MUTUAL EXCLUSION ALGORITHMS

(Robert Meolic, 2000, 2001, 2002, 2003, 2005, 2013)

Many results of this project has been published in papers.

File **me.dat** contains description of the following algorithms:

- Dekker's algorithm,
- Hyman's algorithm,
- Peterson's algorithm
- Lamport's Bakery algorithm,
- Ben-Ari's variant of Bakery algorithm,
- The STeP variant of Bakery algorithm,
- Lamport's (not Bakery) algorithm,
- One-bit algorithms,
- Fischer's algorithms

There are two special processes: EMPTY and EMPTY_wor. They are used to easily add or remove extra notification actions used in model checking (wor = without request signals). We need them for model checking but they can be omitted for more efficient equivalence checking.

File **me.actl** contains formulae for ACTLW model checking:

```
#
#The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false});

#
#The composed model contains a divergent state.
/* NOTE: the paper from 2008 gives incorrect formula EEF EEG {tau} EEX {true};
*/

(EEX {true} AND EEG {tau} EEX {true}) OR
  (EEF (EEX {true} AND EEG {tau} EEX {true}));

#
#If process P1 enters the critical section, then process P2 cannot enter
#it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}];

#
#If process P2 enters the critical section, then process P1 cannot enter
#it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}];

#
#If process P1 wants to enter the critical section, then process P2 can enter
#the critical section at most once before process P1 enters the critical
section.
```

```
AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]];

#
#If process P2 wants to enter the critical section, then process P1 can enter
#the critical section at most once before process P2 enters the critical
section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]];
```

Files **dekker.dat**, **peterson.dat**, **bakery.dat**, and **lamport.dat** contains some results from minimization and equivalence checking.

Here is the log from EST:

```
Efficient Symbolic Tools, 2nd Edition, Copyright (C) 2003-2013 UM-FERI
This is free software, and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute EST; type "license" for details.

Running on i686 (Linux, 3.2.0-40-generic-pae) with tcl 8.5.11 and tk 8.5.11.

Initialization of GUI package... OK
Initialization of BDD package... OK
Initialization of Process_Algebra package... OK
Initialization of Versis package... OK
Initialization of Model checking package... OK
Initialization of Strucval package... OK
Initialization of CCS package... OK
Ready.

>cd "/home/meolic/est/est-2ed/data/me"; source "me.tcl"; cd "/home/meolic/est/est-2ed/data"
Reading file: me.dat
 Algorithms for mutual exclusion
 Dekker, Hyman, Peterson, Bakery, Benari, STEP
 Robert Meolic, July 2005

 Thanks to:
 Tomaz Felicijan, Dekker, September 2000
 Ernest Gungl, Dekker, June 2001
 Ernest Gungl, Hyman, June 2001
 Ernest Gungl, Peterson, June 2001
 David Dedic, Bakery, December 2002
 David Dedic, Benari, December 2002
 David Dedic, STEP, December 2002
  Sort sortME ... OK
  Sort sortMEfischer ... OK
  Process EMPTY ... OK
  Process EMPTY_wor ... OK
  Process B1 ... OK
  Process B2 ... OK
  Process K ... OK
  Process N1 ... OK
  Process N2 ... OK
  Process N1PLUS ... OK
  Process N2PLUS ... OK
  Process NPLUS ... OK
  Process P1DEKKER ... OK
  Process P2DEKKER ... OK
  Process P1HYMAN ... OK
  Process P2HYMAN ... OK
  Process P1PETERSON ... OK
  Process P2PETERSON ... OK
  Process P1BAKERY ... OK
  Process P2BAKERY ... OK
  Process P1BENARI ... OK
  Process P2BENARI ... OK
  Process P1STEP ... OK
  Process P2STEP ... OK

 More algorithms for mutual exclusion
 Lamport, One-bit, Fischer
```

```
 Thanks to:
 Bostjan Vlaovic, Lamport, September 2001
 Ljubo Oberski, One-bit, December 2005
 Aleksander Vreze, Fischer, June 2003

  Process P1LAMPORT ... OK
  Process P2LAMPORT ... OK
  Process P1LAMPORTnew ... OK
  Process P2LAMPORTnew ... OK
  Process P1ONEBIT ... OK
  Process P2ONEBIT ... OK
  Process EMPTYFISCHER ... OK
  Process EMPTYFISCHER_wor ... OK
  Process REALCLOCK ... OK
  Process X ... OK
  Process NOP ... OK
  Process P1FISCHER ... OK
  Process P2FISCHER ... OK


##########################################################

DEKKER'S ALGORITHM
Parallel composition (1): DEKKER...

STATISTICS
pa_comp_state_number = 164
pa_comp_transition_number = 310
pa_comp_transition_visible = 88

ACTL/ACTLW model checking on composition DEKKER using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> TRUE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> FALSE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> FALSE

##########################################################

HYMAN'S ALGORITHM
Parallel composition (1): HYMAN...

STATISTICS
pa_comp_state_number = 96
pa_comp_transition_number = 182
pa_comp_transition_visible = 84

ACTL/ACTLW model checking on composition HYMAN using file me.actl

The composed model contains a deadlocked state.
```

```
(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> FALSE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> FALSE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> FALSE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> FALSE

##########################################################

COUNTEREXAMPLE FOR HYMAN
ACTL/ACTLW model checking on composition HYMAN
AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> FALSE
@@ Diagnostics
@@ #0:AAG [enter1!] AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P0<P1HYMAN>),(P0<P2HYMAN>),
(B1f<B1>),(B2f<B2>),(K1<K>))
@@ There exist a path not satisfying formula #0: ((s0<EMPTY>),(P0<P1HYMAN>),(P0<P2HYMAN>),(B1f<B1>),
(B2f<B2>),(K1<K>))--TAU->((s0<EMPTY>),(P0<P1HYMAN>),(P1a<P2HYMAN>),(B1f<B1>),(B2t<B2>),(K1<K>))--
request2!->((s0<EMPTY>),(P0<P1HYMAN>),(P1<P2HYMAN>),(B1f<B1>),(B2t<B2>),(K1<K>))--TAU->((s0<EMPTY>),
(P0<P1HYMAN>),(P5<P2HYMAN>),(B1f<B1>),(B2t<B2>),(K1<K>))--TAU->((s0<EMPTY>),(P0<P1HYMAN>),
(P6<P2HYMAN>),(B1f<B1>),(B2t<B2>),(K1<K>))--TAU->((s0<EMPTY>),(P1a<P1HYMAN>),(P6<P2HYMAN>),
(B1t<B1>),(B2t<B2>),(K1<K>))--request1!->((s0<EMPTY>),(P1<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),
(B2t<B2>),(K1<K>))--TAU->((s0<EMPTY>),(P2<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),(B2t<B2>),(K1<K>))
@@ #1:[enter1!] AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P2<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),
(B2t<B2>),(K1<K>))
@@ There exist a transition not satisfying formula #1: ((s0<EMPTY>),(P2<P1HYMAN>),(P6<P2HYMAN>),
(B1t<B1>),(B2t<B2>),(K1<K>))--enter1!->((s0<EMPTY>),(P3<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),(B2t<B2>),
(K1<K>))
@@ #2:AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P3<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),(B2t<B2>),
(K1<K>))
@@ There exist a path not satisfying formula #2: ((s0<EMPTY>),(P3<P1HYMAN>),(P6<P2HYMAN>),(B1t<B1>),
(B2t<B2>),(K1<K>))--TAU->((s0<EMPTY>),(P3<P1HYMAN>),(P1<P2HYMAN>),(B1t<B1>),(B2t<B2>),(K2<K>))--TAU-
>((s0<EMPTY>),(P3<P1HYMAN>),(P2<P2HYMAN>),(B1t<B1>),(B2t<B2>),(K2<K>))--enter2!->((s0<EMPTY>),
(P3<P1HYMAN>),(P3<P2HYMAN>),(B1t<B1>),(B2t<B2>),(K2<K>))
@@ Counterexample: (TAU)(request2!)(TAU)(TAU)(TAU)(request1!)(TAU)(enter1!)(TAU)(TAU)(enter2!)
@@ Spin trail generated and saved into file wc.out
@@ MSC generated and saved into file wc.msc
@@ End Of Diagnostic

##########################################################

PETERSON'S ALGORITHM
Parallel composition (1): PETERSON...

STATISTICS
pa_comp_state_number = 68
pa_comp_transition_number = 136
pa_comp_transition_visible = 42

ACTL/ACTLW model checking on composition PETERSON using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.
```

```
(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> TRUE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

########################################################

LAMPORT'S BAKERY ALGORITHM
Parallel composition (1): BAKERY...

STATISTICS
pa_comp_state_number = 292
pa_comp_transition_number = 521
pa_comp_transition_visible = 121

ACTL/ACTLW model checking on composition BAKERY using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> TRUE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

########################################################

BEN-ARI'S VARIANT OF BAKERY ALGORITHM
Parallel composition (1): BENARI...

STATISTICS
pa_comp_state_number = 214
pa_comp_transition_number = 383
pa_comp_transition_visible = 102

ACTL/ACTLW model checking on composition BENARI using file me.actl

The composed model contains a deadlocked state.
```

```
(AAX {false}) OR (EEF AAX {false}) ==> TRUE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

#########################################################

THE STeP VARIANT OF BAKERY ALGORITHM
Parallel composition (1): STEP...

STATISTICS
pa_comp_state_number = 242
pa_comp_transition_number = 451
pa_comp_transition_visible = 162

ACTL/ACTLW model checking on composition STEP using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> FALSE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> FALSE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

#########################################################

THE STeP VARIANT OF BAKERY ALGORITHM - ATOMIC INCREMENTATION
Parallel composition (1): STEP2...

STATISTICS
pa_comp_state_number = 112
pa_comp_transition_number = 191
```

```
pa_comp_transition_visible = 65

ACTL/ACTLW model checking on composition STEP2 using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

########################################################

COUNTEREXAMPLE FOR STEP
ACTL/ACTLW model checking on composition STEP
AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> FALSE
@@ Diagnostics
@@ #0:AAG [enter1!] AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P0<P1STEP>),(P0<P2STEP>),(N10<N1>),
(N20<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ There exist a path not satisfying formula #0: ((s0<EMPTY>),(P0<P1STEP>),(P0<P2STEP>),(N10<N1>),
(N20<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P0<P1STEP>),(P2<P2STEP>),(N10<N1>),
(N20<N2>),(N1PLUS<N1PLUS>),(PL1<N2PLUS>))--TAU->((s0<EMPTY>),(P0<P1STEP>),(P2<P2STEP>),(N10<N1>),
(N20<N2>),(N1PLUS<N1PLUS>),(PL2<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P2<P2STEP>),(N10<N1>),
(N20<N2>),(PL1<N1PLUS>),(PL2<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P2<P2STEP>),(N10<N1>),
(N20<N2>),(PL2<N1PLUS>),(PL2<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P2<P2STEP>),(N10<N1>),
(N21<N2>),(PL2<N1PLUS>),(PL4<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P5a<P2STEP>),(N10<N1>),
(N21<N2>),(PL2<N1PLUS>),(N2PLUS<N2PLUS>))--request2!->((s0<EMPTY>),(P2<P1STEP>),(P5<P2STEP>),
(N10<N1>),(N21<N2>),(PL2<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P7<P2STEP>),
(N10<N1>),(N21<N2>),(PL2<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P8<P2STEP>),
(N10<N1>),(N21<N2>),(PL2<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P2<P1STEP>),(P8<P2STEP>),
(N11<N1>),(N21<N2>),(PL4<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P5a<P1STEP>),(P8<P2STEP>),
(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))--request1!->((s0<EMPTY>),(P5<P1STEP>),
(P8<P2STEP>),(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))--TAU->((s0<EMPTY>),(P7<P1STEP>),
(P8<P2STEP>),(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ #1:[enter1!] AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P7<P1STEP>),(P8<P2STEP>),(N11<N1>),
(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ There exist a transition not satisfying formula #1: ((s0<EMPTY>),(P7<P1STEP>),(P8<P2STEP>),
(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))--enter1!->((s0<EMPTY>),(P8<P1STEP>),
(P8<P2STEP>),(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ #2:AA[{NOT enter2!} W {exit1!}]:F:((s0<EMPTY>),(P8<P1STEP>),(P8<P2STEP>),(N11<N1>),(N21<N2>),
(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ There exist a path not satisfying formula #2: ((s0<EMPTY>),(P8<P1STEP>),(P8<P2STEP>),(N11<N1>),
(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))--enter2!->((s0<EMPTY>),(P8<P1STEP>),(P9<P2STEP>),
(N11<N1>),(N21<N2>),(N1PLUS<N1PLUS>),(N2PLUS<N2PLUS>))
@@ Counterexample: (TAU)(TAU)(TAU)(TAU)(TAU)(TAU)(request2!)(TAU)(TAU)(TAU)(TAU)(request1!)(TAU)
(enter1!)(enter2!)
@@ Spin trail generated and saved into file wc.out
@@ MSC generated and saved into file wc.msc
@@ End Of Diagnostic

########################################################
COMPARISON OF DIFFERENT VARIANTS OF BAKERY ALGORITHM
Trace equivalence checking between STEP and STEP2... NOT EQUIVALENT
Testing equivalence checking between STEP and STEP2... NOT EQUIVALENT
```

```
Trace equivalence checking between STEP and BENARI... NOT EQUIVALENT
Trace equivalence checking between STEP and BAKERY... NOT EQUIVALENT
Trace equivalence checking between BENARI and BAKERY... NOT EQUIVALENT
Testing equivalence checking between STEP and BENARI... NOT EQUIVALENT
Testing equivalence checking between STEP and BAKERY... NOT EQUIVALENT
Testing equivalence checking between BENARI and BAKERY... NOT EQUIVALENT

########################################################

LAMPORTS'S (not Bakery) ALGORITHM
Parallel composition (1): LAMPORT...

STATISTICS
pa_comp_state_number = 26
pa_comp_transition_number = 48
pa_comp_transition_visible = 12

ACTL/ACTLW model checking on composition LAMPORT using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> TRUE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE

########################################################

ONEBIT ALGORITHM
Parallel composition (1): ONEBIT...

STATISTICS
pa_comp_state_number = 53
pa_comp_transition_number = 96
pa_comp_transition_visible = 36

ACTL/ACTLW model checking on composition ONEBIT using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> FALSE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.
```

```
AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> FALSE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> FALSE

########################################################

FISCHER'S ALGORITHM
Parallel composition (1): FISCHER...

STATISTICS
pa_comp_state_number = 520
pa_comp_transition_number = 978
pa_comp_transition_visible = 72

ACTL/ACTLW model checking on composition FISCHER using file me.actl

The composed model contains a deadlocked state.

(AAX {false}) OR (EEF AAX {false}) ==> FALSE

The composed model contains a divergent state.

(EEX {true} AND EEG {TAU} EEX {true}) OR (EEF (EEX {true} AND EEG {TAU} EEX {true})) ==> TRUE

If process P1 enters the critical section, then process P2 cannot enter
it, unless process P1 leaves the critical section.

AAG [enter1!] AA[{NOT enter2!} W {exit1!}] ==> TRUE

If process P2 enters the critical section, then process P1 cannot enter
it, unless process P2 leaves the critical section.

AAG [enter2!] AA[{NOT enter1!} W {exit2!}] ==> TRUE

If process P1 wants to enter the critical section, then process P2 can enter
the critical section at most once before process P1 enters the critical section.

AAG [request1!] NOT EE[{NOT enter1!} U {enter2!} EE[{NOT enter1!} U {enter2!}]] ==> TRUE

If process P2 wants to enter the critical section, then process P1 can enter
the critical section at most once before process P2 enters the critical section.

AAG [request2!] NOT EE[{NOT enter2!} U {enter1!} EE[{NOT enter2!} U {enter1!}]] ==> TRUE
```

Here are some data about timing on Intel Atom D525 with 2GB RAM:

```
[CPU]versis_compose 1 DEKKER {EMPTY P1DEKKER P2DEKKER B1 B2 K} {request1
request2 enter1 exit1 enter2 exit2}[CPU: 45ms]

[CPU]mc_check_actl_file 1 DEKKER me.actl[CPU: 211ms]

[CPU]versis_compose 1 HYMAN {EMPTY P1HYMAN P2HYMAN B1 B2 K} {request1 request2
enter1 exit1 enter2 exit2}[CPU: 37ms]

[CPU]mc_check_actl_file 1 HYMAN me.actl[CPU: 190ms]
[CPU]mc_check_actl 1 HYMAN me_F3 28[CPU: 961ms]

[CPU]versis_compose 1 PETERSON {EMPTY P1PETERSON P2PETERSON B1 B2 K} {request1
request2 enter1 exit1 enter2 exit2}[CPU: 30ms]
[CPU]mc_check_actl_file 1 PETERSON me.actl[CPU: 137ms]
```

```
[CPU]versis_compose 1 BAKERY {EMPTY P1BAKERY P2BAKERY B1 B2 N1 N2 N1PLUS N2PLUS}
{request1 request2 enter1 exit1 enter2 exit2}[CPU: 104ms]

[CPU]mc_check_actl_file 1 BAKERY me.actl[CPU: 362ms]

[CPU]versis_compose 1 BENARI {EMPTY P1BENARI P2BENARI N1 N2 N1PLUS N2PLUS}
{request1 request2 enter1 exit1 enter2 exit2}[CPU: 70ms]

[CPU]mc_check_actl_file 1 BENARI me.actl[CPU: 303ms]

[CPU]versis_compose 1 STEP {EMPTY P1STEP P2STEP N1 N2 N1PLUS N2PLUS} {request1
request2 enter1 exit1 enter2 exit2}[CPU: 71ms]

[CPU]mc_check_actl_file 1 STEP me.actl[CPU: 307ms]

[CPU]versis_compose 1 STEP2 {EMPTY P1STEP P2STEP N1 N2 NPLUS} {request1 request2
enter1 exit1 enter2 exit2}[CPU: 44ms]

[CPU]mc_check_actl_file 1 STEP2 me.actl[CPU: 176ms]
[CPU]mc_check_actl 1 STEP me_F3 28[CPU: 1757ms]

[CPU]versis_trace_equivalence 1 STEP 1 STEP2[CPU: 658ms]
[CPU]versis_testing_equivalence 1 STEP 1 STEP2[CPU: 309ms]
[CPU]versis_trace_equivalence 1 STEP 1 BENARI[CPU: 754ms]
[CPU]versis_trace_equivalence 1 STEP 1 BAKERY[CPU: 1258ms]
[CPU]versis_trace_equivalence 1 BENARI 1 BAKERY[CPU: 74ms]
[CPU]versis_testing_equivalence 1 STEP 1 BENARI[CPU: 199ms]
[CPU]versis_testing_equivalence 1 STEP 1 BAKERY[CPU: 273ms]
[CPU]versis_testing_equivalence 1 BENARI 1 BAKERY[CPU: 90ms]

[CPU]versis_compose 1 LAMPORT {EMPTY P1LAMPORT P2LAMPORT B1 B2} {request1
request2 enter1 exit1 enter2 exit2}[CPU: 25ms]

[CPU]mc_check_actl_file 1 LAMPORT me.actl[CPU: 157ms]

[CPU]versis_compose 1 ONEBIT {EMPTY P1ONEBIT P2ONEBIT B1 B2} {request1 request2
enter1 exit1 enter2 exit2}[CPU: 32ms]

[CPU]mc_check_actl_file 1 ONEBIT me.actl[CPU: 184ms]

[CPU]versis_compose 1 FISCHER {EMPTYFISCHER P1FISCHER P2FISCHER REALCLOCK X NOP}
{request1 request2 enter1 exit1 enter2 exit2}[CPU: 98ms]

[CPU]mc_check_actl_file 1 FISCHER me.actl[CPU: 565ms]
```

Here is a table with comparison of different solutions:

	Deadlock	Divergent	**Mutual exclusion**	Overtaking
Dekker's algorithm	FALSE	TRUE	TRUE	FALSE
Hyman's algorithm	FALSE	FALSE	**FALSE**	FALSE
Peterson's algorithm	FALSE	TRUE	TRUE	TRUE
Bakery algorithm	TRUE	TRUE	TRUE	TRUE
Ben-Ari's variant	TRUE	TRUE	TRUE	TRUE
The SteP variant	FALSE	FALSE	**FALSE**	TRUE
The STeP variant (atomic increm.)	FALSE	FALSE	TRUE	TRUE
Lamport's algorithm	FALSE	TRUE	TRUE	TRUE
Onebit algorithm	FALSE	FALSE	TRUE	FALSE
Fisher's algorithm	FALSE	TRUE	TRUE	TRUE

Here are some old data about timing on Intel Pentium 4 with 1GB RAM (regul.uni-mb.si, 2006, 2010). The reported time is real time, not CPU time. Please note, that in the current version command xsource is implemented completely different.

Parallel composition (1): DEKKER...

STATISTICS
pa_comp_state_number = 164
pa_comp_transition_number = 310
pa_comp_transition_visible = 88

==> CPU: 62 miliseconds <== (EST v4.3, 2006)
==> CPU: 58 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): HYMAN...

STATISTICS
pa_comp_state_number = 96
pa_comp_transition_number = 182
pa_comp_transition_visible = 84

==> CPU: 40 miliseconds <== (EST v4.3, 2006)
==> CPU: 58 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): PETERSON...

STATISTICS
pa_comp_state_number = 68
pa_comp_transition_number = 136
pa_comp_transition_visible = 42

==> CPU: 34 miliseconds <== (EST v4.3, 2006)
==> CPU: 34 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): BAKERY...

STATISTICS
pa_comp_state_number = 292
pa_comp_transition_number = 521
pa_comp_transition_visible = 121

==> CPU: 173 miliseconds <== (EST v4.3, 2006)
==> CPU: 137 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): BENARI...

STATISTICS
pa_comp_state_number = 214
pa_comp_transition_number = 383
pa_comp_transition_visible = 102

==> CPU: 130 miliseconds <== (EST v4.3, 2006)
==> CPU: 112 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): STEP...

STATISTICS
pa_comp_state_number = 242
pa_comp_transition_number = 451
pa_comp_transition_visible = 162

==> CPU: 127 miliseconds <== (EST v4.3, 2006)
==> CPU: 116 miliseconds <== (EST v5.2, 2010)

Parallel composition (1): STEP2...

STATISTICS
pa_comp_state_number = 112
pa_comp_transition_number = 191
pa_comp_transition_visible = 65

==> CPU: 49 miliseconds <== (EST v4.3, 2006)
==> CPU: 60 miliseconds <== (EST v5.2, 2010)

##

Trace equivalence checking between STEP and STEP2... NOT EQUIVALENT
==> CPU: 547 miliseconds <== (EST v4.3, 2006)
==> CPU: 824 miliseconds <==

Testing equivalence checking between STEP and STEP2... NOT EQUIVALENT
==> CPU: 559 miliseconds <== (EST v4.3, 2006)
==> CPU: 927 miliseconds <== (EST v5.2, 2010)

Trace equivalence checking between STEP and BENARI... NOT EQUIVALENT
==> CPU: 669 miliseconds <== (EST v4.3, 2006)
==> CPU: 1211 miliseconds <== (EST v5.2, 2010)

Trace equivalence checking between STEP and BAKERY... NOT EQUIVALENT
==> CPU: 1226 miliseconds <== (EST v4.3, 2006)
==> CPU: 1931 miliseconds <== (EST v5.2, 2010)

Trace equivalence checking between BENARI and BAKERY... NOT EQUIVALENT
==> CPU: 1238 miliseconds <== (EST v4.3, 2006)
==> CPU: 1877 miliseconds <== (EST v5.2, 2010)

Testing equivalence checking between STEP and BENARI... NOT EQUIVALENT
==> CPU: 811 miliseconds <== (EST v4.3, 2006)
==> CPU: 1422 miliseconds <== (EST v5.2, 2010)

Testing equivalence checking between STEP and BAKERY... NOT EQUIVALENT
==> CPU: 1218 miliseconds <== (EST v4.3, 2006)
==> CPU: 2100 miliseconds <== (EST v5.2, 2010)

Testing equivalence checking between BENARI and BAKERY... NOT EQUIVALENT
==> CPU: 1094 miliseconds <== (EST v4.3, 2006)
==> CPU: 2008 miliseconds <== (EST v5.2, 2010)