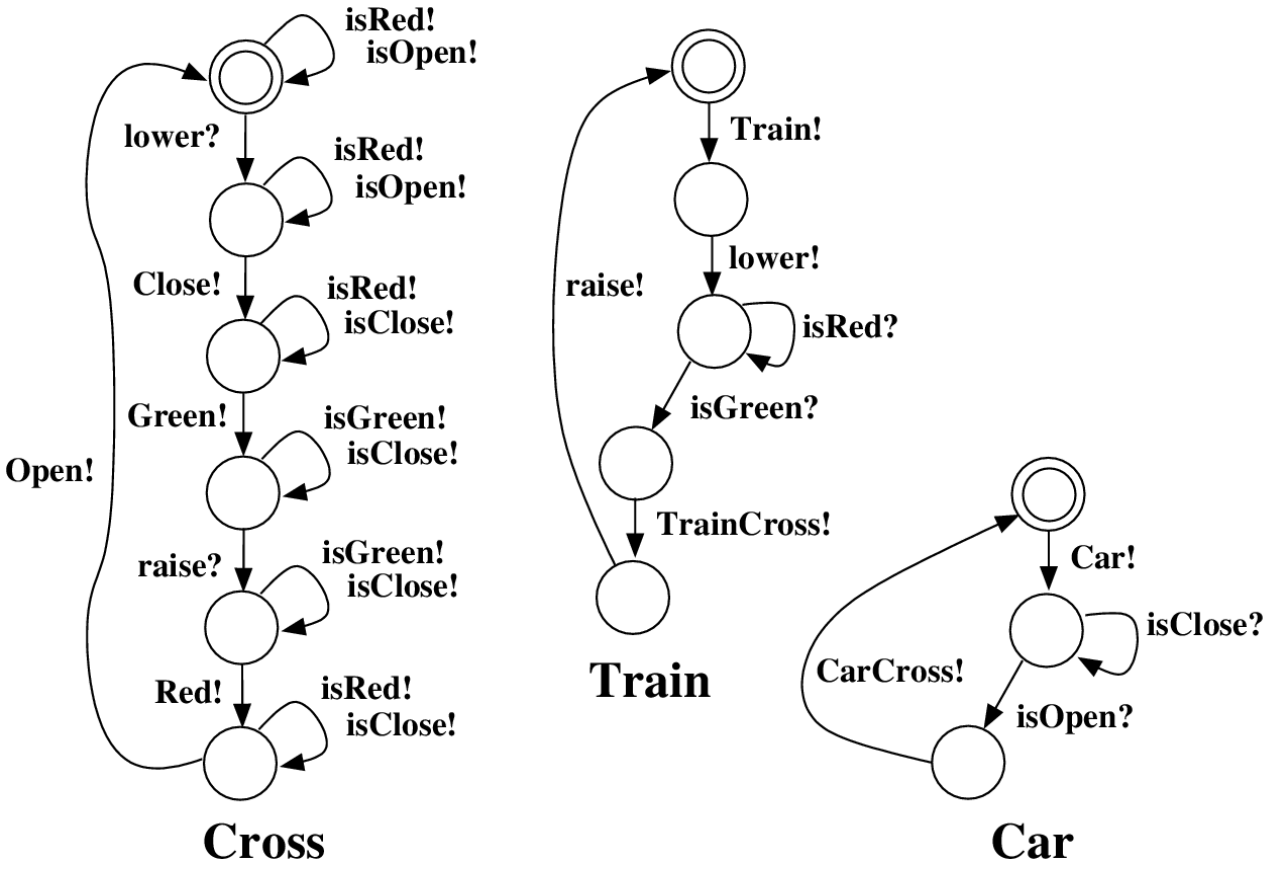
CROSSING OF A ROAD AND A RAILWAY

(Robert Meolic, 2008, 2013)

The crossing consists of two barriers and traffic lights for train. The barriers have to be kept down when the train crosses. The train can cross only if the traffic lights are green, otherwise it has to stop.

The system is represented by processes Cross, Train, and Car. There, actions **isRed**, **isGreen**, **isOpen**, and **isClose** inform a car and a train about the state of the barriers and train traffic lights. Actions **lower** and **raise** are used to change the state of the barriers. Actions **Red**, **Green**, **Close**, and **Open** signal a change of the state of the barriers and train traffic lights to the environment. Actions **Train** and **Car** denote that a train or a car is approaching the crossing. Actions **TrainCross** and **CarCross** denote that a train or a car is crossing.



Here is a description of a system using LTSs:

```
PROCESS cross
SORT sortCROSS
INITIAL STATE p0
TRANSITIONS p0 = isRed!.p0 + isOpen!.p0 + lower?.p1
            p1 = isRed!.p1 + isOpen!.p1 + Close!.p2
            p2 = isRed!.p2 + isClose!.p2 + Green!.p3
            p3 = isGreen!.p3 + isClose!.p3 + raise?.p4
            p4 = isGreen!.p4 + isClose!.p4 + Red!.p5
            p5 = isRed!.p5 + isClose!.p5 + Open!.p0
```

```
PROCESS train
SORT sortCROSS
INITIAL STATE p0
TRANSITIONS p0 = Train!.p1
            p1 = lower!.p2
            p2 = isRed?.p2 + isGreen?.p3
            p3 = TrainCross!.p4
            p4 = raise!.p0

PROCESS car
SORT sortCROSS
INITIAL STATE p0
TRANSITIONS p0 = Car!.p1
            p1 = isClose?.p1 + isOpen?.p2
            p2 = CarCross!.p0
```

Here is a description of a system using CCS:

```
CAR = !Car;CAR_WAIT
CAR_WAIT = ?isClose;CAR_WAIT + ?isOpen;!CarCross;CAR

TRAIN = !Train;!lower;TRAIN_WAIT
TRAIN_WAIT = ?isRed;TRAIN_WAIT + ?isGreen;!TrainCross;!raise;TRAIN

CROSS = !isRed;CROSS + !isOpen;CROSS + ?lower;!Close;!Green;CROSS_CLOSED
CROSS_CLOSED = !isGreen;CROSS_CLOSED + !isClose;CROSS_CLOSED +
               ?raise;!Red;!Open;CROSS

net S = //(CAR,TRAIN,CROSS)\isClose\isOpen\isRed\isGreen\lower\raise
```

Please note, that the given two descriptions of process Cross are not equivalent, but the differences do not have impact on the verification, because the obtained compositions are trace equivalent. Here are ACTL formulae used for verification:

```
/* 1. When the traffic lights turn red, the train will not cross      */
/*    unless the traffic lights turn green.                           */
AAG [Red!] AA[{NOT TrainCross!} W {Green!}];

/* 2. When the barriers are open, the train will not cross            */
/*    unless the barriers lover.                                      */
AAG [Open!] AA[{NOT TrainCross!} W {Close!}];

/* 3. When a train is approaching the crossing, the train will not    */
/*    cross unless the barriers lower.                                */
AAG [Train!] AA[{NOT TrainCross!} W {Close!}];

/* 4. When the barriers are close, they will not raise unless         */
/*    the train crosses.                                              */
AAG [Close!] AA[{NOT Open!} W {TrainCross!}];

/* 5. It never happens that both a car and a train are able to        */
/*    cross at the same time.                                         */
NOT EEF (EE [{tau} U {CarCross!}] AND EE [{tau} U {TrainCross!}]);

/* 6. It never happens that train is able to cross immediately after  */
/*    a car crossed (before any other change in the crossing).        */
NOT EEF (<CarCross!> EE [{tau} U {TrainCross!}]);
```

The last two formulae have very similar meaning but only the last one allows generation of full counterexample.

Here is the log from running crossing.tcl:

```
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, 3.2.0-38-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/crossing"; source "crossing.tcl"; cd "/home/meolic/est/est-
2ed/data"
Reading file: crossing.dat
  Sort sortCROSS ... OK
  Process cross ... OK
  Process train ... OK
  Process car ... OK

Parallel composition (0): crossing...

Reading file: crossing.ccs
  Process CAR ... OK
  Process CAR_WAIT ... OK
  Process TRAIN ... OK
  Process TRAIN_WAIT ... OK
  Process CROSS ... OK
  Process CROSS_CLOSED ... OK
  Net S
    Composition ... OK
    Creating process S ... OK

Strong observational equivalence checking between cross and CROSS... NOT EQUIVALENT
Strong observational equivalence checking between train and TRAIN... OK
Strong observational equivalence checking between car and CAR... OK

Trace equivalence checking between crossing and S... OK
Testing equivalence checking between crossing and S... NOT EQUIVALENT
Weak observational equivalence checking between crossing and S... NOT EQUIVALENT
Strong observational equivalence checking between crossing and S... NOT EQUIVALENT

ACTL/ACTLW model checking on process S using file crossing.actl

AAG [Red!] AA[{NOT TrainCross!} W {Green!}] ==> TRUE
Witness not generated.

AAG [Open!] AA[{NOT TrainCross!} W {Close!}] ==> TRUE
Witness not generated.

AAG [Train!] AA[{NOT TrainCross!} W {Close!}] ==> TRUE
Witness not generated.

AAG [Close!] AA[{NOT Open!} W {TrainCross!}] ==> TRUE
Witness not generated.

NOT EEF (EE[{TAU} U {CarCross!}] AND EE[{TAU} U {TrainCross!}]) ==> FALSE
Counterexample: (Car!)(TAU)(Train!)(TAU)(Close!)(Green!)

NOT EEF (<CarCross!> EE[{TAU} U {TrainCross!}]) ==> FALSE
Counterexample: (Car!)(TAU)(Train!)(TAU)(Close!)(Green!)(CarCross!)(TAU)(TrainCross!)

ACTL/ACTLW model checking on composition crossing
NOT EEF (<CarCross!> EE[{TAU} U {TrainCross!}]) ==> FALSE
```

```
@@ Diagnostics
@@ #0:NOT EEF (<CarCross!> EE[{TAU} U {TrainCross!}]):F:((p0<cross>),(p0<train>),(p0<car>))
@@ Formula #0 is not valid in state ((p0<cross>),(p0<train>),(p0<car>)) because formula #1 is valid
in it.
@@ #1:EEF (<CarCross!> EE[{TAU} U {TrainCross!}]):T:((p0<cross>),(p0<train>),(p0<car>))
@@ There exist a path satisfying formula #1: ((p0<cross>),(p0<train>),(p0<car>))--Train!-
>((p0<cross>),(p1<train>),(p0<car>))--TAU->((p1<cross>),(p2<train>),(p0<car>))--Car!->((p1<cross>),
(p2<train>),(p1<car>))--TAU->((p1<cross>),(p2<train>),(p2<car>))--Close!->((p2<cross>),(p2<train>),
(p2<car>))--Green!->((p3<cross>),(p2<train>),(p2<car>))
@@ #2:<CarCross!> EE[{TAU} U {TrainCross!}]:T:((p3<cross>),(p2<train>),(p2<car>))
@@ There exist a transition satisfying formula #2: ((p3<cross>),(p2<train>),(p2<car>))--CarCross!-
>((p3<cross>),(p2<train>),(p0<car>))
@@ #3:EE[{TAU} U {TrainCross!}]:T:((p3<cross>),(p2<train>),(p0<car>))
@@ There exist a path satisfying formula #3: ((p3<cross>),(p2<train>),(p0<car>))--TAU->((p3<cross>),
(p3<train>),(p0<car>))--TrainCross!->((p3<cross>),(p4<train>),(p0<car>))
@@ #4:true:T:((p3<cross>),(p4<train>),(p0<car>))
@@ Formula #4 is always TRUE.
@@ Counterexample: (Train!)(TAU)(Car!)(TAU)(Close!)(Green!)(CarCross!)(TAU)(TrainCross!)
@@ Spin trail generated and saved into file wc.out
@@ MSC generated and saved into file wc.msc
@@ End Of Diagnostic
```

The results of model checking show a dangerous situation which can arrise in the system. Namely, the last stated ACTLW formula is not valid in the initial state of the compound system. The counterexample is a path where a car drives into the crossing when barriers are open, but then it does not leave the crossing (it does not perform action CarCross!). Because the process Cross does not examine, whether the car has leaved the dangerous area, the traffic lights may turn green before the car leaves the crossing and therefore a collision can appear.

Here is the MSC produced by EST and visualised with mscgen v0.20.