended for

   Testing of the CIAVD measure using the transformation-based method for the
   particular case in which there is an absorbing state and no other
   trapping component and the U'_S set is not empty.

  ->Exact solution

   Maple worksheet CIAVD_9_maple.
*/

parameters

int
N_0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* coverage probability */
P0


state_variables

N,ABS


production_rules

if N>0 && !ABS action FAIL with_rate N*la
  response COV with_prob C
  new_state N--

  response UNCOV with_prob 1-C
  new_state N=N_0,ABS=yes
end

if N<N_0 && !ABS action REPAIR with_rate mu
new_state N++


start_state

N=N_0,ABS=no


reward_rate (double)(N>0 && !ABS)


initial_probability P0*(N==N_0 && !ABS)+(1-P0)*(N==N_0 && ABS)


/*
  ->Example description
```

```
  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the CRDDS measure when all states in the subset have
  reward rate equal to 1 using any of the numerical methods offered by
  METFAC2.1.

  ->Exact solution.

  It can be computed with Maple (command file: CRDDS_1_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C    /* factor de cobertura */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
```

```
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(N>=1 && !F)

initial_probability (double)(N==2 && F==no && ND==ND0)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the CRDDS measure when not all states in the subset have
  reward rate equal to 1 (and there is no state with 0 reward rate)
  using any of the numerical methods offered by METFAC2.1.

  ->Exact solution.

  It can be computed with Maple (command file: CRDDS_2_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C    /* factor de cobertura */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--
```

```
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(N)

initial_probability (double)(N==2 && F==no && ND==ND0)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the CRDDS measure when all states in the subset have
  reward rate equal to either 1 or 0 using any of the numerical
  methods offered by METFAC2.1 not based on regenerative
  randomization.

  ->Exact solution.

  It can be computed with Maple (command file: CRDDS_3_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C    /* factor de cobertura */

state_variables
```

```
N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(N==2 && !F)

initial_probability (double)(N==2 && F==no && ND==ND0)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. There are several non-reachable states.

  ->CTMC description

  8 states; reducible (the transient component is non-reachable).

  ->What tests is the example intended for

  Testing of the CRDDS measure when all reachable states in the subset
  have reward rate equal to 1 and there are non-reachable states using
  any of the numerical methods offered by METFAC2.1.

  ->Exact solution.

  It can be computed with Maple (command file: CRDDS_4_maple).

*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */
```

```
state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,   /* numero de replicas en marcha */
F    /* fallo del sistema yes/no */

production_rules

/* adds a transition rate from state (DEL=yes, N, F), N>=1, F=no
 * to state (DEL=no, N, F) */
if DEL && N>=1 && !F action TO_REACH with_rate la*N new_state DEL=no

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

DEL=yes, N=2, F= no

reward_rate (double)(N>=1 && !F)

initial_probability (double) (DEL==no && N==2 && F==no)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. There are several non-reachable states.

  ->CTMC description

  8 states; reducible (the transient component is non-reachable).

  ->What tests is the example intended for

  Testing of the CRDDS measure when all reachable states in the subset
  have reward rate equal to 0.

  ->Exact solution.

  CRDDS(s)=1 for all s>=0.

*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */
```

```
state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,   /* numero de replicas en marcha */
F    /* fallo del sistema yes/no */

production_rules

/* adds a transition rate from state (DEL=yes, N, F), N>=1, F=no
 * to state (DEL=no, N, F) */
if DEL && N>=1 && !F action TO_REACH with_rate la*N new_state DEL=no

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

DEL=yes, N=2, F= no

reward_rate (double)0

initial_probability (double) (DEL==no && N==2 && F==no)
/*
  ->Example description

  Fictitious system.

  ->CTMC description

  The CTMC has 2 transient components and 2 trapping ones. The first
  transient component consists of states (S,N) with S={1,2},
  N={0,...,N0}; the second transient component consists of states
  (S,N), with S={3,4}, N={0,...,N0}; the first trapping component
  consists of states (S,N), with S={5,6}, N={0,...,N0}; and the second
  trapping component consists of states (S,N), with S={7,8},
  N={0,...,N0}. The transition rates are as follows:

  ld_a from (1,N) to (2,N0)

  mu_a from (2,N) to (1,N0)

  ld_lr from (2,N) to (3,N0)

  ld_udl from (2,N) to (5,N0)

  ld_b from (3,N) to (4,N0)

  mu_b from (4,N) to (3,N0)

  ld_udr from (4,N) to (7,N0)

  ld_c from (5,N) to (6,N0)
```

```
  mu_c from (6,N) to (5,N0)

  ld_d from (7,N) to (8,N0)

  mu_d from (8,N) to (7,N0)

  ld from (S,N) to (S,N-1), S={1,2,3,4,5,6,7,8}, N>0

  mu from (S,N) to (S,N+1), S={1,2,3,4,5,6,7,8}, N<N0

  State (1,N0) has initial probability P0, state (3,N0) has initial
  probability 2*(1-P0)/5, state (5,N0) has initial probability
  2*(1-P0)/5, state (7,N0) has initial probability (1-P0)/5, and the
  remaining states have 0 initial probability.

  States (5,N) and (7,N) have reward rate equal to 1; the remaining
  states have reward rate equal to 0.

  ->What tests is the example intended for

  Testing of the CRDDS measure (subset: S={5,7}) using any of the
  numerical methods offered by METFAC2.1 that do not require the CTMC
  to be irreducible. The CTMC can be made arbitrarily large by taking
  large values for the N0 parameter.

  ->Exact solution

  States (S,N), 0<=N<=N0 can be lumped into a single state and,
  therefore, the exact value of the CRDDS measure can be computed
  considering a CTMC with eight states S=1, 2, ..., 8, reward rate
  equal to 1 for states S=5 and S=7, and 0 for the remaining states,
  initial probability P0 for state S=1, 2*(1-P0)/5 for state S=3,
  2*(1-P0)/5 for state S=5, (1-P0)/5 for state S=7, and 0 for the
  remaining states, and the following transition rates:

  ld_a from S=1 to S=2

  mu_a from S=2 to S=1

  ld_lr from S=2 to S=3

  ld_udl from S=2 to S=5

  ld_b from S=3 to S=4

  mu_b from S=4 to S=3

  ld_c from S=5 to S=6

  mu_c from S=6 to S=5

  ld_udr from S=4 to S=7

  ld_d from S=7 to S=8

  mu_d from S=8 to S=8

  The exact value of the measure can be computed with the aid of
  the Maple program. (Command file: CRDDS_6_maple.)

*/

parameters

int
N0,
```

```
double
P0,
ld_a,
mu_a,
ld_b,
mu_b,
ld_c,
mu_c,
ld_d,
mu_d,
ld_lr,
ld_udl,
ld_udr,
ld,
mu


state_variables
S,N


production_rules

if S==1 action S_1_TO_S_2 with_rate ld_a new_state S=2, N=N0
if S==2 action S_2_TO_S_1 with_rate mu_a new_state S=1, N=N0
if S==2 action S_2_TO_S_3 with_rate ld_lr new_state S=3, N=N0
if S==2 action S_2_TO_S_5 with_rate ld_udl new_state S=5, N=N0
if S==3 action S_3_TO_S_4 with_rate ld_b new_state S=4, N=N0
if S==4 action S_4_TO_S_3 with_rate mu_b new_state S=3, N=N0
if S==4 action S_4_TO_S_7 with_rate ld_udr new_state S=7, N=N0
if S==5 action S_5_TO_S_6 with_rate ld_c new_state S=6, N=N0
if S==6 action S_6_TO_S_5 with_rate mu_c new_state S=5, N=N0
if S==7 action S_7_TO_S_8 with_rate ld_d new_state S=8, N=N0
if S==8 action S_8_TO_S_7 with_rate mu_d new_state S=7, N=N0

if N>0 action RIGTH with_rate ld new_state N--
if N<N0 action LEFT with_rate mu new_state N++


start_state
S=1, N=N0

reward_rate (double)(S==5 || S==7)

initial_probability
(double)
(P0*(S==1 && N==N0)
+(2*(1.0-P0)/5)*(S==3 && N==N0)
+(2*(1.0-P0)/5)*(S==5 && N==N0)
+((1.0-P0)/5)*(S==7 && N==N0))
/*
  ->Example description

  Fictitious system.

  ->CTMC description

  The CTMC has 2 transient components and 2 trapping ones. The first
  transient component consists of states (S,N) with S={1,2},
  N={0,...,N0}; the second transient component consists of states
  (S,N), with S={3,4}, N={0,...,N0}; the first trapping component
  consists of states (S,N), with S={5,6}, N={0,...,N0}; and the second
  trapping component consists of states (S,N), with S={7,8},
  N={0,...,N0}. The transition rates are as follows:

  ld_a from (1,N) to (2,N0)

  mu_a from (2,N) to (1,N0)
```

```
ld_lr from (2,N) to (3,N0)

ld_udl from (2,N) to (5,N0)

ld_b from (3,N) to (4,N0)

mu_b from (4,N) to (3,N0)

ld_udr from (4,N) to (7,N0)

ld_c from (5,N) to (6,N0)

mu_c from (6,N) to (5,N0)

ld_d from (7,N) to (8,N0)

mu_d from (8,N) to (7,N0)

ld from (S,N) to (S,N-1), S={1,2,3,4,5,6,7,8}, N>0

mu from (S,N) to (S,N+1), S={1,2,3,4,5,6,7,8}, N<N0

State (1,N0) has initial probability P0, state (3,N0) has initial
probability 2*(1-P0)/5, state (5,N0) has initial probability
2*(1-P0)/5, state (7,N0) has initial probability (1-P0)/5, and the
remaining states have 0 initial probability.

States (5,N) and (7,N) have reward rate equal to 1; the remaining
states have reward rate equal to 0.

->What tests is the example intended for

Testing of the CRDDS measure (subset: S={5,7}) using any of the
numerical methods offered by METFAC2.1 that do not require the CTMC
to be irreducible. The CTMC can be made arbitrarily large by taking
large values for the N0 parameter. The example considers the
particular case in which CRDTE has to be computed only for one of
the two trapping components.

->Exact solution

States (S,N), 0<=N<=N0 can be lumped into a single state and,
therefore, the exact value of the CRDDS measure can be computed
considering a CTMC with eight states S=1, 2, ..., 8, reward rate
equal to 1 for states S=5 and S=7, and 0 for the remaining states,
initial probability P0 for state S=1, 2*(1-P0)/5 for state S=3,
2*(1-P0)/5 for state S=5, (1-P0)/5 for state S=7, and 0 for the
remaining states, and the following transition rates:

ld_a from S=1 to S=2

mu_a from S=2 to S=1

ld_lr from S=2 to S=3

ld_udl from S=2 to S=5

ld_b from S=3 to S=4

mu_b from S=4 to S=3

ld_c from S=5 to S=6

mu_c from S=6 to S=5

ld_udr from S=4 to S=7
```

```
  ld_d from S=7 to S=8

  mu_d from S=8 to S=8

  The exact value of the measure can be computed with the aid of
  the Maple program. (Command file: CRDDS_7_maple.)

*/

parameters

int
N0,

double
P0,
ld_a,
mu_a,
ld_b,
mu_b,
ld_c,
mu_c,
ld_d,
mu_d,
ld_lr,
ld_udl,
ld_udr,
ld,

mu


state_variables
S,N


production_rules

if S==1 action S_1_TO_S_2 with_rate ld_a new_state S=2, N=N0
if S==2 action S_2_TO_S_1 with_rate mu_a new_state S=1, N=N0
if S==2 action S_2_TO_S_3 with_rate ld_lr new_state S=3, N=N0
if S==2 action S_2_TO_S_5 with_rate ld_udl new_state S=5, N=N0
if S==3 action S_3_TO_S_4 with_rate ld_b new_state S=4, N=N0
if S==4 action S_4_TO_S_3 with_rate mu_b new_state S=3, N=N0
if S==4 action S_4_TO_S_7 with_rate ld_udr new_state S=7, N=N0
if S==5 action S_5_TO_S_6 with_rate ld_c new_state S=6, N=N0
if S==6 action S_6_TO_S_5 with_rate mu_c new_state S=5, N=N0
if S==7 action S_7_TO_S_8 with_rate ld_d new_state S=8, N=N0
if S==8 action S_8_TO_S_7 with_rate mu_d new_state S=7, N=N0

if N>0 action RIGTH with_rate ld new_state N--
if N<N0 action LEFT with_rate mu new_state N++


start_state
S=1, N=N0

reward_rate (double)(S==5)

initial_probability
(double)
(P0*(S==1 && N==N0)
+(2*(1.0-P0)/5)*(S==3 && N==N0)
+(2*(1.0-P0)/5)*(S==5 && N==N0)
+((1.0-P0)/5)*(S==7 && N==N0))
/*
  ->Example description
```

```
  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the CRDDS measure when all states in the subset have
  reward rate equal to 0 using any of the numerical methods offered by
  METFAC2.1.

  ->Exact solution.

  It can be computed with Maple (command file: CRDDS_1_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C    /* factor de cobertura */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
```

```
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)0

initial_probability (double)(N==2 && F==no && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There are two transient components, one
  of which is no reachable, and one absorbing state.)

  ->What tests is the example intended for

  Testing of the CRDTE measure when all reward rates are equal to one
  and there are non-reachable states using any of the methods offered
  by METFAC.

  ->Exact solution.

  It can be obtained with Maple. (Command file: CRDTE_1_maple.)

*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N
```

```
   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability
    (double)(P0*(DEL==no && N==2 && ND==ND0)+(1-P0)*(DEL==no && N==1 && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.)

  ->CTMC description

  2*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the CRDTE measure when all reward rates are equal to one
  using any of the methods offered by METFAC.

  ->Exact solution.

  It can be obtained with Maple. (Command file: CRDTE_2_maple.)

*/

parameters

int
stat_start,  /* the start state is (N=stat_start, ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,          /* numero de replicas en marcha */
ND
```

```
production_rules

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, ND=ND0

reward_rate (double)(N>0)

initial_probability
    (double)(P0*(N==2 && ND==ND0)+(1-P0)*(N==1 && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.)

  ->CTMC description

  2*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the CRDTE measure when all reward rates are equal to one
  using any of the methods offered by METFAC.

  ->Exact solution.

  It can be obtained with Maple. (Command file: CRDTE_3_maple.)

*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables
```

```
N,          /* numero de replicas en marcha */
ND

production_rules

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability
    (double)(P0*(N==2 && ND==ND0)+(1-P0)*(N==1 && ND==ND0))




/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.)

  ->CTMC description

  2*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the CRDTE measure when not all reward rates are equal to
  one and neither of them is equal to zero using any of the methods
  offered by METFAC.

  ->Exact solution.

  It can be obtained with Maple. (Command file: CRDTE_4_maple.)

*/

parameters

int
stat_start,  /* the start state is (N=stat_start, ND=ND0) */
```

```
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,          /* numero de replicas en marcha */
ND

production_rules

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, ND=ND0

reward_rate (double)(N)

initial_probability
    (double)(P0*(N==2 && ND==ND0)+(1-P0)*(N==1 && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.)

  ->CTMC description

  2*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the CRDTE measure when not all reward rates are equal to
  one and some of them are equal to zero using any of the methods
  offered by METFAC.

  ->Exact solution.

  It can be obtained with Maple. (Command file: CRDTE_5_maple.)

*/
```

```
parameters

int
stat_start,  /* the start state is (N=stat_start, ND=ND0) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,          /* numero de replicas en marcha */
ND

production_rules

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, ND=ND0

reward_rate (double)(2*(N==2))


initial_probability
    (double)(P0*(N==2 && ND==ND0)+(1-P0)*(N==1 && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There are two transient components, one
  of which is not reachable, and one absorbing state.)


  ->What tests is the example intended for

  Testing of the CRDTE measure when all reward rates are equal to zero and
```

```
  there are non-reachable states using any of the methods offered by METFAC.

  ->Exact solution.

  CRDTE(s)=1 for all s>=0.
*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)0

initial_probability
    (double)(P0*(DEL==no && N==2 && ND==ND0)+(1-P0)*(DEL==no && N==1 && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description 8 states; reducible: there is a transient,
```

```
  non-reachable component (the 4 states with DEL=yes) and a trapping,
  reachable component (the 4 states with DEL=no)

  ->What tests is the example intended for

  Testing of the EARR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states.

  ->Exact solution.

  It can be computed with Maple (command file: EARR_1_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0


start_state

N=stat_start, F= no, ND=ND0
```

```
reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and coverage
  probability C; the system can only be repaired if there is one unfailed unit
  and the failure of the second one was covered. Moreover, the state with an
  uncovered failure is "split" into several "fake" states in such a way that
  given the reward rate structure, the exact value of the measure can be
  computed regarding those states as lumped into one state. (i.e. the
  partition is lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using Regenerative Randomization and
  Regenerative Randomization with Laplace Transforms.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_10_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0



state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end
```

```
if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if F && ND>0 action with_rate la_d new_state ND--
if F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate 1e8*(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Non fault-tolerant system that fails with rate la and is repaired
  with rate mu.

  ->CTMC description

  2 states; irreducible

  ->What tests is the example intended for

  Testing of the EARR measure using any of the methods offered by METFAC.

  ->Exact solution.

  EARR(t)=la/(la+mu)-(la/(t*(la+mu)^2))*(1-exp(-(la+mu)*t))

*/

parameters

double
la,  /* failure rate */
mu   /* repair rate */

state_variables

N    /* 1 if system operational and 0 otherwise */

production_rules

if N==1 action FAILURE with_rate la
new_state N--

if N==0 action REPAIR with_rate mu
new_state N++

start_state

N=1

reward_rate (double)(N==0)

initial_probability (double)(N==1)


/*
   ->Example description
```

```
   System consisting of a single component that fails with rate ld
   and is repaired with rate mu. Both the non-failed and failed states
   are "split" into several states making up a lumpable partition.

   ->CTMC description

   2*(ND0+1) states; irreducible.

   ->What tests is the example intended for

   Testing of the EARR measure using any of the numerical methods
   offered by METFAC2.1 that do not require the CTMC to have absorbing
   states.

  ->Exact solution

   EARR(t)=ld/(ld+mu)-ld/(t*(ld+mu)^2)*(1-e^(-(ld+mu)*t))
*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu    /* tiempo de reparacion de una replica */


state_variables

N,ND


production_rules


if N==1 action FAIL with_rate la new_state N=0,ND=ND0

if N==0  action  REPAIR with_rate mu new_state N=1,ND=ND0

if ND>0 action FAIL_LEFT with_rate la*ND new_state ND--

if ND<ND0 action REPAIR_RIGHT with_rate mu new_state ND++


start_state

N=1,ND=ND0

reward_rate (double)(N==0)

initial_probability (double)(N==1 && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description
```

```
  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the EARR measure using Quasi-stationarity Detection for the
  particular case in which there are non-absorbing states with non-null
  reward rate.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_2_maple).

*/

parameters


int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(!F && N!=0)
```

```
initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the EARR measure using Bounding Regenerative Randomization
  Randomization with Quasi-stationarity Detection.

  ->Exact solution.

  It can be computed with Maple (command file: EARR_2_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0


   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu
```

```
   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. There have also been added several non-reachable states.

  -> CTMC description

  8 states; reducible: there is a transient, non-reachable component
  (the 4 states with DEL=yes) and a trapping, reachable component (the
  4 states with DEL=no)

  ->What tests is the example intended for

  Testing of the EARR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states in the particular case in which there are non-reachable states.

  ->Exact solution.

  It can be computed with Maple (command file: EARR_3_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la
                mateixa) */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
P0   /* state (DEL=no,N=2,F=no) has initial probability P0; state
        (DEL=no,N=1,F=no) has initial probability 1-P0 */

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules

/* adds a transition rate from state (DEL=yes, N, F), N>=1, F=no
 * to state (DEL=no, N, F) */
if DEL && N>=1 && !F action TO_REACH with_rate la*N new_state DEL=no
```

```
if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

DEL=yes, N=stat_start, F= no

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(DEL==no && N==2 && F==no)+(1-P0)*(DEL==no && N==1 && F==no))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description 8 states; reducible: there is a transient,
  non-reachable component (the 4 states with DEL=yes) and a trapping,
  reachable component (the 4 states with DEL=no)

  ->What tests is the example intended for

  Testing of the EARR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states.

  ->Exact solution.

  It can be computed with Maple (command file: EARR_1_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
```

```
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0


start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the EARR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states.

  ->Exact solution.

  It can be computed with Maple (command file: EARR_5_maple).
```

```
*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0,  /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */
RWD  /* reward rate */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate RWD*(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
```

```
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the EARR measure in the particular case in which all states have
  the same reward rate (5) using any of the numerical methods offered by
  METFAC2.1 that 1) do not require the CTMC to have absorbing states, and 2)
  consider as a particular case the fact that all states have the same reward
  rate regardless whether it is zero or not.

  ->Exact solution.

  EARR(t)=5 for all t>=0.

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
```

```
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate 5.0

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the EARR measure in the particular case in which all states have
  zero reward rate using any of the numerical methods offered by METFAC2.1
  that 1) do not require the CTMC to have absorbing states, and 2) consider as
  a particular case the fact that all states have zero reward rate.

  ->Exact solution.

  EARR(t)=0 for all t>=0.

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
```

```
   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate 0.0

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There are two transient components, one
  of which is not reachable, and one absorbing state.)


  ->What tests is the example intended for

  Testing of the EARR measure when all reward rates are equal to zero and
  there are non-reachable states using any of the methods offered by METFAC
  that do not require the chain to be irreducible.

  ->Exact solution.

  EARR(t)=0 for all t>=0.
*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
```

```
mu_d,
P0

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)0

initial_probability
    (double)(P0*(DEL==no && N==2 && ND==ND0)+(1-P0)*(DEL==no && N==1 && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and coverage
  probability C; the system can only be repaired if there is one unfailed unit
  and the failure of the second one was covered. Moreover, the state with an
  uncovered failure is "split" into several "fake" states in such a way that
  given the reward rate structure, the exact value of the measure can be
  computed regarding those states as lumped into one state. (i.e. the
  partition is lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the EARR measure using Regenerative Randomization and
  Regenerative Randomization with Laplace Transforms for the particular case
  in which the model is stiff.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_10_maple).

*/

parameters
```

```
int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0



state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if F && ND>0 action with_rate la_d new_state ND--
if F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system: components fail with rate la and are
  repaired by a single repair person with rate mu; coverage
  probability is C.

  ->CTMC description

  Eight states; states with DEL=yes are non-reachable and after
  deleting them, the resulting CTMC is irreducible.

  ->What tests is the example intended for
```

```
  Testing of the ECRDS measure (subset: up states) using any of the
  numerical methods offered by METFAC2.1.

  ->Exact solution.

  ECRDS = (2*la+mu+la*(2*C-1)*(1-C)/(1-C+la*C/mu))/(2*la*(la+mu*(1-C)))
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */


state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules

/* adds a transition rate from state (DEL=yes, N, F), N>=1, F=no
 * to state (DEL=no, N, F) */
if DEL && N>=1 && !F action TO_REACH with_rate la*N new_state DEL=no

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

DEL=yes, N=2, F= no

reward_rate (double)(N>=1 && !F)


initial_probability (double) (DEL==no && N==2 && F==no)
/*
  ->Example description

  System consisting of N0 identical units that fail with rate la and
  coverage probability C and are repaired by a single repair person
  with rate mu. The unique state with all components failed is "split"
  into several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description
```

```
  N0+ND0+1 states; irreducible.

  ->What tests is the example intended for

  Testing of the ECRDS measure (subset: up states) using any of the
  numerical methods offered by METFAC2.1 to solve singular linear
  systems of equations with the particularity that keeping N0
  reasonably small (say 2) the exact value of the measure is very easy
  to compute while the CTMC can be made arbitrarily large by taking
  large values for ND0.

  ->Exact solution.

  N0==2: ECRDS = (2*la+mu+la*(2*C-1)*(1-C)/(1-C+la*C/mu))/(2*la*(la+mu*(1-C)))
*/

parameters

int
N0,    /* no. inicial de repliques */
ND0,   /* l'estat amb N=0 i F=no s'"explosiona" en ND0+1 estats,
          ND0>=0 */

double
la,     /* taxa fallada d'una replica */
mu,    /* taxa reparacio d'una replica */
C,     /* factor cobertura */
la_d   /* taxa de transicio entre un estat (N=0,F=no,ND), ND>=1 i
        * l'estat (N=0,F=no,ND-1) */

state_variables
N,            /* no. de repliques ok */
F,     /* yes si sistema fallada de cobertura i no altrament */
ND     /* val sempre 0 per als estats amb N>0 o F=yes; per als
          * estats amb N=0 i F=no, aquesta variable pren els
          * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
  if N==1 response FAIL new_state N--

  if N>1 response COVERED with_prob C new_state N--
  if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes
end

if N<N0 && ND==ND0 action REPAIR with_rate mu
  if F response REPAIR new_state N++, F=no
  if !F response REPAIR new_state N++
end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state
N=N0, F=no, ND=ND0

reward_rate (double)(N>=1 && !F)
/*
  ->Example description

  Fictitious system.

  ->CTMC description

  The CTMC has 2 transient components and 2 trapping ones. The first
  transient component consists of states (S,N) with S={1,2},
```

N={0,...,N0}; the second transient component consists of states (S,N), with S={3,4}, N={0,...,N0}; the first trapping component consists of states (S,N), with S={5,6}, N={0,...,N0}; and the second trapping component consists of states (S,N), with S={7,8}, N={0,...,N0}. The transition rates are as follows:

ld_a from (1,N) to (2,N0)

mu_a from (2,N) to (1,N0)

ld_lr from (2,N) to (3,N0)

ld_udl from (2,N) to (5,N0)

ld_b from (3,N) to (4,N0)

mu_b from (4,N) to (3,N0)

ld_udr from (4,N) to (7,N0)

ld_c from (5,N) to (6,N0)

mu_c from (6,N) to (5,N0)

ld_d from (7,N) to (8,N0)

mu_d from (8,N) to (7,N0)

ld from (S,N) to (S,N-1), S={1,2,3,4,5,6,7,8}, N>0

mu from (S,N) to (S,N+1), S={1,2,3,4,5,6,7,8}, N<N0

State (1,N0) has initial probability P0, state (3,N0) has initial probability 2*(1-P0)/5, state (5,N0) has initial probability 2*(1-P0)/5, state (7,N0) has initial probability (1-P0)/5, and the remaining states have 0 initial probability.

States (5,N) and (7,N) have reward rate equal to 1; the remaining states have reward rate equal to 0.

->What tests is the example intended for

Testing of the ECRDS measure (subset: S={5,7}) using any of the numerical methods offered by METFAC2.1 that do not require the CTMC to be irreducible. The CTMC can be made arbitrarily large by taking large values for the N0 parameter.

->Exact solution

States (S,N), 0<=N<=N0 can be lumped into a single state and, therefore, the exact value of the ECRDS measure can be computed considering a CTMC with eight states S=1, 2, ..., 8, reward rate equal to 1 for states S=5 and S=7, and 0 for the remaining states, initial probability P0 for state S=1, 2*(1-P0)/5 for state S=3, 2*(1-P0)/5 for state S=5, (1-P0)/5 for state S=7, and 0 for the remaining states, and the following transition rates:

ld_a from S=1 to S=2

mu_a from S=2 to S=1

ld_lr from S=2 to S=3

ld_udl from S=2 to S=5

ld_b from S=3 to S=4

mu_b from S=4 to S=3

```
  ld_c from S=5 to S=6

  mu_c from S=6 to S=5

  ld_udr from S=4 to S=7

  ld_d from S=7 to S=8

  mu_d from S=8 to S=8

  The exact value of the measure can be computed with the aid of
  the Maple program. (Command file: ECRDS_3_maple.)

*/

parameters

int
N0,

double
P0,
ld_a,
mu_a,
ld_b,
mu_b,
ld_c,
mu_c,
ld_d,
mu_d,
ld_lr,
ld_udl,
ld_udr,
ld,
mu


state_variables
S,N


production_rules

if S==1 action S_1_TO_S_2 with_rate ld_a new_state S=2, N=N0
if S==2 action S_2_TO_S_1 with_rate mu_a new_state S=1, N=N0
if S==2 action S_2_TO_S_3 with_rate ld_lr new_state S=3, N=N0
if S==2 action S_2_TO_S_5 with_rate ld_udl new_state S=5, N=N0
if S==3 action S_3_TO_S_4 with_rate ld_b new_state S=4, N=N0
if S==4 action S_4_TO_S_3 with_rate mu_b new_state S=3, N=N0
if S==4 action S_4_TO_S_7 with_rate ld_udr new_state S=7, N=N0
if S==5 action S_5_TO_S_6 with_rate ld_c new_state S=6, N=N0
if S==6 action S_6_TO_S_5 with_rate mu_c new_state S=5, N=N0
if S==7 action S_7_TO_S_8 with_rate ld_d new_state S=8, N=N0
if S==8 action S_8_TO_S_7 with_rate mu_d new_state S=7, N=N0

if N>0 action RIGTH with_rate ld new_state N--
if N<N0 action LEFT with_rate mu new_state N++


start_state
S=1, N=N0

reward_rate (double)(S==5 || S==7)

initial_probability
(double)
(P0*(S==1 && N==N0)
```

```
+(2*(1.0-P0)/5)*(S==3 && N==N0)
+(2*(1.0-P0)/5)*(S==5 && N==N0)
+((1.0-P0)/5)*(S==7 && N==N0))
/*
  ->Example description

  Dual repairable system: components fail with rate la and are
  repaired by a single repair person with rate mu; coverage
  probability is C.

  ->CTMC description

  Four states; irreducible.

  ->What tests is the example intended for

  Testing of the ECRDS measure for the particular case in which the
  subset includes just one state (the state with no failed
  components).

  ->Exact solution.

  ECRDS = 1/(2*la)
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */


state_variables

N,   /* numero de replicas en marcha */
F    /* fallo del sistema yes/no */

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

N=2, F= no

reward_rate (double)(N>=1 && !F)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
```

```
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the ECRTE measure when there are non-reachable states
  using any of the methods offered by METFAC.

  ->Exact solution.

  ECRTE = ((la+mu)/(2*la)+C)/(la+mu*(1-C))

*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0
```

```
reward_rate (double)(N>0)

initial_probability (double)(DEL==no && N==2 && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the ECRTE measure when there are non-reachable states
  using any of the methods offered by METFAC. The example allows to
  test a particular case of the "accelerated" Gauss-Seidel and
  Optimized SOR methods, namely when there are two pivot states and
  neither of them has initial probability equal to one.

  ->Exact solution.

  It can be computed with Maple. (Command file: ECRTE_2_maple.)

*/

parameters

int
ND0,

double
P0,  /* probabilitat inicial de l'estat (DEL=no,N=2,ND=ND0) es P0;
      * probabilitat inicial de l'estat (DEL=no,N=1,ND=ND0) es 1-P0 */
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0
```

```
   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability
    P0*(DEL==no && N==2 && ND==ND0)+(1.0-P0)*(DEL==no && N==1 && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the ECRTE measure when there are non-reachable states
  using any of the methods offered by METFAC. The example allows to
  test a particular case of the "accelerated" Gauss-Seidel and
  Optimized SOR methods, namely when there is one pivot state and
  its initial probability is not one.

  ->Exact solution.

  It can be computed with Maple. (Command file: ECRTE_3_maple.)

*/

parameters

int
ND0,

double
P0,  /* probabilitat inicial de l'estat (DEL=no,N=2,ND=ND0) es P0;
      * probabilitat inicial de l'estat (DEL=no,N=1,ND=ND0) es 1-P0 */
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
```

```
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability
    P0*(DEL==no && N==2 && ND==ND0)+(1.0-P0)*(DEL==no && N==1 && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There is a transient component and one
  absorbing state.)

  ->What tests is the example intended for

  Testing of the ECRTE measure when there are non-reachable states
  using any of the methods offered by METFAC. The example allows to
  test a particular case of the "accelerated" Gauss-Seidel and
  Optimized SOR methods, namely when there is one pivot state and
  its initial probability is equal to one.

  ->Exact solution.

  It can be computed with Maple. (Command file: ECRTE_4_maple.)

*/

parameters

int
ND0,
```

```
double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end

if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--

if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability
    (double) (DEL==no && N==2 && ND==ND0)
/*
  ->Example description

  CTMC to compute the MTTF of a dual repairable system in which units fail
  with rate la, failures are covered with probability C, and failed units are
  repaired by single repair person with rate mu.

  ->CTMC description

  3 states, one of which is absorbing.

  ->What tests is the example intended for

  Testing of the ECRTE measure using any of the methods offered by METFAC.

  ->Exact solution.

  ECRTE = ((la+mu)/(2*la)+C)/(la+mu*(1-C))

*/

parameters

double
```

```
la,  /* failure rate */
mu,  /* repair rate */
C    /* coverage probability */

state_variables

N   /* number of unfailed units */

production_rules

if N>0 action FAILURE with_rate N*la
  if N>1 response COVERED with_prob C
  new_state N--
  if N>1 response UNCOVERED with_prob 1-C
  new_state N=0
  if N==1 response FAILURE
  new_state N--
end

if N>0 && N<2 action REPAIR with_rate mu
new_state N++

start_state

N=2

reward_rate (double)(N>0)

initial_probability (double)(N==2)
/*
  ->Example description

  "Lumpable" version of ECRTE_5: each state "N=i" is split into
  ND0+1 states "N=i,ND=j", 0\leq j\leq ND0."

  ->CTMC description

  3*(ND0+1) states (a transient strongly connected component plus an
  absorbing state).

  ->What tests is the example intended for

  Testing of the ECRTE measure using any of the methods offered by METFAC.

  ->Exact solution.

  ECRTE = ((la+mu)/(2*la)+C)/(la+mu*(1-C))

*/

parameters

int
ND0,

double
la,  /* failure rate */
mu,  /* repair rate */
C,   /* coverage probability */
la_d,/* "dummy" failure rate */
mu_d /* "dummy" repair rate */

state_variables

N,  /* number of unfailed units */
ND

production_rules
```

```
if N>0 action FAILURE with_rate N*la
  if N>1 response COVERED with_prob C
  new_state N--, ND=ND0
  if N>1 response UNCOVERED with_prob 1-C
  new_state N=0, ND=ND0
  if N==1 response FAILURE
  new_state N--, ND=ND0
end

if N>0 && N<2 action REPAIR with_rate mu
new_state N++, ND=ND0

if N !=0 && ND>0 action DUMMY_FAILURE with_rate ND*la_d
new_state ND--

if N != 0 && ND<ND0 action DUMMY_REPAIR with_rate mu_d
new_state ND++

start_state

N=2, ND=ND0

reward_rate (double)(N>0)

initial_probability (double)(N==2 && ND==ND0)
/*
  ->Example description

  Dual repairable system: components fail with rate la and are
  repaired by a single repair person with rate mu; coverage
  probability is C.

  ->CTMC description

  Four states; irreducible.

  ->What tests is the example intended for

  Testing of the ESSRR measure using any of the numerical methods
  offered by METFAC2.1 to solve singular linear systems of equations.

  ->Exact solution.

  ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */


state_variables

N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules


if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--
```

```
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu


   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

N=2, F= no

reward_rate (double)(F || N==0)
/*
  ->Example description

  System consisting of N0 identical units that fail with rate la and
  coverage probability C and are repaired by a single repair person
  with rate mu. The unique state with all components unfailed is "split"
  into several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  N0+ND0+1 states; irreducible.

  ->What tests is the example intended for

  Testing of the ESSRR measure using Block Gauss Seidel in the particular case
  in which: 1) there are more than two blocks, and 2) for some of the blocks
  and for some of the states of the block, say state s1, there exist
  transition rates s1->s2 and s1->s3, where states s2 and s3 belong each to a
  different block which is also different from the one which state s1 belongs
  to. (Take N0=2 and ND0=30.)

  ->Exact solution.

  N0==2: ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)
*/

parameters

int
N0,    /* no. inicial de repliques */
ND0,   /* l'estat amb N=0 i F=no s'"explosiona" en ND0+1 estats,
          ND0>=0 */

double
la,     /* taxa fallada d'una replica */
mu,    /* taxa reparacio d'una replica */
C,     /* factor cobertura */
la_d   /* taxa de transicio entre un estat (N=0,F=no,ND), ND>=1 i
        * l'estat (N=0,F=no,ND-1) */

state_variables
N,            /* no. de repliques ok */
F,     /* yes si sistema fallada de cobertura i no altrament */
ND     /* val sempre 0 per als estats amb N>0 o F=yes; per als
          * estats amb N=0 i F=no, aquesta variable pren els
          * valors ND0, ND0-1,..., 0 */
```

```
production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
  if N==1 response FAIL new_state N--
  if N>1 response COVERED with_prob C new_state N--
  if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes
end

if N==N0 && !F && ND>0 action MOVE_UP with_rate ND0*la_d
new_state ND--

if N==N0 && !F && ND<ND0 action MOVE_DOWN with_rate mu
new_state ND++

if N==N0 && !F && ND!=ND0 action MOVE_RIGHT with_rate la*N
  if N==1 response FAIL new_state N--
  if N>1 response COVERED with_prob C new_state N--, ND=ND0
  if N>1 response UNCOVERED with_prob 1-C new_state N--, ND=ND0, F=yes
end

if N<N0 && ND==ND0 action REPAIR with_rate mu
  if F response REPAIR new_state N++, F=no
  if !F response REPAIR new_state N++
end

start_state
N=N0, F=no, ND=ND0

reward_rate (double)(F || N==0)
/*
  ->Example description

  M/M/1/C queue with arrival and service rate LAMBDA and MU, respectively.
  The measure of interest is the expected waiting time, E(T).

  ->CTMC description

  C+1 states; irreducible.

  ->What tests is the example intended for.

  Computing of the ESSRR measure using the SOR method trying to enter as many
  as possible states of the automaton for the dynamincal tuning of the
  overrelaxation factor.

  ->Exact solution.

  E(T)=E(N)/gamma,
  E(N)=\sum_{n=0}^N p(n)*n, p(n)=(1-rho)*rho^n/(1-rho^{n+1}), rho=LAMBDA/MU,
  gamma=LAMBDA*(1-P_loss),
  P_loss = p(C).
*/

parameters

int
C,      /* queue length */

double
LAMBDA, /* arrival rate */
MU      /* service rate */


state_variables
N       /* no. of jobs in the queue including the one being served */
```

```
external

double reward(int, int, double, double)


production_rules

/* job arrival */
if N<C action ARRIVAL with_rate LAMBDA
new_state N++

/* job departure */
if N>0 action DEPARTURE with_rate MU
new_state N--


start_state

N=0


reward_rate reward(C, N, LAMBDA, MU)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the ESSRR measure in the particular case in which all states have
  the same reward rate (5) using the method based on stationarity detection.

  ->Exact solution.

  ESSRR=5

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
```

```
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate 5.0

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and coverage
  probability C; the system can only be repaired if there is one unfailed unit
  and the failure of the second one was covered. Moreover, the state with an
  uncovered failure is "split" into several "fake" states in such a way that
  given the reward rate structure, the exact value of the measure can be
  computed regarding those states as lumped into one state. (i.e. the
  partition is lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using Regenerative Randomization with Laplace
  Transforms in the particular case in which there are absorbing states, the
initial
  probability of the regenerative state is not 1 and either L>K or L=K.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_9_maple).

*/

parameters
```

```
int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0



state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if F && ND>0 action with_rate la_d new_state ND--
if F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  System consisting of N0 identical units that fail with rate la and
  coverage probability C and are repaired by a single repair person
  with rate mu. The unique state with all components failed is "split"
  into several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  N0+ND0+1 states; irreducible.
```

```
  ->What tests is the example intended for

  Testing of the ESSRR measure using any of the numerical methods
  offered by METFAC2.1 to solve singular linear systems of equations
  with the particularity that keeping N0 reasonably small (say 2) the
  exact value of the measure is very easy to compute while the CTMC
  can be made arbitrarily large by taking large values for ND0. In
  this way the methods can be "stressed". Moreover, the example allows
  to test two particular cases of the numerical method Block Gauss
  Seidel: a) at least one block is of size <=30 and therefore the
  corresponding system of equations is solved using a direct method
  (take ND0=10), and b) at least one block is of size >30 and
  therefore the corresponding system of equations is solved using an
  iterative method (take ND0=31).

  ->Exact solution.

  N0==2: ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)
*/

parameters

int
N0,    /* no. inicial de repliques */
ND0,   /* l'estat amb N=0 i F=no s'"explosiona" en ND0+1 estats,
          ND0>=0 */

double
la,     /* taxa fallada d'una replica */
mu,    /* taxa reparacio d'una replica */
C,     /* factor cobertura */
la_d   /* taxa de transicio entre un estat (N=0,F=no,ND), ND>=1 i
        * l'estat (N=0,F=no,ND-1) */

state_variables
N,            /* no. de repliques ok */
F,     /* yes si sistema fallada de cobertura i no altrament */
ND     /* val sempre 0 per als estats amb N>0 o F=yes; per als
          * estats amb N=0 i F=no, aquesta variable pren els
          * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
  if N==1 response FAIL new_state N--
  if N>1 response COVERED with_prob C new_state N--
  if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes
end

if N<N0 && ND==ND0 action REPAIR with_rate mu
  if F response REPAIR new_state N++, F=no
  if !F response REPAIR new_state N++
end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state
N=N0, F=no, ND=ND0

reward_rate (double)(F || N==0)



/*
  ->Example description
```

```
Fictitious system.

->CTMC description

The CTMC has 2 transient components and 2 trapping ones. The first
transient component consists of states (S,N) with S={1,2},
N={0,...,N0}; the second transient component consists of states
(S,N), with S={3,4}, N={0,...,N0}; the first trapping component
consists of states (S,N), with S={5,6}, N={0,...,N0}; and the second
trapping component consists of states (S,N), with S={7,8},
N={0,...,N0}. The transition rates are as follows:

ld_a from (1,N) to (2,N0)

mu_a from (2,N) to (1,N0)

ld_lr from (2,N) to (3,N0)

ld_udl from (2,N) to (5,N0)

ld_b from (3,N) to (4,N0)

mu_b from (4,N) to (3,N0)

ld_udr from (4,N) to (7,N0)

ld_c from (5,N) to (6,N0)

mu_c from (6,N) to (5,N0)

ld_d from (7,N) to (8,N0)

mu_d from (8,N) to (7,N0)

ld from (S,N) to (S,N-1), S={1,2,3,4,5,6,7,8}, N>0

mu from (S,N) to (S,N+1), S={1,2,3,4,5,6,7,8}, N<N0

State (1,N0) has initial probability P0, state (3,N0) has initial
probability 1-P0, and the remaining states have 0 initial
probability.

States (6,N) and (8,N) have reward rate equal to 1; the remaining
states have reward rate equal to 0.

->What tests is the example intended for

Testing of the ESSRR measure using any of the numerical methods
offered by METFAC2.1 to solve singular and nonsingular (there are
two trapping components) linear systems of equations for an
arbitrarily large CTMC. (The larger N0, the more states the CTMC has.)

->Exact solution

States (S,N), 0<=N<=N0 can be lumped into a single state and,
therefore, the exact value of the ESSRR measure can be computed
considering a CTMC with eight states S=1, 2, ..., 8, reward rate
equal to 1 for states S=5 and S=7 and to 0 for the remaining states,
initial probability P0 for states S=1, 1-P0 for state S=3, and 0 for
the remaining states, and the following transition rates:

ld_a from S=1 to S=2

mu_a from S=2 to S=1

ld_lr from S=2 to S=3
```

```
  ld_udl from S=2 to S=5

  ld_b from S=3 to S=4

  mu_b from S=4 to S=3

  ld_c from S=5 to S=6

  mu_c from S=6 to S=5

  ld_udr from S=4 to S=7

  ld_d from S=7 to S=8

  mu_d from S=8 to S=8

  Therefore,

  ESSRR = P0*ld_udl*ld_c/((ld_lr+ld_udl)*(ld_c+mu_c))
         +[(1-P0)/ld_udr+ld_lr*P0/((ld_lr+ld_udl)*ld_udr)]
           *ld_udr*ld_d/(ld_d+mu_d)
*/

parameters

int
N0,

double
P0,
ld_a,
mu_a,
ld_b,
mu_b,
ld_c,
mu_c,
ld_d,
mu_d,
ld_lr,
ld_udl,
ld_udr,
ld,
mu


state_variables
S,N


production_rules

if S==1 action S_1_TO_S_2 with_rate ld_a new_state S=2, N=N0
if S==2 action S_2_TO_S_1 with_rate mu_a new_state S=1, N=N0
if S==2 action S_2_TO_S_3 with_rate ld_lr new_state S=3, N=N0
if S==2 action S_2_TO_S_5 with_rate ld_udl new_state S=5, N=N0
if S==3 action S_3_TO_S_4 with_rate ld_b new_state S=4, N=N0
if S==4 action S_4_TO_S_3 with_rate mu_b new_state S=3, N=N0
if S==4 action S_4_TO_S_7 with_rate ld_udr new_state S=7, N=N0
if S==5 action S_5_TO_S_6 with_rate ld_c new_state S=6, N=N0
if S==6 action S_6_TO_S_5 with_rate mu_c new_state S=5, N=N0
if S==7 action S_7_TO_S_8 with_rate ld_d new_state S=8, N=N0
if S==8 action S_8_TO_S_7 with_rate mu_d new_state S=7, N=N0

if N>0 action RIGTH with_rate ld new_state N--
if N<N0 action LEFT with_rate mu new_state N++


start_state
```

```
S=1, N=N0

reward_rate (double)(S==6 || S==8)

initial_probability (double) (P0*(S==1 && N==N0)+(1.0-P0)*(S==3 && N==N0))
/*
  ->Example description

  Fictitious system.

  ->CTMC description

  Five states S=1, 2, ..., 5. Three components: S={1,2} (transient),
  S={3} (trapping), and S={4,5} (trapping). State S=i has reward
  rate RW_i and initial probability P0_i. Transition rates are as
  follows:

  from S=1 to S=2    LD

  from S=4 to S=5    LD

  from S=2 to S=1    MU

  from S=5 to S=4    MU

  from S=1 to S=3    LDP

  from S=2 to S=4    LDP

  ->What tests is the example intended for

  Computation of the ESSRR measure for the particular case in which there
  is one transient component with just two states (i.e. the corresponding
  nonsingular linear system of equations is solved using the Cramer's rule),
  one trapping component with one state (i.e. it is straightforward to
  solve the corresponding singular linear system of equations), and one
  trapping component with just two states (i.e. the associated singular
  linear system of equations is solved using the Cramer's rule).

  ->Exact solution.

  If p0[i] and rw[i] stand for the initial probability and reward rate
  of state i, then

         ESSRR = (LDP*tau_1+p0[3])*rw[3]
                +(LDP*tau_2+p0[4]+p0[5])*MU*rw[4]/(LD+MU)
                +(LDP*tau_2+p0[4]+p0[5])*LD*rw[5]/(LD+MU)

  where

         tau_2 = (p0[1]*LD+p0[2]*(LD+LDP))/(MU*LDP+LDP*LD+LDP*LDP)

  and

         tau_1 = (p0[1]+MU*tau[2])/(LD+LDP).

*/

parameters

    double LD,      /* transition rate S=1->S=2 and S=4->S=5 */
           LDP,     /* transition rate S=1->S=3 and S=2->S=4 */
           MU,      /* transition rate S=2->S=1 and S=5->S=4 */
           P0_1,    /* initial probability of state S=1 */
           P0_2,    /* initial probability of state S=2 */
           P0_3,    /* initial probability of state S=3 */
           P0_4,    /* initial probability of state S=4 */
           P0_5,    /* initial probability of state S=5 */
```

```
            RW_1,    /* reward rate of state S=1 */
            RW_2,    /* reward rate of state S=2 */
            RW_3,    /* reward rate of state S=3 */
            RW_4,    /* reward rate of state S=4 */
            RW_5     /* reward rate of state S=5 */


state_variables
            S


production_rules

if S==1 || S==4 action with_rate LD new_state S++
if S==2 || S==5 action with_rate MU new_state S--
if S==1 || S==2 action with_rate LDP
  if S==1 response new_state S=3
  if S==2 response new_state S=4
end


start_state

S=1


reward_rate
    (double)((S==1)*RW_1+(S==2)*RW_2+(S==3)*RW_3+(S==4)*RW_4+(S==5)*RW_5)


initial_probability
    (double)((S==1)*P0_1+(S==2)*P0_2+(S==3)*P0_3+(S==4)*P0_4+(S==5)*P0_5)
/*
  ->Example description

  Fictitious system.

  ->CTMC description

  Five states S=1, 2, ..., 5. Four components: S={1} (transient),
  S={2} (transient), S={3} (trapping), and S={4,5} (trapping). State
  S=i has reward rate RW_i and initial probability P0_i. Transition
  rates are as follows:

  from S=1 to S=2    LD

  from S=4 to S=5    LD

  from S=5 to S=4    MU

  from S=1 to S=3    LDP

  from S=2 to S=4    LDP

  ->What tests is the example intended for

  Computation of the ESSRR measure for the particular case in which
  there are two transient components with one state each (i.e. it is
  straightforward to solve the corresponding nonsingular linear
  systems of equations), one trapping component with one state
  (i.e. it is straightforward to solve the corresponding singular
  linear system of equations), and one trapping component with just
  two states (i.e. the associated singular linear system of equations
  is solved using the Cramer's rule).

  ->Exact solution.

  If p0[i] and rw[i] stand for the initial probability and reward rate
```

```
  of state i, then

         ESSRR = (LDP*tau_1+p0[3])*rw[3]
                +(LDP*tau_2+p0[4]+p0[5])*MU*rw[4]/(LD+MU)
                +(LDP*tau_2+p0[4]+p0[5])*LD*rw[5]/(LD+MU)

  where

         tau_1 = p0[1]/(LD+LDP),

         tau_2 = LD*p0[1]/(LDP*(LD+LDP))+p0[2]/LDP.

*/

parameters

    double LD,      /* transition rate S=1->S=2 and S=4->S=5 */
           LDP,     /* transition rate S=1->S=3 and S=2->S=4 */
           MU,      /* transition rate S=2->S=1 and S=5->S=4 */
           P0_1,    /* initial probability of state S=1 */
           P0_2,    /* initial probability of state S=2 */
           P0_3,    /* initial probability of state S=3 */
           P0_4,    /* initial probability of state S=4 */
           P0_5,    /* initial probability of state S=5 */
           RW_1,    /* reward rate of state S=1 */
           RW_2,    /* reward rate of state S=2 */
           RW_3,    /* reward rate of state S=3 */
           RW_4,    /* reward rate of state S=4 */
           RW_5     /* reward rate of state S=5 */


state_variables
           S


production_rules

if S==1 || S==4 action with_rate LD new_state S++
if S==5 action with_rate MU new_state S--
if S==1 || S==2 action with_rate LDP
  if S==1 response new_state S=3
  if S==2 response new_state S=4
end


start_state

S=1


reward_rate
    (double)((S==1)*RW_1+(S==2)*RW_2+(S==3)*RW_3+(S==4)*RW_4+(S==5)*RW_5)


initial_probability
    (double)((S==1)*P0_1+(S==2)*P0_2+(S==3)*P0_3+(S==4)*P0_4+(S==5)*P0_5)
/*
  ->Example description

  System consisting of N0 identical units that fail with rate la and
  coverage probability C and are repaired by a single repair person
  with rate mu. The unique state with all components failed is "split"
  into several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description
```

```
  N0+ND0+1 states; irreducible.

  ->What tests is the example intended for

  Testing of the ESSRR measure using any of the numerical methods
  offered by METFAC2.1 to solve singular linear systems of equations
  with the particularity that keeping N0 reasonably small (say 2) the
  exact value of the measure is very easy to compute while the CTMC
  can be made arbitrarily large by taking large values for ND0. In
  this way the example allows to "trigger" some internal functions in
  charge of enlarging the data structures that hold the CTMC.

  ->Exact solution.

  N0==2: ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)
*/

parameters

int
N0,    /* no. inicial de repliques */
ND0,   /* l'estat amb N=0 i F=no s'"explosiona" en ND0+1 estats,
          ND0>=0 */

double
la,     /* taxa fallada d'una replica */
mu,    /* taxa reparacio d'una replica */
C,     /* factor cobertura */
la_d   /* taxa de transicio entre un estat (N=0,F=no,ND), ND>=1 i
        * l'estat (N=0,F=no,ND-1) */

state_variables
N,            /* no. de repliques ok */
F,     /* yes si sistema fallada de cobertura i no altrament */
ND     /* val sempre 0 per als estats amb N>0 o F=yes; per als
          * estats amb N=0 i F=no, aquesta variable pren els
          * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
  if N==1 response FAIL new_state N--
  if N>1 response COVERED with_prob C new_state N--
  if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes
end

if N<N0 && ND==ND0 action REPAIR with_rate mu
  if F response REPAIR new_state N++, F=no
  if !F response REPAIR new_state N++
end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state
N=N0, F=no, ND=ND0

reward_rate (double)(F || N==0)
/*
  ->Example description

  Dual repairable system: components fail with rate la and are
  repaired by a single repair person with rate mu; coverage
  probability is C.

  ->CTMC description
```

```
  Four states; irreducible.

  ->What tests is the example intended for

  Testing of the ESSRR measure using any of the numerical methods
  offered by METFAC2.1 to solve singular linear systems of equations.
  By defining a suitable block() function, the example allows to test
  the Block Gauss-Seidel method when at least one of the blocks
  consists of just one state.

  ->Exact solution.

  ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)
*/

parameters

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C    /* factor de cobertura */


state_variables

N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--


   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++


end

start_state

N=2, F= no

reward_rate (double)(F || N==0)
/*
  ->Example description

  Dual repairable system: components fail with rate la and are
  repaired by a single repair person with rate mu; coverage
  probability is C.

  ->CTMC description

  Four states; irreducible.

  ->What tests is the example intended for
```

```
  Testing of the ESSRR measure using any of the numerical methods
  offered by METFAC2.1 to solve singular linear systems of equations
  for the particular case in which the specification does not use
  parameters.

  ->Exact solution.

  ESSRR = 2*1e-4*(1e-4*0.999+1*(1-0.999))/(1*1+2*1e-4*1e-4*0.999+2*1e-4*1)
*/

state_variables

N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules

if N>=1 && !F action FAILURE with_rate 1e-4*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob 0.999 new_state N--

   if N>1 response UNCOVERED with_prob 1-0.999 new_state N--, F=yes

end

if N<2 action REPAIR with_rate 1.0

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

N=2, F= no

reward_rate (double)(F || N==0)
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+ND0 states; irreducible.

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_1_maple).

*/
```

```
parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and coverage
  probability C; the system can only be repaired if there is one unfailed unit
  and the failure of the second one was covered. Moreover, the state with an
  uncovered failure is "split" into several "fake" states in such a way that
  given the reward rate structure, the exact value of the measure can be
  computed regarding those states as lumped into one state. (i.e. the
  partition is lumpable.)
```

```
  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using Bounding Regenerative Randomization for the
  particular case in which the model is stiff.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_10_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0



state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0


end

if F && ND>0 action with_rate la_d new_state ND--
if F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0
```

```
reward_rate (double)(N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using any numerical method that requires or
  allows the chain to have absorbing states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_11_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0



state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end
```

```
if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate 1e8*(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Non-reparirable fault-tolerant system that uses the triple modular
  redundancy technique. The system is made up of 3 identical units that that
  compare their results. When one unit fails, it is switched off and the
  system degrades to a dual configuration. In that configuration, the failure
  of a unit is detected with probability C, and is not, and therefore the
  system fails, with probability 1-C. When only one unit is operational, its
  failure makes the system fail.

  ->CTMC description

  4 states, one of which is absorbing; each state makes up a strongly
  connected component.

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the methods offered by METFAC.

  ->Exact solution.

  ETRR(t)=1-3*C*exp(-la*t)+(3-6(1-C))*exp(-2*la*t)-(1-3(1-C))*exp(-3*la*t)

*/

parameters

double
la,  /* failure rate */
C    /* coverage probability */

state_variables

N   /* number of unfailed units */

production_rules

if N>0 action FAILURE with_rate N*la
  if N==3 || N==1 response
  new_state N--
  if N==2 response COVERED with_prob C
  new_state N--
  if N==2 response UNCOVERED with_prob 1-C
  new_state N=0
end

start_state
```

```
N=3

reward_rate (double) (N==0)

initial_probability (double) (N==3)
/*
  ->Example description

  "Lumpable" version of ETRR_12: each state "N=i" is split into
  ND0+1 states "N=i,ND=j", 0\leq j\leq ND0."

  ->CTMC description

  4*(ND0+1) states, ND0+1 of which are absorbing; the chain has 4*(ND0+1)
  strongly connected components

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the methods offered by METFAC.

  ->Exact solution.

  ETRR(t)=1-3*C*exp(-la*t)+(3-6(1-C))*exp(-2*la*t)-(1-3(1-C))*exp(-3*la*t)

*/

parameters

int
ND0, /* each state "N=i" is split into ND0+1 states "N=i, ND=j", 0<=j<=ND0 */

double
la,  /* failure rate */
C,   /* coverage probability */
la_d_1,
la_d_2,
la_d_3,
la_d_4  /* "dummy" failure rates */

state_variables

N,   /* number of unfailed units */
ND

production_rules

if N>0 action FAILURE with_rate N*la
  if N==3 || N==1 response
  new_state N--, ND=ND0
  if N==2 response COVERED with_prob C
  new_state N--, ND=ND0
  if N==2 response UNCOVERED with_prob 1-C
  new_state N=0, ND=ND0
end

if ND>0 action DUMMY_FAILURE with_rate (double)ND
  if N==3 response DUMMY with_prob la_d_1
  new_state ND--

  if N==2 response DUMMY with_prob la_d_2
  new_state ND--

  if N==1 response DUMMY with_prob la_d_3
  new_state ND--

  if N==0 response DUMMY with_prob la_d_4
  new_state ND--
```

```
end

start_state

N=3, ND=ND0

reward_rate (double) (N==0)

initial_probability (double) (N==3 && ND==ND0)
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using Quasi-stationarity Detection for the
  particular case in which there are non-absorbing states with non-null
  reward rate.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_2_maple).

*/

parameters


int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0
```

```
   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(!F && N!=0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using Bounding Regenerative Randomization
  Randomization and Quasi-stationarity Detection.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_2_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
```

```
P0

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0

   if N>1 response COVERED with_prob C new_state N--, ND=ND0

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no, ND=ND0

   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. There have also been added several non-reachable states.

  -> CTMC description

  8 states; reducible: there is a transient, non-reachable component
  (the 4 states with DEL=yes) and a trapping, reachable component (the
  4 states with DEL=no)

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states in the particular case in which there are non-reachable states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_3_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
```

```
                a generar es N=stat_start, F=no (la cadena sera la
                mateixa) */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
P0   /* state (DEL=no,N=2,F=no) has initial probability P0; state
        (DEL=no,N=1,F=no) has initial probability 1-P0 */

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
F   /* fallo del sistema yes/no */

production_rules

/* adds a transition rate from state (DEL=yes, N, F), N>=1, F=no
 * to state (DEL=no, N, F) */
if DEL && N>=1 && !F action TO_REACH with_rate la*N new_state DEL=no

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--

   if N>1 response COVERED with_prob C new_state N--

   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 action REPAIR with_rate mu

   if F response REPAIR new_state N++, F=no

   if !F response REPAIR new_state N++

end

start_state

DEL=yes, N=stat_start, F= no

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(DEL==no && N==2 && F==no)+(1-P0)*(DEL==no && N==1 && F==no))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
```

```
  states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_4_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
   if N==1 response FAIL new_state N--
   if N>1 response COVERED with_prob C new_state N--
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu
   if F response REPAIR new_state N++, F=no
   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
```

```
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the ETRR measure using any of the numerical methods
  offered by METFAC2.1 that do not require the CTMC to have absorbing
  states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_5_maple).

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--
   if N>1 response COVERED with_prob C new_state N--
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu
   if F response REPAIR new_state N++, F=no
   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0
```

```
reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was covered. The
  state with no failed units and the state with an unfailed unit in
  which repair is allowed are "split" into several "fake" states in
  such a way that given the reward rate structure, the exact value of
  the measure can be computed regarding those states as lumped into
  one state. (i.e. the partition is lumpable.) Moreover, there are
  some non-reachable states.

  ->CTMC description

  4*ND0+5 states; reducible. (There are two transient components, one
  of which is not reachable, and one absorbing state.)


  ->What tests is the example intended for

  Testing of the ETRR measure when all reward rates are equal to zero and
  there are non-reachable states using any of the methods offered by METFAC
  that do not require the chain to be irreducible.

  ->Exact solution.

  ETRR(t)=0 for all t>=0.
*/

parameters

int
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0

state_variables

DEL, /* yes for states belonging to the nonreachable component
      * and no otherwise */
N,  /* numero de replicas en marcha */
ND

production_rules

/* adds a transition rate from state (DEL=yes, N), N>=1
 * to state (DEL=no, N) */
if DEL && N>=1 action TO_REACH with_rate la*N new_state DEL=no

if N>=1 action FAILURE with_rate la*N

   if N==1 response FAIL new_state DEL=no, N=0, ND=ND0
   if N>1 response COVERED with_prob C new_state N--, ND=ND0
   if N>1 response UNCOVERED with_prob 1-C new_state DEL=no, N=0, ND=ND0

end
```

```
if N==1 action REPAIR with_rate mu new_state N++, ND=ND0

if N>=1 && ND>0 action with_rate la_d new_state ND--
if N>=1 && ND<ND0 action with_rate mu_d new_state ND++

start_state

DEL=yes, N=2, ND=ND0

reward_rate (double)0

initial_probability
    (double)(P0*(DEL==no && N==2 && ND==ND0)+(1-P0)*(DEL==no && N==1 && ND==ND0))
/*
  ->Example description

  Dual repairable system. Units fail with rate la and coverage
  probability C and are repaired by a single repair person with rate
  mu. The unique state with all components failed is "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state.  (i.e. the partition is
  lumpable.)

  ->CTMC description

  4+N0 states; irreducible.

  ->What tests is the example intended for

  Testing of the ETRR measure in the particular case in which all states have
  zero reward rate using any of the numerical methods offered by METFAC2.1
  that do not require the CTMC to have absorbing states.

  ->Exact solution.

  ETRR(t)=0 for all t>=0.

*/

parameters

int
stat_start,  /* ha de valer 0, 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no (la cadena sera la mateixa
                ates que es irreduible */
ND0,         /* l'estat amb N=0 i F=no es "parteix" en ND0+1 estats,
                ND0>=0 */

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
la_d,
C,   /* factor de cobertura */
P0   /* probabilitat inicial de l'estat (N=2,F=no,ND=ND0) es P0;
        prob. init. de l'estat (N=1,F=no,ND=ND0) es 1-P0 */

state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND  /* val sempre 0 per als estats amb N>0 o F=yes; per als
     * estats amb N=0 i F=no, aquesta variable pren els
     * valors ND0, ND0-1,..., 0 */

production_rules

if N>=1 && !F && ND==ND0 action FAILURE with_rate la*N
```

```
   if N==1 response FAIL new_state N--
   if N>1 response COVERED with_prob C new_state N--
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes

end

if N<2 && ND==ND0 action REPAIR with_rate mu
   if F response REPAIR new_state N++, F=no
   if !F response REPAIR new_state N++

end

/* "particio" estat amb N=0 i F=no */
if N==0 && !F && ND>0 action with_rate la_d new_state ND--
if N==0 && !F && ND<ND0 action with_rate mu new_state N++, ND=ND0

start_state

N=stat_start, F= no, ND=ND0

reward_rate 0.0

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Modified dual repairable system: units fail with rate la and
  coverage probability C; the system can only be repaired if there is
  one unfailed unit and the failure of the second one was
  covered. Moreover, both the state with no failed units and the state
  with an unfailed unit in which repair is allowed are "split" into
  several "fake" states in such a way that given the reward rate
  structure, the exact value of the measure can be computed regarding
  those states as lumped into one state. (i.e. the partition is
  lumpable.)

  ->CTMC description

  2*(ND0+2) states; reducible. (There is a transient component plus
  two absorbing states.)

  ->What tests is the example intended for

  Testing of the ETRR measure using any numerical method that requires or
  allows the chain to have absorbing states.

  ->Exact solution.

  It can be computed with Maple (command file: ETRR_8_maple).

*/

parameters

int
stat_start,  /* ha de valer 1 o 2; l'estat des del que es comenca
                a generar es N=stat_start, F=no, ND=ND0 (la cadena sera la
                mateixa) */
ND0,

double
la,  /* tasa de fallo de una replica */
mu,  /* tiempo de reparacion de una replica */
C,   /* factor de cobertura */
la_d,
mu_d,
P0
```

```
state_variables

N,  /* numero de replicas en marcha */
F,  /* fallo del sistema yes/no */
ND

production_rules

if N>=1 && !F action FAILURE with_rate la*N

   if N==1 response FAIL new_state N--, ND=ND0
   if N>1 response COVERED with_prob C new_state N--, ND=ND0
   if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes, ND=ND0

end

if N==1 && !F action REPAIR with_rate mu
   if F response REPAIR new_state N++, F=no, ND=ND0
   if !F response REPAIR new_state N++, ND=ND0

end

if N>=1 && !F && ND>0 action with_rate la_d new_state ND--
if N>=1 && !F && ND<ND0 action with_rate mu_d new_state ND++

start_state

N=stat_start, F= no, ND=ND0

reward_rate (double)(F || N==0)

initial_probability
    (double)(P0*(N==2 && F==no && ND==ND0)+(1-P0)*(N==1 && F==no && ND==ND0))
/*
  ->Example description

  Fictitious system made up of N0 identical units. The system may be
  working or on stand-by. If the system is working, units fail with
  rate la and are repaired by a single repair person with rate mu, and
  the coverage probability is C; if the system is on stand-by, it
  changes to the working state with rate la_u, units fail with rate
  la_s (coverage probability is equal to one), and when there is any
  failed unit, the system is brought to the non-failed state with rate
  mu_s.

  ->CTMC description

  3*N0+1 states (assuming N0>=1); if states are labeled by the 3-tuple
  (N,ON,F), states of the form (N,ON=no,F) make up a transient
  component and states of the form (N,ON=yes,F) make up a trapping
  component. The initial probability is P0 for the state
  (N=N0,ON=yes,F=no), 1-P0 for the state (N=N0,ON=no,F=no) and 0 for
  the remaining states. In this way, if P0=1, the transient component
  will not be reachable.

  ->What tests is the example intended for

  N0=2, P0=1: Testing of the ESSRR measure when there are
  non-reachable states using any of the numerical methods offered by
  METFAC2.1 to solve singular linear systems of equations.

  N0=2, P0=0.5: Testing of the ESSRR measure when there are two
  components (one transient and one trapping) using any of the
  numerical methods offered by METFAC2.1 to solve singular and
  nonsingular linear systems of equations.
```

```
  N0=1, P0=0.5: Testing of the ESSRR measure when there are two
  components (one transient and one trapping) for the particular case
  in which the two linear systems that have to be dealt with (one
  singular and one nonsingular) are of size 2 and, therefore, they are
  solved using the Cramer's rule instead of any of the iterative
  solvers offered by METFAC2.1.

  ->Exact solution.

  N0=2:

   ESSRR = 2*la*(la*C+mu*(1-C))/(mu*mu+2*la*la*C+2*la*mu)

  N0=1:

   ESSRR = la/(la+mu)
*/

parameters

int
N0,    /* no. inicial de repliques */

double
P0,    /* probabilitat inicial de l'estat amb totes les repliques
         * ok i sistema en marxa es P0, i probabilitat inicial
         * de l'estat amb totes les repliques ok i sistema aturat es 1-P0 */
la_s,  /* taxa fallada d'una replica amb sistema aturat */
mu_s,   /* taxa reparacio d'una replica amb sistema aturat */
la_u,  /* taxa posta en marxa */
la,     /* taxa fallada d'una replica amb sistema en marxa */
mu,    /* taxa reparacio d'una replica amb sistema en marxa */
C      /* factor cobertura amb sistema en marxa */

state_variables
N,           /* no. de repliques ok */
ON,    /* yes si sistema en marxa i no altrament */
F      /* yes si sistema fallada de cobertura i no altrament */

production_rules

/* sistema aturat */

if N>=1 && !ON action SOFT_FAILURE with_rate la_s*N new_state N--
if N<N0 && !ON action SOFT_REPAIR with_rate mu_s new_state N=N0
if !ON action ON with_rate la_u new_state ON=yes

/* sistema en marxa */

if ON && N>=1 && !F action FAILURE with_rate la*N
  if N==1 response FAIL new_state N--
  if N>1 response COVERED with_prob C new_state N--
  if N>1 response UNCOVERED with_prob 1-C new_state N--, F=yes
end

if ON && N<N0 action REPAIR with_rate mu
  if F response REPAIR new_state N++, F=no
  if !F response REPAIR new_state N++
end

start_state
N=N0, ON=no, F=no

reward_rate (double)(F || N==0)

initial_probability P0*(N==N0 && ON==yes && F==no)
                    +(1-P0)*(N==N0 && ON==no && F==no)
```

A.2. Fitxers auxiliars d'especificació de CMTC

```
#include "ECRTE_2.h"

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

```
{
  DECLARE_SYMBOLS;

  if (N==2 && (ND==ND0 || ND==ND0-1)) return 1;
  else return 0;
}
#include <math.h>

#include "CCRD_4.h"

/*
 * Returns the reward rate for state (p,m) and mission time t.
 */

double
REWARD(int p, int m, double t)
{
   double val;

   if (p > 0 && m > 0) val = ((pow(m,p)-pow(m-1.0,p))/pow(m,p-1.0))*(1.0/t);
   else val = 1/(3.0*t); /* for the smallest reward rate not to be 0
                        * (it's a particular case of CCRD_SR_comp())
                        * */
   return val;
}


/*
 * Returns the initial probability of state (p,m) given the initial
 * number of processors, np, the initial number of memories, nm, and
 * the parameter p_0, 0<=p_0<=1.
 */

double
INITP(int p, int m, int np, int nm, double p_0)
{
   if (p == np && m == nm) return p_0;
   else if (p == np-1 && m == nm) return (1.0-p_0)/2.0;
   else if (p == np && m == nm-1) return (1.0-p_0)/2.0;
   return 0.0;
}
#include "CIAVD_1.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1 && ND == ND0) return 1;
   else return 0;
}
#include "CIAVD_10.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (PH==0 && N == 0 && !ABS) return 1;
   else return 0;
}
#include "CIAVD_11.h"

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

```
   if (N == N_0 && !ABS) return 1;
   else return 0;
}
#include "CIAVD_12.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == N_0-1 && !ABS) return 1;
   else return 0;
}
#include "CIAVD_13.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1) return 1;
   else return 0;
}
#include "CIAVD_2.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1) return 1;
   else return 0;
}
#include <math.h>

#include "CIAVD_3.h"

/*
 * Returns the reward rate for state (p,m).
 */

double
REWARD(int p, int m)
{
   if (p > 0 && m > 0 && (p+m > 2)) return 1.0;
   else return 0.0;
}


/*
 * Returns the initial probability of state (p,m) given the initial
 * number of processors, np, the initial number of memories, nm, and
 * the parameter p_0, 0<=p_0<=1.
 */

double
INITP(int p, int m, int np, int nm, double p_0)
{
   if (p == np && m == nm) return p_0;
   else if (p == np-1 && m == nm) return (1.0-p_0)/2.0;
   else if (p == np && m == nm-1) return (1.0-p_0)/2.0;
   return 0.0;
}


/*
 * Flags state (NP, NM) as the regenerative state.
 */
```

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

   if (P == NP && M == NM) return 1;
   else return 0;
}
#include "CIAVD_4.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1 && ND == ND0) return 1;
   else return 0;
}
#include "CIAVD_5.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 0 && ND == ND0) return 1;
   else return 0;
}
#include "CIAVD_6.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1 && ND == ND0) return 1;
   else return 0;
}
#include "CIAVD_7.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (!ABS && N == 1 && ND0 == ND0_INIT) return 1;
   else return 0;
}
#include "CIAVD_8.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 1 && ND == ND0 && !ABS) return 1;
   else return 0;
}
#include "CIAVD_9.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
   DECLARE_SYMBOLS;

   if (N == 0 && !ABS) return 1;
   else return 0;
```

```
}
#include "CRDDS_1.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 2;
}
#include "CRDDS_2.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 2;
}
#include "CRDDS_3.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}

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

```
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 2;
}
#include "CRDDS_4.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 0;
}
#include "CRDDS_5.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 0;
}
#include "CRDDS_6.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (S==5 || S==7) return 1;
  else return 0;
}
#include "CRDDS_7.h"


int
subset(int sv[], int ipar[], double dpar[])
{
```

```
  DECLARE_SYMBOLS;
  if (S==5 || S==7) return 1;
  else return 0;
}
#include "CRDDS_8.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  else return 2;
}
#include "CRDTE_2.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "CRDTE_3.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "CRDTE_4.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "CRDTE_5.h"

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

```
  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "CRDTE_6.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "EARR_1.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_10.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 1 && F && ND == 0) return 1;
  return 0;
}
#include "EARR_13.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_2.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_3.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}
#include "EARR_4.h"
```

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

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_5.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_6.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_7.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "EARR_8.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "EARR_9.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 1 && F && ND == 0) return 1;
  return 0;
}
#include "ECRDS_1.h"

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

```
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ECRDS_2.h"


long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}


int
subset (int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N>=1 && !F) return 1;
  else return 0;
}
#include "ECRDS_3.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (S==5 || S==7) return 1;
  else return 0;
}
#include "ECRDS_4.h"


int
subset(int sv[], int ipar[], double dpar[])
{
  DECLARE_SYMBOLS;
  if (N==2 && !F) return 1;
  else return 0;
}

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ECRTE_1.h"

int
pivot(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && (ND==ND0 || ND==ND0-1)) return 1;
```

```
  else return 0;
}
#include <math.h>

#include "CCRD_3.h"

/*
 * Returns the reward rate for state (p,m) and mission time t.
 */

double
REWARD(int p, int m, double t)
{
   double val;

   val = ((pow(m,p)-pow(m-1.0,p))/pow(m,p-1.0))*(1.0/t);
   return val;

}


/*
 * Returns the initial probability of state (p,m) given the initial
 * number of processors, np, the initial number of memories, nm, and
 * the parameter p_0, 0<=p_0<=1.
 */

double
INITP(int p, int m, int np, int nm, double p_0)
{
   if (p == np && m == nm) return p_0;
   else if (p == np-1 && m == nm) return (1.0-p_0)/2.0;
   else if (p == np && m == nm-1) return (1.0-p_0)/2.0;
   return 0.0;
}
#include "ETRR_9.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 1 && F && ND == ND0) return 1;
  return 0;
}
#include "ECRTE_4.h"

int
pivot(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "ESSRR_1.h"

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
  return 0;
}
#include "ESSRR_10.h"
```

```
long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N<N0 && !F) return 1;
  if (N==N0 && !F) return 2;
  if (F) return 3;
}
#include <math.h>
#include <float.h>
#include "ESSRR_11.h"


/*
 * Reward rate is n*factor, where factor is the reciprocal of the throughput
 * gamma=lambda*(1-p_loss), p_loss = (1-rho)*rho^C/(1-rho^(C+1)),
 * rho=lambda/mu. With this reward rate, the ESSRR measure will be the
 * expected waiting time in the system.
 */

double
reward(int c, int n, double lambda, double mu)
{
    static double factor = -1.0;

    if (factor == -1.0) {
       double rho = lambda/mu;

       if (rho > pow(1.0/DBL_EPSILON, 1.0/(c+1.0))) { /* p_loss */
           factor = (rho-1.0)/rho;
       } else if (rho != 1.0) {
           factor = (1.0-rho)/(pow(rho, -c)-rho);
       } else {
           factor = 1.0-c/(1.0+c);
       }
       factor = 1.0/(lambda*(1.0-factor));               /* throughput^{-1} */
    }
    return factor*n;
}
#include "ESSRR_12.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ESSRR_2.h"


long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ESSRR_3.h"


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

```
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ESSRR_4.h"


long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  return S;
}
#include "ESSRR_6.h"

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  return S;
}
#include "ESSRR_7.h"


long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ESSRR_8.h"

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (F) return 2;
  if (N==0) return 3;
}
#include "ESSRR_9.h"

long
block(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;
  if (N>0 && !F) return 1;
  if (N==0 || F) return 2;
}
#include "ETRR_1.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_10.h"


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

```
{
  DECLARE_SYMBOLS;

  if (N == 1 && F && ND == 0) return 1;
  return 0;
}
#include "ETRR_11.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_14.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_2.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_3.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F) return 1;
  return 0;
}
#include "ETRR_4.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_5.h"


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

```
  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_6.h"

int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
#include "ETRR_7.h"


int
regstat(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ETRR_8.h"


int
regstat (int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N == 2 && !F && ND == ND0) return 1;
  return 0;
}
#include "ECRTE_3.h"

int
pivot(int sv[], int ipar[], double dpar[], long index)
{
  DECLARE_SYMBOLS;

  if (N==2 && ND==ND0) return 1;
  else return 0;
}
```

A.3. Fitxers de comandes per Maple

```
Digits:=30:
with(linalg):
with(inttrans):
```

```
A:=matrix([[s+2*ld,0,-mu,0],[-2*ld*(1-C),s,0,0],[-2*ld*C,0,s+mu+ld,0],[0,0,-
ld,s]]):
b:=vector([p0,0,1-p0,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
C:=0.99:
p0:=0.9:
np:=1:
t_arr:=vector([1000]):
ld_arr:=vector([0.001]):
mu_arr:=vector([0.001]):
for i from 1 to np do
  t:=t_arr[i]:
  ld:=ld_arr[i]:
  mu:=mu_arr[i]:
  printf("%2.28g\n", evalf(actual_sym_sol)):
end do:
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([10]):
nt:=1:
pv:=vector([0.999]):
np:=1:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    printf("%2.28g\n", evalf(ciavd)):
  end do:
end do:
quit:
#
# Worksheet that  computes the cumulative reward distribution of the
# multiprocessor system described in V. Grassi, L. Donatiello, and G.
# Iazeolla, "Performability Evaluation of a Multicomponent
# Fault-Tolerant System," IEEE Trans. on Reliability, 37(2): 216-222,
# June 1988. The recurrences, which are also taken from that paper, can
# be used for any acyclic CTMC provided that:
# 1) its state space is of the form E=(0,...,S), where 0 is the
# absorbing state that represents system failure and S is the fault-free
# state;
# 2) if L_k denotes the set of states i in E with output rate
# ld[i]=ld[k] and F_k denotes the set of states i in E with reward rate
#      r[i]=r[k], then the following conditions hold:
# 2-1) (ld[i]-ld[j])/(ld[k]-ld[j]) is different from
# (r[i]-r[j])/(r[k]-r[j]) for all i,k,j in E such that  i is different
# from k, k is different from j, and          nor i neither k belong to
# F_j or L_j;
# 2-2) ld[i] is different from ld[j] or r[i] is different from r[j] for
# all i,j in E such that i is different from j.
#
# computes M_k^(s), 0<=s<=S, 0<=k<=s.
# global variables: the array ld[0]...ld[S] of output rates
comp_M:=proc(S) local s,k,i:global M,ld:
  M[0,0]:=0.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      if (ld[s]<>ld[k]) then
        M[k,s]:=M[k,s-1]:
      else
        M[k,s]:=M[k,s-1]+1:
      end if:
    end do:
```

```
    M[s,s]:=M[0,s]:
    for i from 1 to s-1 do M[k,s]:=max(M[k,s],M[i,s]) end do:
  end do:
end proc:

# computes W_k^(s), 0<=s<=S, 0<=k<=s. (This is recurrence B4 in the paper.)
# global variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_W:=proc(S) local s,k,j,val:global W,Q,ld,r:
  W[0,0]:=1.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      W[k,s]:=0.0:
      if (r[k]=r[s]) then
        for j from k to s-1 do W[k,s]:=W[k,s]+Q[s,j]*W[k,j] end do:
        W[k,s]:=W[k,s]*ld[s]/(ld[s]-ld[k]):
        if (type(W[k,s],undefined)) then
          error "W[k=%1,s=%2] undefined (ld[s]=%3,ld[k]=%4)",k,s,ld[s],ld[k]:
        end if:
        if (W[k,s]=Float(infinity)) then
          error "W[k=%1,s=%2] overflows",k,s:
        end if:
      end if:
    end do:
    val:=0.0:
    for k from 0 to s-1 do val:=val+W[k,s] end do:
    W[s,s]:=1.0-val:
  end do:
end proc:

thet:=proc(a,b,c) local num,den,val:global ld,r:
  num:=r[a]-r[b]:
  den:=ld[c]*r[a]+ld[a]*r[b]+ld[b]*r[c]
      -(ld[c]*r[b]+ld[b]*r[a]+ld[a]*r[c]):
  val:=num/den:
  if (type(expr,undefined)) then
    error "undefined for a:%1,b=%2,c=%3",a,b,c:
  end if:
  if (val=Float(infinity)) then
    error "overflow for
a:%1,b:%2,c:%3:ld[a]=%4,ld[b]=%5,ld[c]=%6,r[a]=%7,r[b]=%8,r[c]=%9",a,b,c,ld[a],ld
[b],ld[c],r[a],r[b],r[c]:
  end if:
  return val:
end proc:

# computes H_k^(s)(i), 0<=s<=S, 0<=k<=s, 0<=i<=s, i not belonging to
# the reunion of sets F_k and L_k, and Z_k^(s)(i), 0<=s<=S, 0<=k<=s,
# 1<=i<=M_k^(s). (These are recurrences B1-B3.)
# blobal variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_HZ:=proc(S) local s,k,i,j,p,val:global ld,r,Q,H,Z:
  for s from 0 to S do
    for k from 0 to s-1 do

#                        H_k^(s)(i), k=0,...,s-1, i=0,...,s-1,
# i\notin(F_k\cup L_k), Z_k^(s)(i), k=0,...,s-1,i=1,...,M_k^(s)

        for i from 0 to s-1 do
          if (ld[i]<>ld[k] and r[i]<>r[k]) then
            val:=0.0:
            for j from max(i,k) to s-1 do
```

```
              val:=val+Q[s,j]*H[k,j,i]
            end do:
            H[k,s,i]:=ld[s]*thet(i,k,s)*val
          end if:
          if (i>=1 and i<=M[k,s]) then
            val:=0.0:
            if (ld[s]<>ld[k] and r[s]<>r[k]) then
              for j from k to s-1 do
                for p from i to M[k,j] do

val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^(p-i))*Z[k,j,p]
              end do:
            end do:
            end if:
            Z[k,s,i]:=val*ld[s]/(ld[s]-ld[k]):
            val:=0.0:
            if (ld[s]=ld[k]) then
              for j from k to s-1 do
                if (i>=2 and i<=M[k,j]+1) then
                  val:=val+Q[s,j]*Z[k,j,i-1]
                end if:
              end do:
              if (i=1) then
                for j from k to s-1 do
                  val:=val+Q[s,j]*W[k,j]
                end do:
                for j from k to s-1 do
                  for p from 0 to j do
                    if (ld[p]<>ld[k] and r[p]<>r[k]) then
                      val:=val+Q[s,j]*H[k,j,p]/(ld[p]-ld[k])
                    end if:
                  end do:
                end do:
                Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(r[s]-r[k]):
                if (type(Z[k,s,i],undefined)) then
                  error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
                end if:
                if (Z[k,s,i]=Float(infinity)) then
                  error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
                end if:
              end if:
            end if:
            val:=0.0:
            if (r[s]=r[k]) then
              for j from k to s-1 do
                if (1<=i and i<=M[k,j]) then
                  val:=val+Q[s,j]*Z[k,j,i]:
                end if:
              end do:
              Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(ld[s]-ld[k]):
              if (type(Z[k,s,i],undefined)) then
                error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
              end if:
              if (Z[k,s,i]=Float(infinity)) then
                error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
              end if:
            end if:
          end if:
#
        end do:
#                              H_k^(s)(s), k=0,...,s-1, k\notin L_s\cup
# F_s

          if (ld[k]<>ld[s] and r[k]<>r[s]) then
            val:=0:
            for j from k to s-1 do
              for i from 0 to j do
                if (ld[i]<>ld[k] and r[i]<>r[k]) then
```

```
                  val:=val+Q[s,j]*thet(s,k,i)*H[k,j,i]
                end if:
              end do:
            end do:
            for j from k to s-1 do
              for i from 1 to M[k,j] do

val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^i)*Z[k,j,i]
              end do:
            end do:
            for j from k to s-1 do
              val:=val+Q[s,j]*W[k,j]
            end do:
            H[k,s,s]:=val*ld[s]
          end if:
      end do:
#                   H_s^(s)(k), k=0,...,s-1, k\notin L_s\cup F_s:
# Z_s^(s)(i), i=1,...,M_s^(s)

      for k from 0 to s-1 do
        H[s,s,k]:=H[k,s,s]
      end do:
      for i from 1 to M[s,s] do
        val:=0.0:
        for k from 0 to s-1 do
          if (i<=M[k,s]) then
            val:=val+Z[k,s,i]
          end if:
        end do:
        Z[s,s,i]:=-val
      end do:
    end do:
  end proc:

  # computes Prob{CR(t)<=y}, i.e. the probability that the reward accumulated
  # in the time interval [0,t] is smaller than or equal to y being S the initial
  # state
  # global variables:
  # the array ld[0] ... ld[S]
  # the array r[0] ... r[S] of reward rates
  # the array M filled in by comp_M()
  # the array W filled in by comp_W()
  # the arrays Z and H filled in by comp_HZ()
  CRD:=proc(tm,y,S) local crd,aux1,aux2,k,i:global H,ld,r,Z,M,W:
  crd:=0.0:
  for k from 0 to S do
    for i from 0 to S do
      if (r[i]<>r[k] and ld[i]<>ld[k]) then
        aux1:=H[k,S,i]*exp(-ld[k]*tm)/(ld[i]-ld[k]):
        if (y>r[k]*tm) then
           aux2:=1.0-exp(-((ld[i]-ld[k])/(r[i]-r[k]))*(y-r[k]*tm))
        else
          aux2:=0.0
        end if:
        crd:=crd+aux1*aux2:
      end if:
    end do:
  end do:
  for k from 0 to S do
    for i from 1 to M[k,S] do
      if (y>r[k]*tm) then
        crd:=crd+(Z[k,S,i]/i!)*exp(-ld[k]*tm)*((y-r[k]*tm)^i)
      end if:
    end do:
  end do:
  for k from 0 to S do
    if (y>r[k]*tm) then
      crd:=crd+W[k,S]*exp(-ld[S]*tm):
```

```
    end if:
  end do:
  return crd:
end proc:

# Implements the multiprocessor system described in V. Grassi, L.
# Donatiello, and G. Iazeolla, "Performability Evaluation of a
# Multicomponent Fault-Tolerant System," IEEE Trans. on Reliability,
# 37(2): 216-222, June 1988. The system consists of P processors and M
# memories that fail with rate ldp and ldm, respectively. The system is
# up if there are at least one fault-free processor and one fault-free
# memory. States are (conceptually) labelled (p,m), where p and m are
# the no. of fault-free processors and memories, respectively. State
# (p,m) is assigned the reward rate m*(1-(1-1/m)^p)/T, which is an
# estimate of the steady-state average number of active processors in
# [0,t].
# returns the number of states
count_st:=proc(P,M) local idx,p,m:
  idx:=0:
  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
    end do:
  end do:
  return (idx+1):
end proc:

# assigns indices to states and bookkeeps them in the global array
# state_idx[p,m], 1<=p<=P, 1<=m<=M (State (p,m) has index state_idx[p,m].)
#
# Note: for convenience, state_idx[0,m], 1<=m<=M and state_idx[p,0], 1<=p<=P
# are all set to 0
set_stidx:=proc(P,M,S) local idx,p,m:global state_idx:
  idx:=0:
  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
      if (idx>S) then
        error "state index (%1) larger than the expected maximum one (%2)",idx,S:
      end if:
      state_idx[p,m]:=idx:
    end do:
  end do:
  for p from 1 to P do state_idx[p,0]:=0 end do:
  for m from 1 to M do state_idx[0,m]:=0 end do:
end proc:

# returns the reward rate for state (p,m) when the mission time is tm
strwd:=proc(p,m,tm) local val:global state_idx:
  val:=((m^p-(m-1)^p)/m^(p-1))*(1.0/tm):
  return val:
end proc:

# computes the states' reward rate
comp_r:=proc(P,M,tm) local p,m,idx:global state_idx,r:
  for p from 1 to P do
    for m from 1 to M do
      idx:=state_idx[p,m]:
      r[idx]:=strwd(p,m,tm):
    end do:
  end do:
  r[0]:=0.0:
end proc:

# computes the states' initial probability, which is 0 for them all except
# that state (P,M) is assigned initial probability P_0, and both state
# (P-1,M) and (P,M-1) are assigned initial probability (1-P_0)/2
comp_initp:=proc(P,M,P_0) local p,m:global init_p,state_idx:
```

```
  if (P_0<0.0 or P_0>1) then
    error "wrong value for P_0 (%1)", P_0:
  end if:
  for p from 1 to P do
    for m from 1 to M do
      init_p[state_idx[p,m]]:=0.0:
    end do:
  end do:
  init_p[0]:=0.0:
  init_p[state_idx[P,M]]:=P_0:
  init_p[state_idx[P-1,M]]:=(1-P_0)/2.0:
  init_p[state_idx[P,M-1]]:=(1-P_0)/2.0:
end proc:

# computes the output rates and transition probabilities (arrays ld[]
# and Q[], respectively)
comp_ld_Q:=proc(P,M,S,ldp,ldm) local p,m,i,j,val:global
state_idx,ld,Q:
  for i from 0 to S do
    ld[i]:=0.0:
    for j from 0 to S do
      Q[i,j]:=0.0:
    end do:
  end do:
  for p from 1 to P do
    for m from 1 to M do
      i:=state_idx[p,m]:
      ld[i]:=p*ldp+m*ldm:
      if (state_idx[p-1,m]<>state_idx[p,m-1]) then
        j:=state_idx[p-1,m]:
        Q[i,j]:=p*ldp/ld[i]:
        j:=state_idx[p,m-1]:
        Q[i,j]:=m*ldm/ld[i]:
      else
        j:=state_idx[p-1,m]:
        Q[i,j]:=1.0:
      end if:
    end do:
  end do:
end proc:
Digits:=70:
ldp:=.6e-4:
ldm:=1e-4:
T:=1000:
P:=3:
M:=3:
P_0:=0.75:
S:=count_st(P,M)-1:
state_idx:=array(0..P,0..M):
r:=array(0..S):
ld:=array(0..S):
init_p:=array(0..S):
Q:=array(0..S,0..S):
set_stidx(P,M,S):
comp_r(P,M,T):
comp_ld_Q(P,M,S,ldp,ldm):
comp_initp(P,M,P_0):
M:=array(0..S,0..S):
comp_M(S):
W:=array(0..S,0..S):
comp_W(S):
H:=array(0..S,0..S,0..S):
dim_Z:=1:
for s from 0 to S do
  for k from 0 to s-1 do
    dim_Z:=max(dim_Z,M[k,s]):
  end do:
end do:
```

```
Z:=array(0..S,0..S,0..dim_Z):
comp_HZ(S):
for y in 1e-6,0.8,1,1.3,1.5,1.55,1.667,1.7,1.75,2,2.111,2.2 do
  CCRD:=0.0:
  for i from 0 to S do
    if (init_p[i]>0.0) then
      CCRD:=CCRD+init_p[i]*CRD(T,y,i):
    end if:
  end do:
  CCRD:=1.0-CCRD:
  if (CCRD < 1e-40) CCRD:=0:
  printf("%2.28g\n", CCRD):
end do:
quit:
#
# Worksheet that  computes the cumulative reward distribution of the
# multiprocessor system described in V. Grassi, L. Donatiello, and G.
# Iazeolla, "Performability Evaluation of a Multicomponent
# Fault-Tolerant System," IEEE Trans. on Reliability, 37(2): 216-222,
# June 1988, except that the smalles reward rate is 1/(3*T) instead
# of 0. The recurrences, which are also taken from that paper, can
# be used for any acyclic CTMC provided that:
# 1) its state space is of the form E=(0,...,S), where 0 is the
# absorbing state that represents system failure and S is the fault-free
# state;
# 2) if L_k denotes the set of states i in E with output rate
# ld[i]=ld[k] and F_k denotes the set of states i in E with reward rate
#      r[i]=r[k], then the following conditions hold:
# 2-1) (ld[i]-ld[j])/(ld[k]-ld[j]) is different from
# (r[i]-r[j])/(r[k]-r[j]) for all i,k,j in E such that  i is different
# from k, k is different from j, and          nor i neither k belong to
# F_j or L_j;
# 2-2) ld[i] is different from ld[j] or r[i] is different from r[j] for
# all i,j in E such that i is different from j.
#
# IMPORTANT NOTE: it seems that the algorithm does not break down
# as long as there not exist an integer k>0 such that ldp=k*ldm or ldm=k*ldp.

#
# computes M_k^(s), 0<=s<=S, 0<=k<=s.
# global variables: the array ld[0]...ld[S] of output rates
comp_M:=proc(S) local s,k,i:global M,ld:
  M[0,0]:=0.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      if (ld[s]<>ld[k]) then
        M[k,s]:=M[k,s-1]:
      else
        M[k,s]:=M[k,s-1]+1:
      end if:
    end do:
    M[s,s]:=M[0,s]:
    for i from 1 to s-1 do M[k,s]:=max(M[k,s],M[i,s]) end do:
  end do:
end proc:

# computes W_k^(s), 0<=s<=S, 0<=k<=s. (This is recurrence B4 in the paper.)
# global variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_W:=proc(S) local s,k,j,val:global W,Q,ld,r:
  W[0,0]:=1.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      W[k,s]:=0.0:
      if (r[k]=r[s]) then
```

```
        for j from k to s-1 do W[k,s]:=W[k,s]+Q[s,j]*W[k,j] end do:
        W[k,s]:=W[k,s]*ld[s]/(ld[s]-ld[k]):
        if (type(W[k,s],undefined)) then
          error "W[k=%1,s=%2] undefined (ld[s]=%3,ld[k]=%4)",k,s,ld[s],ld[k]:
        end if:
        if (W[k,s]=Float(infinity)) then
          error "W[k=%1,s=%2] overflows",k,s:
        end if:
      end if:
    end do:
    val:=0.0:
    for k from 0 to s-1 do val:=val+W[k,s] end do:
    W[s,s]:=1.0-val:
  end do:
end proc:

thet:=proc(a,b,c) local num,den,val:global ld,r:
  num:=r[a]-r[b]:
  den:=ld[c]*r[a]+ld[a]*r[b]+ld[b]*r[c]
      -(ld[c]*r[b]+ld[b]*r[a]+ld[a]*r[c]):
  val:=num/den:
  if (type(expr,undefined)) then
    error "undefined for a:%1,b=%2,c=%3",a,b,c:
  end if:
  if (val=Float(infinity)) then
    error "overflow for
a:%1,b:%2,c:%3:ld[a]=%4,ld[b]=%5,ld[c]=%6,r[a]=%7,r[b]=%8,r[c]=%9",a,b,c,ld[a],ld
[b],ld[c],r[a],r[b],r[c]:
  end if:
  return val:
end proc:

# computes H_k^(s)(i), 0<=s<=S, 0<=k<=s, 0<=i<=s, i not belonging to
# the reunion of sets F_k and L_k, and Z_k^(s)(i), 0<=s<=S, 0<=k<=s,
# 1<=i<=M_k^(s). (These are recurrences B1-B3.)
# blobal variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_HZ:=proc(S) local s,k,i,j,p,val:global ld,r,Q,H,Z:
  for s from 0 to S do
    for k from 0 to s-1 do

#                        H_k^(s)(i), k=0,...,s-1, i=0,...,s-1,
# i\notin(F_k\cup L_k), Z_k^(s)(i), k=0,...,s-1,i=1,...,M_k^(s)

        for i from 0 to s-1 do
          if (ld[i]<>ld[k] and r[i]<>r[k]) then
            val:=0.0:
            for j from max(i,k) to s-1 do
              val:=val+Q[s,j]*H[k,j,i]
            end do:
            H[k,s,i]:=ld[s]*thet(i,k,s)*val
          end if:
          if (i>=1 and i<=M[k,s]) then
            val:=0.0:
            if (ld[s]<>ld[k] and r[s]<>r[k]) then
              for j from k to s-1 do
                for p from i to M[k,j] do

val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^(p-i))*Z[k,j,p]
                end do:
              end do:
            end if:
            Z[k,s,i]:=val*ld[s]/(ld[s]-ld[k]):
            val:=0.0:
            if (ld[s]=ld[k]) then
```

```
              for j from k to s-1 do
                if (i>=2 and i<=M[k,j]+1) then
                  val:=val+Q[s,j]*Z[k,j,i-1]
                end if:
              end do:
              if (i=1) then
                for j from k to s-1 do
                  val:=val+Q[s,j]*W[k,j]
                end do:
                for j from k to s-1 do
                  for p from 0 to j do
                    if (ld[p]<>ld[k] and r[p]<>r[k]) then
                      val:=val+Q[s,j]*H[k,j,p]/(ld[p]-ld[k])
                    end if:
                  end do:
                end do:
                Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(r[s]-r[k]):
                if (type(Z[k,s,i],undefined)) then
                  error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
                end if:
                if (Z[k,s,i]=Float(infinity)) then
                  error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
                end if:
              end if:
            end if:
            val:=0.0:
            if (r[s]=r[k]) then
              for j from k to s-1 do
                if (1<=i and i<=M[k,j]) then
                  val:=val+Q[s,j]*Z[k,j,i]:
                end if:
              end do:
              Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(ld[s]-ld[k]):
              if (type(Z[k,s,i],undefined)) then
                error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
              end if:
              if (Z[k,s,i]=Float(infinity)) then
                error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
              end if:
            end if:
          end if:
#
        end do:
#                             H_k^(s)(s), k=0,...,s-1, k\notin L_s\cup
# F_s

          if (ld[k]<>ld[s] and r[k]<>r[s]) then
            val:=0:
            for j from k to s-1 do
              for i from 0 to j do
                if (ld[i]<>ld[k] and r[i]<>r[k]) then
                  val:=val+Q[s,j]*thet(s,k,i)*H[k,j,i]
                end if:
              end do:
            end do:
            for j from k to s-1 do
              for i from 1 to M[k,j] do

val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^i)*Z[k,j,i]
              end do:
            end do:
            for j from k to s-1 do
              val:=val+Q[s,j]*W[k,j]
            end do:
            H[k,s,s]:=val*ld[s]
          end if:
      end do:
#                   H_s^(s)(k), k=0,...,s-1, k\notin L_s\cup F_s:
```

```
# Z_s^(s)(i), i=1,...,M_s^(s)

      for k from 0 to s-1 do
        H[s,s,k]:=H[k,s,s]
      end do:
      for i from 1 to M[s,s] do
        val:=0.0:
        for k from 0 to s-1 do
          if (i<=M[k,s]) then
            val:=val+Z[k,s,i]
          end if:
        end do:
        Z[s,s,i]:=-val
      end do:
    end do:
  end proc:

  # computes Prob{CR(t)<=y}, i.e. the probability that the reward accumulated
  # in the time interval [0,t] is smaller than or equal to y being S the initial
  # state
  # global variables:
  # the array ld[0] ... ld[S]
  # the array r[0] ... r[S] of reward rates
  # the array M filled in by comp_M()
  # the array W filled in by comp_W()
  # the arrays Z and H filled in by comp_HZ()
  CRD:=proc(tm,y,S) local crd,aux1,aux2,k,i:global H,ld,r,Z,M,W:
  crd:=0.0:
  for k from 0 to S do
    for i from 0 to S do
      if (r[i]<>r[k] and ld[i]<>ld[k]) then
        aux1:=H[k,S,i]*exp(-ld[k]*tm)/(ld[i]-ld[k]):
        if (y>r[k]*tm) then
           aux2:=1.0-exp(-((ld[i]-ld[k])/(r[i]-r[k]))*(y-r[k]*tm))
        else
          aux2:=0.0
        end if:
        crd:=crd+aux1*aux2:
      end if:
    end do:
  end do:
  for k from 0 to S do
    for i from 1 to M[k,S] do
      if (y>r[k]*tm) then
        crd:=crd+(Z[k,S,i]/i!)*exp(-ld[k]*tm)*((y-r[k]*tm)^i)
      end if:
    end do:
  end do:
  for k from 0 to S do
    if (y>r[k]*tm) then
      crd:=crd+W[k,S]*exp(-ld[S]*tm):
    end if:
  end do:
  return crd:
end proc:

# Implements the multiprocessor system described in [1] V. Grassi, L.
# Donatiello, and G. Iazeolla, "Performability Evaluation of a
# Multicomponent Fault-Tolerant System," IEEE Trans. on Reliability,
# 37(2): 216-222, June 1988 but using _different_ reward rates.
# The system consists of P processors and M
# memories that fail with rate ldp and ldm, respectively. The system is
# up if there are at least one fault-free processor and one fault-free
# memory. States are (conceptually) labelled (p,m), where p and m are
# the no. of fault-free processors and memories, respectively. As in [1], if
# p>0 and m>0, state (p,m) is assigned the reward rate m*(1-(1-1/m)^p)/T,
# which is an estimate of the steady-state average number of active
# processors in [0,t]; unlike [1], if p=m=0, the assigned reward rate is 1/(3*T).
```

```
# returns the number of states
count_st:=proc(P,M) local idx,p,m:
  idx:=0:

  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
    end do:
  end do:
  return (idx+1):
end proc:

# assigns indices to states and bookkeeps them in the global array
# state_idx[p,m], 1<=p<=P, 1<=m<=M (State (p,m) has index state_idx[p,m].)
#
# Note: for convenience, state_idx[0,m], 1<=m<=M and state_idx[p,0], 1<=p<=P
# are all set to 0
set_stidx:=proc(P,M,S) local idx,p,m:global state_idx:
  idx:=0:
  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
      if (idx>S) then
        error "state index (%1) larger than the expected maximum one (%2)",idx,S:
      end if:
      state_idx[p,m]:=idx:
    end do:
  end do:
  for p from 1 to P do state_idx[p,0]:=0 end do:
  for m from 1 to M do state_idx[0,m]:=0 end do:
end proc:

# returns the reward rate for state (p,m) when the mission time is tm
strwd:=proc(p,m,tm) local val:global state_idx:
  if (p>0 and m>0) then
    val:=((m^p-(m-1)^p)/m^(p-1))*(1.0/tm):
  else
    val:=1/(3*tm):
  end if:
  return val:
end proc:

# computes the states' reward rate
comp_r:=proc(P,M,tm) local p,m,idx:global state_idx,r:
  for p from 1 to P do
    for m from 1 to M do
      idx:=state_idx[p,m]:
      r[idx]:=strwd(p,m,tm):
    end do:
  end do:
  r[0]:=strwd(0,0,tm):
end proc:

# computes the states' initial probability, which is 0 for them all except
# that state (P,M) is assigned initial probability P_0, and both state
# (P-1,M) and (P,M-1) are assigned initial probability (1-P_0)/2
comp_initp:=proc(P,M,P_0) local p,m:global init_p,state_idx:
  if (P_0<0.0 or P_0>1) then
    error "wrong value for P_0 (%1)", P_0:
  end if:
  for p from 1 to P do
    for m from 1 to M do
      init_p[state_idx[p,m]]:=0.0:
    end do:
  end do:
  init_p[0]:=0.0:
  init_p[state_idx[P,M]]:=P_0:
```

```
  init_p[state_idx[P-1,M]]:=(1-P_0)/2.0:
  init_p[state_idx[P,M-1]]:=(1-P_0)/2.0:
end proc:

# computes the output rates and transition probabilities (arrays ld[]
# and Q[], respectively)
comp_ld_Q:=proc(P,M,S,ldp,ldm) local p,m,i,j,val:global
state_idx,ld,Q:
  for i from 0 to S do
    ld[i]:=0.0:
    for j from 0 to S do
      Q[i,j]:=0.0:
    end do:
  end do:
  for p from 1 to P do
    for m from 1 to M do
      i:=state_idx[p,m]:
      ld[i]:=p*ldp+m*ldm:
      if (state_idx[p-1,m]<>state_idx[p,m-1]) then
        j:=state_idx[p-1,m]:
        Q[i,j]:=p*ldp/ld[i]:
        j:=state_idx[p,m-1]:
        Q[i,j]:=m*ldm/ld[i]:
      else
        j:=state_idx[p-1,m]:
        Q[i,j]:=1.0:
      end if:
    end do:
  end do:
end proc:
Digits:=70:
ldp:=.6e-4:
ldm:=1e-4:
T:=1000:
P:=3:
M:=3:
P_0:=0.75:
S:=count_st(P,M)-1:
state_idx:=array(0..P,0..M):
r:=array(0..S):
ld:=array(0..S):
init_p:=array(0..S):
Q:=array(0..S,0..S):
set_stidx(P,M,S):
comp_r(P,M,T):
comp_ld_Q(P,M,S,ldp,ldm):
comp_initp(P,M,P_0):
M:=array(0..S,0..S):
comp_M(S):
W:=array(0..S,0..S):
comp_W(S):
H:=array(0..S,0..S,0..S):
dim_Z:=1:
for s from 0 to S do
  for k from 0 to s-1 do
    dim_Z:=max(dim_Z,M[k,s]):
  end do:
end do:
Z:=array(0..S,0..S,0..dim_Z):
comp_HZ(S):
for y in 0.1,0.95,1.8,1.9,0.1,2.5 do
  CCRD:=0.0:
  for i from 0 to S do
    if (init_p[i]>0.0) then
      CCRD:=CCRD+init_p[i]*CRD(T,y,i):
    end if:
  end do:
  CCRD:=1.0-CCRD:
```

```
  printf("%2.28g\n", CCRD):
end do:
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([10]):
nt:=1:
pv:=vector([0.1,0.5,0.9,0.99,0.999,1.1]):
np:=6:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    if (p>=1) then
      actual_ciavd:=0:
    else
      actual_ciavd:=evalf(ciavd):
    end if:
    printf("%2.28g\n", actual_ciavd):
  end do:
end do:
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([0.1,10,100]):
nt:=3:
pv:=vector([0.1,0.5,0.9,0.99,0.999]):
np:=5:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    printf("%2.28g\n", evalf(ciavd)):
  end do:
end do:
quit:
#
#
# Model parameter values: NDO=10,la=1E-4,la_ph=1E-3,mu=0.02,C=0.95,PO=0.5
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using A algorithm with abs. err. 1E-12
np:=5:
ciavd:=vector([9.78897348477680728784e-01,
               9.51900983616881801730e-01,
               8.94275667811246344208e-01,
               8.73937388321453734186e-01,
               8.71700636342605195139e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
#
# Model parameter values: NDO=10,la=1E-4,mu=1,C=0.95,PO=0.5
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#
```

```
# Values computed using A algorithm with abs. err. 1E-12
np:=5:
ciavd:=vector([8.7455660900534126778 2e-01,
               8.7280775016491440876 4e-01,
               8.7104639400670225146 7e-01,
               7.9488395443288306641 0e-01,
               6.9565381239421852921 4e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
#
# Model parameter values: NDO=10,la=1E-4,mu=1,C=0.95,PO=0.5
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using A algorithm with abs. err. 1E-12
np:=5:
ciavd:=vector([9.9948344457246007799 3e-01,
               9.9748476839769595336 8e-01,
               9.9538343366011772239 7e-01,
               8.2565297271092463748 9e-01,
               6.9618469708561969699 9e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
Digits:=30:
la:=1e-4:
mu:=1:
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([10]):
nt:=1:
pv:=vector([0.1,0.5,0.9,0.99,0.999]):
np:=5:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    printf("%2.28g\n", evalf(ciavd)):
  end do:
end do:
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([10]):
nt:=1:
pv:=vector([0.999]):
np:=1:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    printf("%2.28g\n", evalf(ciavd)):
  end do:
end do:
quit:
#
# Worksheet that  computes the cumulative reward distribution of the
# multiprocessor system described in V. Grassi, L. Donatiello, and G.
# Iazeolla, "Performability Evaluation of a Multicomponent
# Fault-Tolerant System," IEEE Trans. on Reliability, 37(2): 216-222,
```

```
# June 1988, except that the reward rates are different: states such
# that the number of unfailed
# processors, p, is larger than 0, the number of unfailed memories, m,
# is larger than 0, and p+m>2 are assigned a reward rate equal to 1;
# the remaining states are assigned a reward rate equal to 0. In this
# way, regarding as up states those with reward rate equal to 1 and
# as down states those with reward rate equal to 0, the value of the
# distribution for time t and abscissa s equals
# the interval availability for abscissa s/t. The recurrences,
# which are also taken from that paper, can be used for any acyclic
# CTMC provided that:
# 1) its state space is of the form E=(0,...,S), where 0 is the
# absorbing state that represents system failure and S is the fault-free
# state;
# 2) if L_k denotes the set of states i in E with output rate
# ld[i]=ld[k] and F_k denotes the set of states i in E with reward rate
#      r[i]=r[k], then the following conditions hold:
# 2-1) (ld[i]-ld[j])/(ld[k]-ld[j]) is different from
# (r[i]-r[j])/(r[k]-r[j]) for all i,k,j in E such that  i is different
# from k, k is different from j, and neither i nor k belong to
# F_j or L_j;
# 2-2) ld[i] is different from ld[j] or r[i] is different from r[j] for
# all i,j in E such that i is different from j.
#
# IMPORTANT NOTE: it seems that the algorithm does not break down
# as long as there not exist an integer k>0 such that ldp=k*ldm or ldm=k*ldp.
#
# computes M_k^(s), 0<=s<=S, 0<=k<=s.
# global variables: the array ld[0]...ld[S] of output rates
comp_M:=proc(S) local s,k,i:global M,ld:
  M[0,0]:=0.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      if (ld[s]<>ld[k]) then
        M[k,s]:=M[k,s-1]:
      else
        M[k,s]:=M[k,s-1]+1:
      end if:
    end do:
    M[s,s]:=M[0,s]:
    for i from 1 to s-1 do M[k,s]:=max(M[k,s],M[i,s]) end do:
  end do:
end proc:

# computes W_k^(s), 0<=s<=S, 0<=k<=s. (This is recurrence B4 in the paper.)
# global variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_W:=proc(S) local s,k,j,val:global W,Q,ld,r:
  W[0,0]:=1.0:
  for s from 0 to S do
    for k from 0 to s-1 do
      W[k,s]:=0.0:
      if (r[k]=r[s]) then
        for j from k to s-1 do W[k,s]:=W[k,s]+Q[s,j]*W[k,j] end do:
        W[k,s]:=W[k,s]*ld[s]/(ld[s]-ld[k]):
        if (type(W[k,s],undefined)) then
          error "W[k=%1,s=%2] undefined (ld[s]=%3,ld[k]=%4)",k,s,ld[s],ld[k]:
        end if:
        if (W[k,s]=Float(infinity)) then
          error "W[k=%1,s=%2] overflows",k,s:
        end if:
      end if:
    end do:
    val:=0.0:
    for k from 0 to s-1 do val:=val+W[k,s] end do:
```

```
    W[s,s]:=1.0-val:
  end do:
end proc:

thet:=proc(a,b,c) local num,den,val:global ld,r:
  num:=r[a]-r[b]:
  den:=ld[c]*r[a]+ld[a]*r[b]+ld[b]*r[c]
      -(ld[c]*r[b]+ld[b]*r[a]+ld[a]*r[c]):
  val:=num/den:
  if (type(expr,undefined)) then
    error "undefined for a:%1,b=%2,c=%3",a,b,c:
  end if:
  if (val=Float(infinity)) then
    error "overflow for
a:%1,b:%2,c:%3:ld[a]=%4,ld[b]=%5,ld[c]=%6,r[a]=%7,r[b]=%8,r[c]=%9",a,b,c,ld[a],ld
[b],ld[c],r[a],r[b],r[c]:
  end if:
  return val:
end proc:

# computes H_k^(s)(i), 0<=s<=S, 0<=k<=s, 0<=i<=s, i not belonging to
# the reunion of sets F_k and L_k, and Z_k^(s)(i), 0<=s<=S, 0<=k<=s,
# 1<=i<=M_k^(s). (These are recurrences B1-B3.)
# blobal variables:
# the transition probability matrix of the embedded DTMC (_not_ the infinitesimal
# generator), Q[i,j]=ld[i,j]/ld[i], i different from j, 1<=i<=S,0<=j<=S
# the array ld[0] ... ld[S]
# the array r[0] ... r[S] of reward rates
comp_HZ:=proc(S) local s,k,i,j,p,val:global ld,r,Q,H,Z:
  for s from 0 to S do
    for k from 0 to s-1 do

#                         H_k^(s)(i), k=0,...,s-1, i=0,...,s-1,
# i\notin(F_k\cup L_k), Z_k^(s)(i), k=0,...,s-1,i=1,...,M_k^(s)

        for i from 0 to s-1 do
          if (ld[i]<>ld[k] and r[i]<>r[k]) then
            val:=0.0:
            for j from max(i,k) to s-1 do
              val:=val+Q[s,j]*H[k,j,i]
            end do:
            H[k,s,i]:=ld[s]*thet(i,k,s)*val
          end if:
          if (i>=1 and i<=M[k,s]) then
            val:=0.0:
            if (ld[s]<>ld[k] and r[s]<>r[k]) then
              for j from k to s-1 do
                for p from i to M[k,j] do

val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^(p-i))*Z[k,j,p]
              end do:
            end do:
          end if:
          Z[k,s,i]:=val*ld[s]/(ld[s]-ld[k]):
          val:=0.0:
          if (ld[s]=ld[k]) then
            for j from k to s-1 do
              if (i>=2 and i<=M[k,j]+1) then
                val:=val+Q[s,j]*Z[k,j,i-1]
              end if:
            end do:
            if (i=1) then
              for j from k to s-1 do
                val:=val+Q[s,j]*W[k,j]
              end do:
              for j from k to s-1 do
                for p from 0 to j do
                  if (ld[p]<>ld[k] and r[p]<>r[k]) then
```

```
                    val:=val+Q[s,j]*H[k,j,p]/(ld[p]-ld[k])
                  end if:
                end do:
              end do:
              Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(r[s]-r[k]):
              if (type(Z[k,s,i],undefined)) then
                error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
              end if:
              if (Z[k,s,i]=Float(infinity)) then
                error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
              end if:
            end if:
          end if:
          val:=0.0:
          if (r[s]=r[k]) then
            for j from k to s-1 do
              if (1<=i and i<=M[k,j]) then
                val:=val+Q[s,j]*Z[k,j,i]:
              end if:
            end do:
            Z[k,s,i]:=Z[k,s,i]+val*ld[s]/(ld[s]-ld[k]):
            if (type(Z[k,s,i],undefined)) then
              error "Z[k=%1,s=%2,i=%3] undefined",k,s,i:
            end if:
            if (Z[k,s,i]=Float(infinity)) then
              error "Z[k=%1,s=%2,i=%3] overflows",k,s,i:
            end if:
          end if:
        end if:
#

      end do:
#                           H_k^(s)(s), k=0,...,s-1, k\notin L_s\cup
# F_s

       if (ld[k]<>ld[s] and r[k]<>r[s]) then
         val:=0:
         for j from k to s-1 do
           for i from 0 to j do
             if (ld[i]<>ld[k] and r[i]<>r[k]) then
               val:=val+Q[s,j]*thet(s,k,i)*H[k,j,i]
             end if:
           end do:
         end do:
         for j from k to s-1 do
           for i from 1 to M[k,j] do
             val:=val+Q[s,j]*(((r[k]-r[s])/(ld[s]-ld[k]))^i)*Z[k,j,i]
           end do:
         end do:
         for j from k to s-1 do
           val:=val+Q[s,j]*W[k,j]
         end do:
         H[k,s,s]:=val*ld[s]
       end if:
     end do:
#                   H_s^(s)(k), k=0,...,s-1, k\notin L_s\cup F_s:
# Z_s^(s)(i), i=1,...,M_s^(s)

     for k from 0 to s-1 do
       H[s,s,k]:=H[k,s,s]
     end do:
     for i from 1 to M[s,s] do
       val:=0.0:
       for k from 0 to s-1 do
         if (i<=M[k,s]) then
           val:=val+Z[k,s,i]
         end if:
       end do:
```

```
        Z[s,s,i]:=-val
      end do:
    end do:
  end proc:

  # computes Prob{CR(t)<=y}, i.e. the probability that the reward accumulated
  # in the time interval [0,t] is smaller than or equal to y being S the initial
  # state
  # global variables:
  # the array ld[0] ... ld[S]
  # the array r[0] ... r[S] of reward rates
  # the array M filled in by comp_M()
  # the array W filled in by comp_W()
  # the arrays Z and H filled in by comp_HZ()
  CRD:=proc(tm,y,S) local crd,aux1,aux2,k,i:global H,ld,r,Z,M,W:
  crd:=0.0:
  for k from 0 to S do
    for i from 0 to S do
      if (r[i]<>r[k] and ld[i]<>ld[k]) then
        aux1:=H[k,S,i]*exp(-ld[k]*tm)/(ld[i]-ld[k]):
        if (y>r[k]*tm) then
           aux2:=1.0-exp(-((ld[i]-ld[k])/(r[i]-r[k]))*(y-r[k]*tm))
        else
          aux2:=0.0
        end if:
        crd:=crd+aux1*aux2:
      end if:
    end do:
  end do:
  for k from 0 to S do
    for i from 1 to M[k,S] do
      if (y>r[k]*tm) then
        crd:=crd+(Z[k,S,i]/i!)*exp(-ld[k]*tm)*((y-r[k]*tm)^i)
      end if:
    end do:
  end do:
  for k from 0 to S do
    if (y>r[k]*tm) then
      crd:=crd+W[k,S]*exp(-ld[S]*tm):
    end if:
  end do:
  return crd:
end proc:

# Implements the multiprocessor system described in V. Grassi, L.
# Donatiello, and G. Iazeolla, "Performability Evaluation of a
# Multicomponent Fault-Tolerant System," IEEE Trans. on Reliability,
# 37(2): 216-222, June 1988. The system consists of P processors and M
# memories that fail with rate ldp and ldm, respectively. The system is
# up if there are at least one fault-free processor and one fault-free
# memory. States are (conceptually) labelled (p,m), where p and m are
# the no. of fault-free processors and memories, respectively. State
# (p,m) is assigned the reward rate m*(1-(1-1/m)^p)/T, which is an
# estimate of the steady-state average number of active processors in
# [0,t].
# returns the number of states
count_st:=proc(P,M) local idx,p,m:
  idx:=0:
  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
    end do:
  end do:
  return (idx+1):
end proc:

# assigns indices to states and bookkeeps them in the global array
# state_idx[p,m], 1<=p<=P, 1<=m<=M (State (p,m) has index state_idx[p,m].)
```

```
#
# Note: for convenience, state_idx[0,m], 1<=m<=M and state_idx[p,0], 1<=p<=P
# are all set to 0
set_stidx:=proc(P,M,S) local idx,p,m:global state_idx:
  idx:=0:
  for p from 1 to P do
    for m from 1 to M do
      idx:=idx+1:
      if (idx>S) then
        error "state index (%1) larger than the expected maximum one (%2)",idx,S:
      end if:
      state_idx[p,m]:=idx:
    end do:
  end do:
  for p from 1 to P do state_idx[p,0]:=0 end do:
  for m from 1 to M do state_idx[0,m]:=0 end do:
end proc:

# returns the reward rate for state (p,m)
strwd:=proc(p,m) local val:global state_idx:
  if (p>0 and m>0 and (p+m)>2) then
    val:=1.0:
  else
    val:=0.0:
  end if:
  return val:
end proc:

# computes the states' reward rate
comp_r:=proc(P,M) local p,m,idx:global state_idx,r:
  for p from 1 to P do
    for m from 1 to M do
      idx:=state_idx[p,m]:
      r[idx]:=strwd(p,m):
    end do:
  end do:
  r[0]:=0.0:
end proc:

# computes the states' initial probability, which is 0 for them all except
# that state (P,M) is assigned initial probability P_0, and both state
# (P-1,M) and (P,M-1) are assigned initial probability (1-P_0)/2
comp_initp:=proc(P,M,P_0) local p,m:global init_p,state_idx:
  if (P_0<0.0 or P_0>1) then
    error "wrong value for P_0 (%1)", P_0:
  end if:
  for p from 1 to P do
    for m from 1 to M do
      init_p[state_idx[p,m]]:=0.0:
    end do:
  end do:
  init_p[0]:=0.0:
  init_p[state_idx[P,M]]:=P_0:
  init_p[state_idx[P-1,M]]:=(1-P_0)/2.0:
  init_p[state_idx[P,M-1]]:=(1-P_0)/2.0:
end proc:

# computes the output rates and transition probabilities (arrays ld[]
# and Q[], respectively)
comp_ld_Q:=proc(P,M,S,ldp,ldm) local p,m,i,j,val:global
state_idx,ld,Q:
  for i from 0 to S do
    ld[i]:=0.0:
    for j from 0 to S do
      Q[i,j]:=0.0:
    end do:
  end do:
  for p from 1 to P do
```

```
    for m from 1 to M do
      i:=state_idx[p,m]:

      ld[i]:=p*ldp+m*ldm:
      if (state_idx[p-1,m]<>state_idx[p,m-1]) then
        j:=state_idx[p-1,m]:
        Q[i,j]:=p*ldp/ld[i]:
        j:=state_idx[p,m-1]:
        Q[i,j]:=m*ldm/ld[i]:
      else
        j:=state_idx[p-1,m]:
        Q[i,j]:=1.0:
      end if:
    end do:
  end do:
end proc:
Digits:=70:
ldp:=2e-4:
ldm:=4.5e-4:
T:=1000:
P:=3:
M:=3:
P_0:=0.75:
S:=count_st(P,M)-1:
state_idx:=array(0..P,0..M):
r:=array(0..S):
ld:=array(0..S):
init_p:=array(0..S):
Q:=array(0..S,0..S):
set_stidx(P,M,S):
comp_r(P,M):
comp_ld_Q(P,M,S,ldp,ldm):
comp_initp(P,M,P_0):
M:=array(0..S,0..S):
comp_M(S):
W:=array(0..S,0..S):
comp_W(S):
H:=array(0..S,0..S,0..S):
dim_Z:=1:
for s from 0 to S do
  for k from 0 to s-1 do
    dim_Z:=max(dim_Z,M[k,s]):
  end do:
end do:
Z:=array(0..S,0..S,0..dim_Z):
comp_HZ(S):
n_absc:=7:
absc:=vector([.85,.95,.99,.999,.9999,.99999,.999999]):
for j from 1 to n_absc do
  y:=absc[j]*T:
  CCRD:=0.0:
  for i from 0 to S do
    if (init_p[i]>0.0) then
      CCRD:=CCRD+init_p[i]*CRD(T,y,i):
    end if:
  end do:
  CCRD:=1.0-CCRD:
  printf("%2.28g\n", CCRD):
end do:
#
Digits:=30:
printf("%2.28g\n", 0.0):
printf("%2.28g\n", 1.0):
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
```

```
f:=Int((exp(-mu*y)/sqrt(y))*BesselI(1,2*sqrt(la*mu*p*t*y)),y=0..t*(1-p)):
ciavd:=exp(-la*t*p)*(1+f*sqrt(la*mu*p*t)):
tv:=vector([10]):
nt:=1:
pv:=vector([0.1,0.5,0.9,0.99,0.999]):
np:=5:
for i from 1 to nt do
  for j from 1 to np do
    t:=tv[i]:
    p:=pv[j]:
    printf("%2.28g\n", evalf(ciavd)):
  end do:
end do:
quit:
#
#
# Model parameter values: stat_start=1,ND0=10,la=1e-4,mu=1,P0=0.9
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using Rubino & Sericola A algorithm with abs. err. 1e-12
np:=5:
ciavd:=vector([9.9998763679889746125 4e-01,
               9.9932148507254836555 8e-01,
               9.6288097185429100122 2e-01,
               9.0870145343785058145 2e-01,
               9.0010431429122261803 9e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
#
# Model parameter values: NDO_INIT=10,ND1_INIT=10,LA=1E-4,MU=0.5,PO=0.5
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using ETRR with abs. err. 1e-12
np:=5:
ciavd:=vector([4.9959957441489000196 1e-01,
               4.9760623853784191617 0e-01,
               4.9562018570640864467 7e-01,
               4.9517441735423639581 6e-01,
               4.9512986257603180950 7e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
#
# Model parameter values: NDO=10,la=1E-4,mu=0.5,PO=0.5
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using ETRR with abs. err. 1e-12
np:=5:
ciavd:=vector([7.4994986003756847203 8e-01,
               7.4994926019126595395 6e-01,
               7.4811521539642000799 8e-01,
               5.9458167203334333361 7e-01,
               5.0734485861374500892 4e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
#
# Model parameter values: NDO=10,la=1E-4,mu=1,C=0.95,PO=0.5
```

```
#
# Abscissae: 0.1, 0.5, 0.9, 0.99, 0.999
#

# Values computed using A algorithm with abs. err. 1E-12
np:=5:
ciavd:=vector([7.4962512579928974876 3e-01,
               7.4812751605190963566 4e-01,
               7.4663289824668310767 9e-01,
               7.4629702095633199121 5e-01,
               7.4626344153872392617 2e-01]):
for i from 1 to np do
  printf("%2.28g\n", ciavd[i]):
end do:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-3:
mu:=1:
C:=0.99:
k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
p:=vector([0,0,0,0]):
p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
for i from 2 to 4 do
  p[i]:=p[1]*k[i]:
end do:
q:=vector([0,0]):
q[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
q[2]:=1-q[1]:
A:=matrix([[s+2*ld,-mu,0],[-2*ld*C,s+mu+ld,0],[-2*ld*(1-C),-ld,s]]):
b:=vector([q[1],q[2],0]):
trf_sol:=linsolve(A,b):
actual_sol:=simplify(invlaplace(trf_sol[3],s,t)):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-3:
mu:=1:
C:=0.99:
k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
p:=vector([0,0,0,0]):
p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
for i from 2 to 4 do
  p[i]:=p[1]*k[i]:
end do:
q:=vector([0,0]):
q[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
q[2]:=1-q[1]:
A:=matrix([[s+ld,-mu,0],[-ld*C,s+mu+ld,0],[-ld*(1-C),-ld,s]]):
b:=vector([q[1],q[2],0]):
trf_sol:=linsolve(A,b):
actual_sol:=simplify(invlaplace(trf_sol[3],s,t)):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
```

```
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-3:
mu:=1:
C:=0.99:
k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
p:=vector([0,0,0,0]):
p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
for i from 2 to 4 do
  p[i]:=p[1]*k[i]:
end do:
#for i from 1 to 4 do
#  printf("p[%d]=%2.28g\n",i,p[i]):
#end do:
q:=vector([0,0]):
q[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
q[2]:=1-q[1]:
#for i from 1 to 2 do
#  printf("q[%d]=%2.28g\n",i,q[i]):
#end do:
A:=matrix([[s+2*ld*(1-C)+2*ld*C*ld/(ld+mu),0],[-(2*ld*(1-
C)+2*ld*C*ld/(ld+mu)),s]]):
b:=vector([q[1]+mu*q[2]/(ld+mu),ld*q[2]/(ld+mu)]):
#for i from 1 to 2 do
#  printf("new_q[%d]=%2.28g\n",i,b[i]):
#end do:
trf_sol:=linsolve(A,b):
actual_sol:=simplify(invlaplace(trf_sol[2],s,t)):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-3:
mu:=1:
C:=0.99:
k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
p:=vector([0,0,0,0]):
p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
for i from 2 to 4 do
  p[i]:=p[1]*k[i]:
end do:
q:=vector([0,0]):
q[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
q[2]:=1-q[1]:
A:=matrix([[s+2*ld,-mu,0],[-2*ld*C,s+mu+ld,0],[-2*ld*(1-C),-ld,s]]):
b:=vector([q[1],q[2],0]):
trf_sol:=linsolve(A,b):
actual_sol:=simplify(invlaplace(trf_sol[3],s,t)):
n_absc:=3:
absc:=vector([0.1,10,1000]):
```

```
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
n_absc:=3:
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 1):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
p0:=0.1:
ld_a:=1e-4:
ld_b:=2e-4:
ld_c:=3e-4:
ld_d:=4e-4:
ld_lr:=5e-4:
ld_udl:=6e-4:
ld_udr:=7e-4:
mu_a:=0.25:
mu_b:=0.5:
A:=matrix([[-ld_a,ld_a,0,0],[mu_a,-(mu_a+ld_lr+ld_udl),ld_lr,0],[0,0,-
ld_b,ld_b],[0,0,mu_b,-(mu_b+ld_udr)]]):
b:=vector([-p0,0,-2*(1-p0)/5,0]):
tau:=linsolve(transpose(A),b):
crdds:=(2*(1-p0)/5+ld_udl*tau[2])*(1-exp(-ld_c*t))+((1-p0)/5+tau[4]*ld_udr)*(1-
exp(-ld_d*t)):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(crdds)):
end do:
#
Digits:=30:
with(linalg):
with(inttrans):
p0:=0.1:
ld_a:=1e-4:
ld_b:=2e-4:
ld_c:=3e-4:
ld_d:=4e-4:
ld_lr:=5e-4:
ld_udl:=6e-4:
ld_udr:=7e-4:
mu_a:=0.25:
mu_b:=0.5:
A:=matrix([[-ld_a,ld_a,0,0],[mu_a,-(mu_a+ld_lr+ld_udl),ld_lr,0],[0,0,-
ld_b,ld_b],[0,0,mu_b,-(mu_b+ld_udr)]]):
b:=vector([-p0,0,-2*(1-p0)/5,0]):
tau:=linsolve(transpose(A),b):
crdds:=(2*(1-p0)/5+ld_udl*tau[2])*(1-exp(-ld_c*t))+(1-p0)/5+tau[4]*ld_udr:
n_absc:=3:
absc:=vector([0.1,10,1000]):
```

```
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(crdds)):
end do:
#
Digits:=30:
n_absc:=3:
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 1.0):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu],[-2*ld*(1-C),s,-ld],[-2*ld*C,0,s+mu+ld]]):
b:=vector([x,0,1-x]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu],[-2*ld*(1-C),s,-ld],[-2*ld*C,0,s+mu+ld]]):
b:=vector([x,0,1-x]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu],[-2*ld*(1-C),s,-ld],[-2*ld*C,0,s+mu+ld]]):
```

```
b:=vector([x,0,1-x]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+ld,0,-mu],[-ld*(1-C),s,-ld],[-ld*C,0,s+mu+ld]]):
b:=vector([x,0,1-x]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+ld*(1-C)+ld*C*ld/(ld+mu),0],[-(ld*(1-C)+ld*C*ld/(ld+mu)),s]]):
b:=vector([x+mu*(1-x)/(ld+mu),(1-x)*ld/(ld+mu)]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=1.0:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  t:=absc[i]:
  printf("%2.28g\n", evalf(actual_sol)):
end do:
quit:
#
Digits:=30:
n_absc:=3:
#
```

```
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 1.0):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,-mu,-mu,0],[-2*ld*(1-C),mu+s,0,0],[-2*ld*C,0,s+mu+ld,-
mu],[0,0,-ld,s+mu]]):
b:=vector([x,0,1-x,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", (1/absc[i])*evalf(int(actual_sol,t=0..absc[i]))):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu,0],[-2*ld*(1-C),s,0,0],[-2*ld*C,0,s+mu+ld,0],[0,0,-
ld,s]]):
b:=vector([p0,0,1-p0,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=1e8*(invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t
)):
C:=0.99:
p0:=0.9:
np:=1:
t_arr:=vector([0.01]):
ld_arr:=vector([4e-7]):
mu_arr:=vector([1e-7]):
for i from 1 to np do
  ld:=ld_arr[i]:
  mu:=mu_arr[i]:
  printf("%2.28g\n",
(1.0/t_arr[i])*evalf(Int(actual_sym_sol,t=0..t_arr[i],digits=16,method=_Dexp))):
end do:
quit:
#
Digits:=30:
la:=1e-4:
mu:=1:
t:=1000:
aua:=la/(la+mu)-(la/(t*(la+mu)^2))*(1-exp(-(la+mu)*t)):
printf("%2.28g\n", aua):
quit:
Digits:=30:
ld:=1e-3:
mu:=1:
t:=1000:
earr:=ld/(ld+mu)-(ld/((ld+mu)*(ld+mu)*t))*(1-exp(-t*(ld+mu))):
printf("%2.28g\n", earr):
quit:
Digits:=30:
```

```
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu,0],[-2*ld*(1-C),s,0,0],[-2*ld*C,0,s+mu+ld,0],[0,0,-
ld,s]]):
b:=vector([p0,0,1-p0,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[1],s,t)+invlaplace(trf_sym_sol[3],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
p0:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", (1/absc[i])*evalf(int(actual_sol,t=0..absc[i]))):

end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu,0],[-2*ld*(1-C),s,0,0],[-2*ld*C,0,s+mu+ld,0],[0,0,-
ld,s]]):
b:=vector([x,0,1-x,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", (1/absc[i])*evalf(int(actual_sol,t=0..absc[i]))):
end do:
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,-mu,-mu,0],[-2*ld*(1-C),mu+s,0,0],[-2*ld*C,0,s+mu+ld,-
mu],[0,0,-ld,s+mu]]):
b:=vector([x,0,1-x,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=0.9:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", (1/absc[i])*evalf(int(actual_sol,t=0..absc[i]))):
end do:
quit:
```

```
#
Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,-mu,-mu,0],[-2*ld*(1-C),mu+s,0,0],[-2*ld*C,0,s+mu+ld,-
mu],[0,0,-ld,s+mu]]):
b:=vector([x,0,1-x,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
ld:=1e-3:
mu:=1:
C:=0.99:
x:=1:
actual_sol:=evalf(actual_sym_sol):
n_absc:=3:
absc:=vector([0.1,10,1000]):
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", (1/absc[i])*evalf(int(actual_sol,t=0..absc[i]))):
end do:
quit:
#
Digits:=40:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,-mu,-mu,0],[-2*ld*(1-C),mu+s,0,0],[-2*ld*C,0,s+mu+ld,-
mu],[0,0,-ld,s+mu]]):
b:=vector([p0,0,1-p0,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
C:=0.99:
p0:=0.9:
n_t:=13:
t_arr:=vector([500,1.998000999,0.09,10,100,10000,50000,500,1000,1000,100,1000,500
0]):
ld_arr:=([1,1e-3,1e-3,1e-3,1e-3,1e-3,1e-3,1e-3,0.01,1,1e-3,1e-3,1e-3]):
mu_arr:=vector([1e-3,1,10,1,1,1,1,1e-3,1e-3,1e-3,1,1,1]):
rwd_arr:=vector([1,1,1,1,1,1,1,1,1,.1,.1,.1]):
#
# grid: list of values
#
for i from 1 to n_t do
  ld:=ld_arr[i]:
  mu:=mu_arr[i]:
  printf("%2.28g\n",
(rwd_arr[i]/t_arr[i])*evalf(int(actual_sym_sol,t=0..t_arr[i]))):
end do:
quit:
#
Digits:=30:
n_absc:=3:
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 5.0):
end do:
quit:
#
Digits:=30:
n_absc:=3:
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 0.0):
```

```
end do:
quit:
#
Digits:=30:
n_absc:=3:
#
# grid: list of values
#
for i from 1 to n_absc do
  printf("%2.28g\n", 0.0):
end do:
quit:
#
#Digits:=30:
with(linalg):
with(inttrans):
A:=matrix([[s+2*ld,0,-mu,0],[-2*ld*(1-C),s,0,0],[-2*ld*C,0,s+mu+ld,0],[0,0,-
ld,s]]):
b:=vector([p0,0,1-p0,0]):
trf_sym_sol:=linsolve(A,b):
actual_sym_sol:=invlaplace(trf_sym_sol[2],s,t)+invlaplace(trf_sym_sol[4],s,t):
C:=0.99:
p0:=0.9:
np:=1:
t_arr:=vector([1000]):
ld_arr:=vector([0.001]):
mu_arr:=vector([0.058]):
for i from 1 to np do
  ld:=ld_arr[i]:
  mu:=mu_arr[i]:
  printf("%2.28g\n",
(1.0/t_arr[i])*evalf(Int(actual_sym_sol,t=0..t_arr[i],digits=16,method=_Dexp))):
end do:
quit:
#
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
#k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
#p:=vector([0,0,0,0]):
#p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
#for i from 2 to 4 do
#  p[i]:=p[1]*k[i]:
#end do:
#for i from 1 to 4 do
#  printf("p[%d]=%2.28g\n",i,p[i]):
#end do:
#alph:=vector([0,0]):
#alph[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
#alph[2]:=1-alph[1]:
#for i from 1 to 2 do
#  printf("alph[%d]=%2.28g\n",i,alph[i]):
#end do:
#tau:=vector([0,0]):
#tau[1]:=(alph[1]*ld+mu)/(2*ld*(ld+mu*(1-C))):
#tau[2]:=(2*ld*(1-alph[1]*(1-C)))/(2*ld*(ld+mu*(1-C))):
#for i from 1 to 2 do
#  printf("tau[%d]=%2.28g\n",i,tau[i]):
#end do:
#ecrds:=tau[1]+tau[2]:
#printf("%2.28g\n", ecrds):
ecrds:=(2*ld+mu+ld*(2*C-1)*(1-C)/(1-C+ld*C/mu))/(2*ld*(ld+mu*(1-C))):
printf("%2.28g\n", ecrds):
quit:
Digits:=30:
ld:=1e-4:
mu:=1:
```

```
C:=0.999:
#k:=vector([1,2*ld*C/mu,2*ld*ld*C/(mu*mu),2*ld*(1-C)/mu]):
#p:=vector([0,0,0,0]):
#p[1]:=1/(k[1]+k[2]+k[3]+k[4]):
#for i from 2 to 4 do
#  p[i]:=p[1]*k[i]:
#end do:
#for i from 1 to 4 do
#  printf("p[%d]=%2.28g\n",i,p[i]):
#end do:
#alph:=vector([0,0]):
#alph[1]:=p[4]*mu/(p[3]*mu+p[4]*mu):
#alph[2]:=1-alph[1]:
#for i from 1 to 2 do
#  printf("alph[%d]=%2.28g\n",i,alph[i]):
#end do:
#tau:=vector([0,0]):
#tau[1]:=(alph[1]*ld+mu)/(2*ld*(ld+mu*(1-C))):
#tau[2]:=(2*ld*(1-alph[1]*(1-C)))/(2*ld*(ld+mu*(1-C))):
#for i from 1 to 2 do
#  printf("tau[%d]=%2.28g\n",i,tau[i]):
#end do:
#ecrds:=tau[1]+tau[2]:
#printf("%2.28g\n", ecrds):
ecrds:=(2*ld+mu+ld*(2*C-1)*(1-C)/(1-C+ld*C/mu))/(2*ld*(ld+mu*(1-C))):
printf("%2.28g\n", ecrds):
quit:
Digits:=30:
with(linalg):
with(inttrans):
p0:=0.1:
ld_a:=1e-4:
ld_b:=2e-4:
ld_c:=3e-4:
ld_d:=4e-4:
ld_lr:=5e-4:
ld_udl:=6e-4:
ld_udr:=7e-4:
mu_a:=0.25:
mu_b:=0.5:
A:=matrix([[-ld_a,ld_a,0,0],[mu_a,-(mu_a+ld_lr+ld_udl),ld_lr,0],[0,0,-
ld_b,ld_b],[0,0,mu_b,-(mu_b+ld_udr)]]):
b:=vector([-p0,0,-2*(1-p0)/5,0]):
tau:=linsolve(transpose(A),b):
ecrds:=(1/ld_c)*(2*(1-p0)/5+ld_udl*tau[2])+(1/ld_d)*((1-p0)/5+tau[4]*ld_udr):
printf("%2.28g\n", ecrds):
quit:
#
Digits:=30:
ld:=1e-4:
ecrds:=1/(2*ld):
printf("%2.28g\n", ecrds):
quit:
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
ecrte:=((ld+mu)/(2*ld)+C)/(ld+mu*(1-C)):
printf("%2.28g\n", ecrte):
quit:
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-4:
mu:=1:
C:=0.999:
p0:=0.75:
A:=matrix([[-2*ld,2*ld*C],[mu,-(mu+ld)]]):
```

```
b:=vector([-p0,-(1-p0)]):
tau:=linsolve(transpose(A),b):
ecrte:=tau[1]+tau[2]:
printf("%2.28g\n", ecrte):
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-4:
mu:=1:
C:=0.999:
p0:=0.75:
A:=matrix([[-2*ld,2*ld*C],[mu,-(mu+ld)]]):
b:=vector([-p0,-(1-p0)]):
tau:=linsolve(transpose(A),b):
ecrte:=tau[1]+tau[2]:
printf("%2.28g\n", ecrte):
quit:
#
Digits:=30:
with(linalg):
with(inttrans):
ld:=1e-4:
mu:=1:
C:=0.999:
p0:=1.0:
A:=matrix([[-2*ld,2*ld*C],[mu,-(mu+ld)]]):
b:=vector([-p0,-(1-p0)]):
tau:=linsolve(transpose(A),b):
ecrte:=tau[1]+tau[2]:
printf("%2.28g\n", ecrte):
quit:
#
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
ecrte:=((ld+mu)/(2*ld)+C)/(ld+mu*(1-C)):
printf("%2.28g\n", ecrte):
quit:
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
ecrte:=((ld+mu)/(2*ld)+C)/(ld+mu*(1-C)):
printf("%2.28g\n", ecrte):
quit:
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
ua:=(2*ld/mu)*(1-C+C*ld/mu)/(1+(2*ld/mu)*(1+C*ld/mu)):
printf("%2.28g\n", ua):
quit:
Digits:=30:
ld:=1e-4:
mu:=1:
C:=0.999:
ua:=(2*ld/mu)*(1-C+C*ld/mu)/(1+(2*ld/mu)*(1+C*ld/mu)):
printf("%2.28g\n", ua):
quit:
exp_T:=proc(C,lambda,mu) local rho, onemrho, ploss, gamma, p, ET, n:
  rho:=lambda/mu:
  onemrho:=1.0-rho:
  ploss:=onemrho/(rho^(-C)-rho):
  gamma:=lambda*(1.0-ploss):
  p:=ploss:
```

```
  ET:=0.0:
  for n from C by -1 to 1 while p>0.0
    do
      ET:=ET+n*p:
      p:=p/rho:
    end do:
  ET:=ET/gamma:
  return ET:
end proc:
#
Digits:=30:
#
C:=100:
lambda:=1.01:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=2000:
lambda:=5:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=100:
lambda:=1.001:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=1.00001:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=5:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=2:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=1.5:
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=1.5:
mu:=1.2:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=500:
lambda:=1.5:
mu:=1.44:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
C:=700:
lambda:=2:
```

```
mu:=1:
ET:=exp_T(C,lambda,mu):
printf("%2.28g\n", ET):
#
quit:
#
Digits:=30:
printf("%2.28g\n", 5.0):
```

A.4. Fitxers d'execució de models

```
/*
 * Driver for the CIAVD_5 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {stat_start = 1} X {time=10} X abscissae=list of values: (0.1,
 * 0.5, 0.9, 0.99, 0.999) X {meth: 2} X {err: 1e-6}
 */



#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

```
#include "driver_lib.h"

#define MODELNAME "CIAVD_5"
#define MEASURENAME "IACD"
#define ND0 1
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 5


int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int it,ii,n_it,codi_mesura;
    int n_absc;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {2},
      stat_start[]={1};
    float tol[]= {1e-6},
      timer[] = {10};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    n_it = 1;
    for (it = 0; it < n_it; it++) {
      if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
      build_parset(fp, parset, "%d\n%d\n%g\n%g\n",
                   stat_start[it], ND0, LAMBDA, MU);
      fprintf (fp, "1\n");
      fprintf (fp, "%d\n",codi_mesura);
      fprintf (fp, "%d\n", methods[it]);
      fprintf (fp, "%g\n",timer[it]);
      fprintf (fp, "%g\n", tol[it]);
      fprintf (fp, "%d\n", n_absc);
      if (n_absc > 1) fprintf (fp, "s\n");
      for (ii = 1; ii <= n_absc; ii++) {
          (void) fprintf(fp, "%g", absc[ii]);
          if (ii < n_absc) (void) fputc(',', fp);
          else (void) fputc('\n', fp);
```

```
      }
      fprintf (fp, "y\n");
      if (fclose (fp) != 0) goto on_error;

      /* run */
      if (system(modelexe)) goto on_error;

      /* read the computed value of the measure */
      if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                        mesura) != 0) goto on_error;

      /* log file: print vector identity */
      print_ident_vector_one_meth(MODELNAME, parset,
                           MEASURENAME, methods[it],
                           tol[it], logfile);

      /* log file: print results about the correctness
       * of the computed value of the measure */
      (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
      for (ii = 1; ii <= n_absc; ii++) {
          if (check_result_CIAVD(methods[it], tol[it], true_meas[ii],
                           mesura[ii], logfile) != 0) {
            goto on_error;
          }
      }
      (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

      /* log file: print results about the test of the CTMC */
      if (check_Markov_chain(MODELNAME, logfile) != 0) goto on_error;

      /* log file: print results about memory checking */
      if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                  DRIVER_LOG_END_MEMRES) != 0) goto on_error;
    }
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CCRD_2 model.
 * measure: CRCD.
 * available numerical methods:
 * 1 (Nabli & Sericola).
 * {time: 10} X abscissae=list of values: (0.999)*time X {meth: 1} X
{err:
 * 1e-6, 1e-9, 1e-12}
 */
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CCRD_2"
#define MEASURENAME "CRCD"
```

```
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 1



int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1};
    float tol[]= {1e-6,1e-9,1e-12},
      timer[] = {10};
    double absc[] = {0.0, 0.999};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%g\n%g\n",
                         LAMBDA, MU);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]*timer[m]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
```

```
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j], true_meas[ii],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }}
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                            DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CCRD_3 model.
 *
 * measure: CRCD.
 *
 * available numerical methods:
 *
 * 1 (Nabli & Sericola).
 *
 * {time=1000} X abscissae=list of values: (1e-6, 0.8, 1.0, 1.3, 1.5,
1.55,
 * 1.667, 1.7, 1.75, 2.0, 2.111, 2.2) X {meth: 1} X {err: 1e-6, 1e-9,
 * 1e-12}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <string.h>

#include "driver_lib.h"

#define MODELNAME "CCRD_3"
#define MEASURENAME "CRCD"
#define NP 3
#define NM 3
#define T 1000.0
#define P0 0.75
#define LDP 0.6e-4
#define LDM 1e-4
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 12

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1};
    float tol[]= {1e-6,1e-9,1e-12},
      timer[] = {1000};
    double absc[] = {0.0,1e-
6,.8,1,1.3,1.5,1.55,1.667,1.7,1.75,2,2.111,2.2};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<n_methods;i++){
      for (m=0;m<n_timer;m++){
          for (j=0;j<n_tol;j++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n%g\n",
                       NP, NM, T, P0, LDP, LDM);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
```

```
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j], true_meas[ii],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                        DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }
        }
    }
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}
/*
 * Driver for the CCRD_4 model.
 *
 * measure: CRCD.
 *
 * available numerical methods:
 *
 * 1 (Nabli & Sericola).
```

```
 *
 * {time=1000} X abscissae=list of values: (0.1, 0.95) X {meth: 1} X
{err: 1e-6}
 *
 * {time=1000} X abscissae=list of values: (1.8, 1.9) X {meth: 1} X {err:
1e-6}
 *
 * {time=1000} X abscissae=list of values: (0.1, 2.5) X {meth: 1} X {err:
1e-6}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CCRD_4"
#define MEASURENAME "CRCD"
#define NP 3
#define NM 3
#define T 1000.0
#define P0 0.75
#define LDP 0.6e-4
#define LDM 1e-4
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 6

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer, n_exec;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1};
    float tol[]= {1e-6},
      timer[] = {1000};
    double absc[] = {0.0, 0.1,0.95,1.8,1.9,0.1,2.5};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = 2;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;
```

```
    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    n_exec=1;
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n%g\n",
                        NP, NM, T, P0, LDP, LDM);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1+(n_exec-1)*n_absc; ii <= n_exec*n_absc; ii++) {
                fprintf(fp, "%g", absc[ii]);
                if (ii < n_exec*n_absc) fputc(' ', fp);
                else fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j],
true_meas[ii+(n_exec-1)*n_absc],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                        DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }
      }
    }
    ++n_exec;
```

```
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n%g\n",
                         NP, NM, T, P0, LDP, LDM);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);

            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1+(n_exec-1)*n_absc; ii <= n_exec*n_absc; ii++) {
                fprintf(fp, "%g", absc[ii]);
                if (ii < n_exec*n_absc) fputc(' ', fp);
                else fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j],
true_meas[ii+(n_exec-1)*n_absc],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                            DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }
      }
    }
    ++n_exec;
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
```

```
            build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n%g\n",
                       NP, NM, T, P0, LDP, LDM);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1+(n_exec-1)*n_absc; ii <= n_exec*n_absc; ii++) {
                fprintf(fp, "%g", absc[ii]);
                if (ii < n_exec*n_absc) fputc(' ', fp);
                else fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j],
true_meas[ii+(n_exec-1)*n_absc],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                            DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }
        }
    }
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
```

```
 * Driver for the CCRD_5 model.
 *
 * measure: CCRD.
 *
 * available numerical methods:
 *
 * 1 (Nabli & Sericola).
 *
 * {time=10} X {abscissae=list of values: 0.1, 0.5, 0.9, 0.99, 0.999,
1.1} X
 * {meth: 1} X {err: 1e-6}
 *
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CCRD_5"
#define MEASURENAME "CRCD"
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 6

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1};
    float tol[]= {1e-6},
      timer[] = {10};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999, 1.1};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
```

```
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%g\n%g\n", LAMBDA, MU);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]*timer[m]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CCRD(methods[i], tol[j],
                                      true_meas[ii+m*n_absc],
                                      mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                        DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
```

```
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_1 model.
 *
 * measure: IACD.
 *
 * available numerical methods:
 *
 * 1 (A), 2 (A_TRF).
 *
 * {stat_start = 1} X {time=0.1, 10, 100} X abscissae=list of values:
(0.1,
 * 0.5, 0.9, 0.99, 0.999) X {meth: 1} X {err: 1e-6, 1e-9, 1e-12}
 *
 * {stat_start = 0, 1} X {time=0.1, 10, 100} X abscissae=list of values:
(0.1,
 * 0.5, 0.9, 0.99, 0.999) X {meth: 2} X {err: 1e-6, 1e-9, 1e-12}
 *
 * {stat_start = 0} X {time=100} X abscissae=list of values: (0.1, 0.5) X
 * {meth: 1} X {err: 1e-6} (NOTE: this tests triggers calling
trunc_poiss()
 * with mode=2.)
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "driver_lib.h"

#define MODELNAME "CIAVD_1"
#define MEASURENAME "IACD"
#define ND0 1
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 15


int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,k,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_stat_start, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {1,2},
      stat_start[]={0,1};;
    float tol[]= {1e-6,1e-9,1e-12},
      timer[] = {0.1,10,100};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_stat_start = sizeof(stat_start)/sizeof(int);
```

```
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<1;i++){
      for (j=0;j<n_tol;j++){
          for (k=0;k<1; k++) {
            for (m=0;m<n_timer;m++){
                if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
                build_parset(fp, parset, "%d\n%d\n%g\n%g\n",
                             stat_start[k], ND0, LAMBDA, MU);
                fprintf (fp, "1\n");
                fprintf (fp, "%d\n",codi_mesura);
                fprintf (fp, "%d\n", methods[i]);
                fprintf (fp, "%g\n",timer[m]);
                fprintf (fp, "%g\n", tol[j]);
                fprintf (fp, "%d\n", n_absc);
                if (n_absc > 1) fprintf (fp, "s\n");
                for (ii = 1; ii <= n_absc; ii++) {
                  (void) fprintf(fp, "%g", absc[ii]);
                  if (ii < n_absc) (void) fputc(',', fp);
                  else (void) fputc('\n', fp);
                }
                fprintf (fp, "y\n");
                if (fclose (fp) != 0) goto on_error;

                /* run */
                if (system(modelexe)) goto on_error;

                /* read the computed value of the measure */
                if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ",
n_absc,
                                       mesura) != 0) goto on_error;

                /* log file: print vector identity */
                print_ident_vector_one_meth(MODELNAME, parset,
                                            MEASURENAME, methods[i],
                                            tol[j], logfile);

                /* log file: print results about the correctness
                 * of the computed value of the measure */
                (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
                for (ii = 1; ii <= n_absc; ii++) {
                  if (check_result_CIAVD(methods[i], tol[j],
true_meas[ii+m*n_absc],
```

```
                                        mesura[ii], logfile) != 0) {
                        goto on_error;
                    }}
                (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

                /* log file: print results about memory checking */
                if (append_file(logfile, VALGRIND_OUT,
DRIVER_LOG_BEG_MEMRES,
                                DRIVER_LOG_END_MEMRES) != 0) goto on_error;
            }}}}
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (k=0;k<n_stat_start; k++) {
            for (m=0;m<n_timer;m++){
                if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
                build_parset(fp, parset, "%d\n%d\n%g\n%g\n",
                         stat_start[k], ND0, LAMBDA, MU);
                fprintf (fp, "1\n");
                fprintf (fp, "%d\n",codi_mesura);
                fprintf (fp, "%d\n", methods[i]);
                fprintf (fp, "%g\n",timer[m]);
                fprintf (fp, "%g\n", tol[j]);
                fprintf (fp, "%d\n", n_absc);
                if (n_absc > 1) fprintf (fp, "s\n");
                for (ii = 1; ii <= n_absc; ii++) {
                  (void) fprintf(fp, "%g", absc[ii]);
                  if (ii < n_absc) (void) fputc(',', fp);
                  else (void) fputc('\n', fp);
                }
                fprintf (fp, "y\n");
                if (fclose (fp) != 0) goto on_error;

                /* run */
                if (system(modelexe)) goto on_error;

                /* read the computed value of the measure */
                if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ",
n_absc,
                                       mesura) != 0) goto on_error;

                /* log file: print vector identity */
                print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

                /* log file: print results about the correctness
                 * of the computed value of the measure */
                (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
                for (ii = 1; ii <= n_absc; ii++) {
                  if (check_result_CIAVD(methods[i], tol[j],
true_meas[ii+m*n_absc],
                                        mesura[ii], logfile) != 0) {
                        goto on_error;
                  }}
                (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
```

```
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

                /* log file: print results about memory checking */
                if (append_file(logfile, VALGRIND_OUT,
DRIVER_LOG_BEG_MEMRES,
                                DRIVER_LOG_END_MEMRES) != 0) goto on_error;
            }}}}
    for (i=0;i<1;i++){
      for (j=0;j<1;j++){
          for (k=0;k<1; k++) {
            for (m=2;m<n_timer;m++){
                if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
                build_parset(fp, parset, "%d\n%d\n%g\n%g\n",
                          stat_start[k], ND0, LAMBDA, MU);
                fprintf (fp, "1\n");
                fprintf (fp, "%d\n",codi_mesura);
                fprintf (fp, "%d\n", methods[i]);
                fprintf (fp, "%g\n",timer[m]);
                fprintf (fp, "%g\n", tol[j]);
                fprintf (fp, "%d\n", n_absc-3);
                if (n_absc > 1) fprintf (fp, "s\n");
                for (ii = 1; ii <= n_absc-3; ii++) {
                  (void) fprintf(fp, "%g", absc[ii]);
                  if (ii < n_absc-3) (void) fputc(',', fp);
                  else (void) fputc('\n', fp);
                }
                fprintf (fp, "y\n");
                if (fclose (fp) != 0) goto on_error;

                /* run */
                if (system(modelexe)) goto on_error;

                /* read the computed value of the measure */
                if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ",
n_absc-3,
                                       mesura) != 0) goto on_error;

                /* log file: print vector identity */
                print_ident_vector_one_meth(MODELNAME, parset,
                                      MEASURENAME, methods[i],
                                      tol[j], logfile);

                /* log file: print results about the correctness
                 * of the computed value of the measure */
                (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
                for (ii = 1; ii <= n_absc-3; ii++) {
                  if (check_result_CIAVD(methods[i], tol[j],
true_meas[ii+m*n_absc],
                                         mesura[ii], logfile) != 0) {
                      goto on_error;
                  }}
                (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

                /* log file: print results about memory checking */
                if (append_file(logfile, VALGRIND_OUT,
DRIVER_LOG_BEG_MEMRES,
```

```
                                DRIVER_LOG_END_MEMRES) != 0) goto on_error;
            }}}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_10 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {NDO=10,la=1E-4,la_ph=1E-3,mu=0.02,C=0.95,PO=0.75} X {time=100} X
 * {abscissae=list of values: (0.1, 0.5, 0.9, 0.99, 0.999)} X {meth: 2} X
 * {err: 1e-9}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_10"
#define MEASURENAME "IACD"
#define ND0 10
#define LAMBDA 0.0001
#define LAMBDA_PH 0.001
#define MU 0.02
#define C 0.95
#define P0 0.75
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 5

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int it,ii,n_it,codi_mesura;
    int n_absc;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {2};
    float tol[]= {1e-9},
      timer[] = {100};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
```

```
if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
  goto on_error;
}

/* opens the log file */
if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

/* measure code */
if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

/* main loop */
n_it = 1;
for (it = 0; it < n_it; it++) {
  if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
  build_parset(fp, parset, "%d\n%g\n%g\n%g\n%g\n%g\n",
               ND0, LAMBDA, LAMBDA_PH, MU, C, P0);
  fprintf (fp, "1\n");
  fprintf (fp, "%d\n",codi_mesura);
  fprintf (fp, "%d\n", methods[it]);
  fprintf (fp, "%g\n",timer[it]);
  fprintf (fp, "%g\n", tol[it]);
  fprintf (fp, "%d\n", n_absc);
  if (n_absc > 1) fprintf (fp, "s\n");
  for (ii = 1; ii <= n_absc; ii++) {
      (void) fprintf(fp, "%g", absc[ii]);
      if (ii < n_absc) (void) fputc(',', fp);
      else (void) fputc('\n', fp);
  }
  fprintf (fp, "y\n");
  if (fclose (fp) != 0) goto on_error;

  /* run */
  if (system(modelexe)) goto on_error;

  /* read the computed value of the measure */
  if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                     mesura) != 0) goto on_error;

  /* log file: print vector identity */
  print_ident_vector_one_meth(MODELNAME, parset,
                         MEASURENAME, methods[it],
                         tol[it], logfile);

  /* log file: print results about the correctness
   * of the computed value of the measure */
  (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
  for (ii = 1; ii <= n_absc; ii++) {
      if (check_result_CIAVD(methods[it], tol[it], true_meas[ii],
                      mesura[ii], logfile) != 0) {
        goto on_error;
      }
  }
  (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

  /* log file: print results about the test of the CTMC */
  if (check_Markov_chain(MODELNAME, logfile) != 0) goto on_error;

  /* log file: print results about memory checking */
  if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
              DRIVER_LOG_END_MEMRES) != 0) goto on_error;
}
```

```
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_11 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {NDO=10,la=1E-4,mu=1,C=0.95,PO=0.75} X {time=100} X
 * {abscissae=list of values: (0.1, 0.5, 0.9, 0.99, 0.999)} X {meth: 2} X
 * {err: 1e-9}
 */


#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_11"
#define MEASURENAME "IACD"
#define ND0 10
#define LAMBDA 0.0001
#define MU 1.0
#define C 0.95
#define P0 0.75
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 5

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int it,ii,n_it,codi_mesura;
    int n_absc;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {2};
    float tol[]= {1e-9},
      timer[] = {100};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
```

```
}

/* opens the log file */
if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

/* measure code */
if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

/* main loop */
n_it = 1;
for (it = 0; it < n_it; it++) {
  if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
  build_parset(fp, parset, "%d\n%g\n%g\n%g\n%g\n",
               ND0, LAMBDA, MU, C, P0);
  fprintf (fp, "1\n");
  fprintf (fp, "%d\n",codi_mesura);
  fprintf (fp, "%d\n", methods[it]);
  fprintf (fp, "%g\n",timer[it]);
  fprintf (fp, "%g\n", tol[it]);
  fprintf (fp, "%d\n", n_absc);
  if (n_absc > 1) fprintf (fp, "s\n");
  for (ii = 1; ii <= n_absc; ii++) {

      (void) fprintf(fp, "%g", absc[ii]);
      if (ii < n_absc) (void) fputc(',', fp);
      else (void) fputc('\n', fp);
  }
  fprintf (fp, "y\n");
  if (fclose (fp) != 0) goto on_error;

  /* run */
  if (system(modelexe)) goto on_error;

  /* read the computed value of the measure */
  if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                     mesura) != 0) goto on_error;

  /* log file: print vector identity */
  print_ident_vector_one_meth(MODELNAME, parset,
                        MEASURENAME, methods[it],
                        tol[it], logfile);

  /* log file: print results about the correctness
   * of the computed value of the measure */
  (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
  for (ii = 1; ii <= n_absc; ii++) {
      if (check_result_CIAVD(methods[it], tol[it], true_meas[ii],
                      mesura[ii], logfile) != 0) {
        goto on_error;
      }
  }
  (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

  /* log file: print results about the test of the CTMC */
  if (check_Markov_chain(MODELNAME, logfile) != 0) goto on_error;

  /* log file: print results about memory checking */
  if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
              DRIVER_LOG_END_MEMRES) != 0) goto on_error;
}
if (fclose(logfile) != 0) goto on_error;
```

```
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_12 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {NDO=10,la=1E-4,mu=1,C=0.95,PO=0.75} X {time=100} X
 * {abscissae=list of values: (0.1, 0.5, 0.9, 0.99, 0.999)} X {meth: 2} X
 * {err: 1e-9}
 */


#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_12"
#define MEASURENAME "IACD"
#define ND0 10
#define LAMBDA 0.0001
#define MU 1.0
#define C 0.95
#define P0 0.75
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 5

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int it,ii,n_it,codi_mesura;
    int n_absc;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {2};
    float tol[]= {1e-9},
      timer[] = {100};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
                VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }
```

```
/* opens the log file */
if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

/* measure code */
if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

/* main loop */
n_it = 1;
for (it = 0; it < n_it; it++) {
  if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
  build_parset(fp, parset, "%d\n%g\n%g\n%g\n%g\n",
               ND0, LAMBDA, MU, C, P0);
  fprintf (fp, "1\n");
  fprintf (fp, "%d\n",codi_mesura);
  fprintf (fp, "%d\n", methods[it]);
  fprintf (fp, "%g\n",timer[it]);
  fprintf (fp, "%g\n", tol[it]);
  fprintf (fp, "%d\n", n_absc);
  if (n_absc > 1) fprintf (fp, "s\n");
  for (ii = 1; ii <= n_absc; ii++) {

      (void) fprintf(fp, "%g", absc[ii]);
      if (ii < n_absc) (void) fputc(',', fp);
      else (void) fputc('\n', fp);
  }
  fprintf (fp, "y\n");
  if (fclose (fp) != 0) goto on_error;

  /* run */
  if (system(modelexe)) goto on_error;

  /* read the computed value of the measure */
  if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                     mesura) != 0) goto on_error;

  /* log file: print vector identity */
  print_ident_vector_one_meth(MODELNAME, parset,
                         MEASURENAME, methods[it],
                         tol[it], logfile);

  /* log file: print results about the correctness
   * of the computed value of the measure */
  (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
  for (ii = 1; ii <= n_absc; ii++) {
      if (check_result_CIAVD(methods[it], tol[it], true_meas[ii],
                       mesura[ii], logfile) != 0) {
        goto on_error;
      }
  }
  (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

  /* log file: print results about the test of the CTMC */
  if (check_Markov_chain(MODELNAME, logfile) != 0) goto on_error;

  /* log file: print results about memory checking */
  if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
              DRIVER_LOG_END_MEMRES) != 0) goto on_error;
}
if (fclose(logfile) != 0) goto on_error;
if (lookfor_errors(MODELNAME) != 0) goto on_error;
```

```
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_13 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {stat_start = 1} X {time=10} X abscissae=list of values: (0.1,
 * 0.5, 0.9, 0.99, 0.999) X {meth: 1} X {err: 1e-6}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_13"
#define MEASURENAME "IACD"
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 5

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,k,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_stat_start, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {1},
      stat_start[]={1};
    float tol[]= {1e-6},
      timer[] = {10};
    double absc[] = {0.0, 0.1, 0.5, 0.9, 0.99, 0.999};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_stat_start = sizeof(stat_start)/sizeof(int);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
                VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }
```

```
    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<1;i++){
      for (j=0;j<n_tol;j++){
          for (k=0;k<1; k++) {
            for (m=0;m<n_timer;m++){
                if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
                build_parset(fp, parset, "%d\n%g\n%g\n",
                          stat_start[k], LAMBDA, MU);
                fprintf (fp, "1\n");
                fprintf (fp, "%d\n",codi_mesura);
                fprintf (fp, "%d\n", methods[i]);
                fprintf (fp, "%g\n",timer[m]);
                fprintf (fp, "%g\n", tol[j]);
                fprintf (fp, "%d\n", n_absc);
                if (n_absc > 1) fprintf (fp, "s\n");
                for (ii = 1; ii <= n_absc; ii++) {
                  (void) fprintf(fp, "%g", absc[ii]);
                  if (ii < n_absc) (void) fputc(',', fp);
                  else (void) fputc('\n', fp);
                }
                fprintf (fp, "y\n");
                if (fclose (fp) != 0) goto on_error;

                /* run */
                if (system(modelexe)) goto on_error;

                /* read the computed value of the measure */
                if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ",
                                n_absc,
                                mesura) != 0) goto on_error;

                /* log file: print vector identity */
                print_ident_vector_one_meth(MODELNAME, parset,
                                   MEASURENAME, methods[i],
                                   tol[j], logfile);

                /* log file: print results about the correctness
                 * of the computed value of the measure */
                (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
                for (ii = 1; ii <= n_absc; ii++) {
                  if (check_result_CIAVD(methods[i], tol[j],
                                    true_meas[ii+m*n_absc],
                                    mesura[ii], logfile) != 0) {
                      goto on_error;
                  }
                }
                (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

                /* log file: print results about memory checking */
                if (append_file(logfile, VALGRIND_OUT,
                            DRIVER_LOG_BEG_MEMRES,
```

```
                                DRIVER_LOG_END_MEMRES) != 0) goto on_error;
            }}}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}

/*
 * Driver for the CIAVD_2 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {time: 10} X abscissae=list of values: (0.999) X {meth: 1} X {err: 1e-
6,
 * 1e-9, 1e-12}
 * {time: 10} X abscissae=list of values: (0.999) X {meth: 2} X {err: 1e-
6,
 * 1e-9, 1e-12}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_2"
#define MEASURENAME "IACD"
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 1

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1,2};
    float tol[]= {1e-6,1e-9,1e-12},
      timer[] = {10};
    double absc[] = {0.0, 0.999};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
               VALGRIND_OUT);
```

```
    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }
    /* opens the log file */

    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;

    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<1;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%g\n%g\n",
                         LAMBDA, MU);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CIAVD(methods[i], tol[j], true_meas[ii],
                                       mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;
```

```
            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                        DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }}}
    for (i=1;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%g\n%g\n",
                       LAMBDA, MU);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                    MEASURENAME, methods[i],
                                    tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CIAVD(methods[i], tol[j], true_meas[ii],
                                    mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                        DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
```

```
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}
/*
 * Driver for the CIAVD_3 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 *
 * {time=1000} X abscissae=list of values: (0.85, 0.95, 0.99, 0.999,
0.9999,
 * 0.99999, 0.999999) X {meth: 1} X {err: 1e-10}
 */

#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"

#define MODELNAME "CIAVD_3"
#define MEASURENAME "IACD"
#define NP 3
#define NM 3
#define P0 0.75
#define LDP 2e-4
#define LDM 4.5e-4
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 7
int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,m,ii,codi_mesura;
    int n_methods,n_tol,n_absc, n_timer;
    double true_meas[N_TRUE_MEAS+1];
    int methods []= {1,2};
    float tol[]= {1e-10},
      timer[] = {1000};
    double absc[] = {0, 0.85,0.95,0.99,0.999,0.9999,0.99999,0.999999};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;

    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
              VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }

    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;
```

```
    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<1;i++){
      for (j=0;j<n_tol;j++){
          for (m=0;m<n_timer;m++){
            if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
            build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n",
                         NP, NM, P0, LDP, LDM);
            fprintf (fp, "1\n");
            fprintf (fp, "%d\n",codi_mesura);
            fprintf (fp, "%d\n", methods[i]);
            fprintf (fp, "%g\n",timer[m]);
            fprintf (fp, "%g\n", tol[j]);
            fprintf (fp, "%d\n", n_absc);
            if (n_absc > 1) fprintf (fp, "s\n");
            for (ii = 1; ii <= n_absc; ii++) {
                (void) fprintf(fp, "%g", absc[ii]);
                if (ii < n_absc) (void) fputc(',', fp);
                else (void) fputc('\n', fp);
            }
            fprintf (fp, "y\n");
            if (fclose (fp) != 0) goto on_error;

            /* run */
            if (system(modelexe)) goto on_error;

            /* read the computed value of the measure */
            if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ", n_absc,
                                   mesura) != 0) goto on_error;

            /* log file: print vector identity */
            print_ident_vector_one_meth(MODELNAME, parset,
                                        MEASURENAME, methods[i],
                                        tol[j], logfile);

            /* log file: print results about the correctness
             * of the computed value of the measure */
            (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
            for (ii = 1; ii <= n_absc; ii++) {
                if (check_result_CIAVD(methods[i], tol[j], true_meas[ii],
                                       mesura[ii], logfile) != 0) {
                  goto on_error;
                }
            }
            (void) fprintf(logfile, "%s\n\n", DRIVER_LOG_END_NUMRES);

            /* log file: print results about the test of the CTMC */
            if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

            /* log file: print results about memory checking */
            if (append_file(logfile, VALGRIND_OUT, DRIVER_LOG_BEG_MEMRES,
                            DRIVER_LOG_END_MEMRES) != 0) goto on_error;
          }}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
```

```
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}
/*
 * Driver for the CIAVD_4 model.
 * measure: IACD.
 * available numerical methods:
 * 1 (A), 2 (A_TRF).
 * {stat_start = 1} X {rw = 0, 1} X {time = 10} X abscissae=list of
values:
 * (0.1) X {meth: 1} X {err: 1e-6}
 */
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "driver_lib.h"
#define MODELNAME "CIAVD_4"
#define MEASURENAME "IACD"
#define ND0 1
#define LAMBDA 0.0001
#define MU 1.0
#define MAXSTEPS 1000000
#define N_TRUE_MEAS 2

int
main(void)
{
    FILE *fp, *logfile;
    char tmp[BUF_SZ], modelexe[BUF_SZ], parset[BUF_SZ];
    double mesura[N_TRUE_MEAS+1];
    int i,j,k,m,ii,l, codi_mesura;
    int n_methods,n_tol,n_absc, n_stat_start, n_timer, n_rw;
    double true_meas[N_TRUE_MEAS+1];
    int methods[]= {1},
      stat_start[]={1};
    float tol[]= {1e-6},
      timer[] = {10};
    double absc[] = {0.0, 0.1},
      rw[] = {0.0, 1.0};

    n_methods  = sizeof(methods)/sizeof(int);
    n_tol  = sizeof (tol)/sizeof(float);
    n_stat_start = sizeof(stat_start)/sizeof(int);
    n_timer= sizeof(timer)/sizeof(float);
    n_absc = sizeof(absc)/sizeof(double)-1;
    n_rw = sizeof(rw)/sizeof(double);
    /* builds up the string to call the model executable file */
    build_met2comm(modelexe, MODELNAME, MODEL_TMP_INP, "/dev/null",
              VALGRIND_OUT);

    /* gets the "exact" values of the measure that have been computed
     * elsewhere */
    if (read_exact_values_meas(MODELNAME, N_TRUE_MEAS, true_meas) != 0) {
      goto on_error;
    }
    /* opens the log file */
    if (!(logfile = open_driver_logfile(MODELNAME))) goto on_error;
```

```
    /* measure code */
    if (meas_num_code(MEASURENAME, &codi_mesura) != 0) goto on_error;

    /* main loop */
    for (i=0;i<n_methods;i++){
      for (j=0;j<n_tol;j++){
          for (k=0;k<1; k++) {
            for (m=0;m<n_timer;m++){
                for (l = 0; l < n_rw; l++) {
                  if (!(fp = fopen(MODEL_TMP_INP, "w"))) goto on_error;
                  build_parset(fp, parset, "%d\n%d\n%g\n%g\n%g\n",
                               stat_start[k], ND0, LAMBDA, MU, rw[l]);
                  fprintf (fp, "1\n");
                  fprintf (fp, "%d\n",codi_mesura);
                  fprintf (fp, "%d\n", methods[i]);
                  fprintf (fp, "%g\n",timer[m]);
                  fprintf (fp, "%g\n", tol[j]);
                  fprintf (fp, "%d\n", n_absc);
                  if (n_absc > 1) fprintf (fp, "s\n");
                  for (ii = 1; ii <= n_absc; ii++) {
                      (void) fprintf(fp, "%g", absc[ii]);
                      if (ii < n_absc) (void) fputc(',', fp);
                      else (void) fputc('\n', fp);
                  }
                  fprintf (fp, "y\n");
                  if (fclose (fp) != 0) goto on_error;

                  /* run */
                  if (system(modelexe)) goto on_error;

                  /* read the computed value of the measure */
                  if (read_met2_val_meas(MODELNAME, MEASURENAME, " = ",
n_absc,
                                         mesura) != 0) goto on_error;

                  /* log file: print vector identity */
                  print_ident_vector_one_meth(MODELNAME, parset,
                                              MEASURENAME, methods[i],
                                              tol[j], logfile);

                  /* log file: print results about the correctness
                   * of the computed value of the measure */
                  (void) fprintf(logfile, "%s\n", DRIVER_LOG_BEG_NUMRES);
                  for (ii = 1; ii <= n_absc; ii++) {
                      if (check_result_CIAVD(methods[i], tol[j],
true_meas[ii+l],
                                             mesura[ii], logfile) != 0) {
                        goto on_error;
                      }}
                  (void) fprintf(logfile, "%s\n\n",
DRIVER_LOG_END_NUMRES);

                  /* log file: print results about the test of the CTMC
*/
                  if (check_Markov_chain(MODELNAME, logfile) != 0) goto
on_error;

                  /* log file: print results about memory checking */
                  if (append_file(logfile, VALGRIND_OUT,
DRIVER_LOG_BEG_MEMRES,
                                  DRIVER_LOG_END_MEMRES) != 0) goto on_error;
```

```
                }}}}}
    if (fclose(logfile) != 0) goto on_error;
    if (lookfor_errors(MODELNAME) != 0) goto on_error;
    return EXIT_SUCCESS;
 on_error:
    build_driver_filename(MODELNAME, tmp);
    (void) fprintf(stderr, "%s: execution failed.\n", tmp);
    return EXIT_FAILURE;
}
```