System Verification through Specification and Model-Checking

(Introduction)

Hui Shi

<u>Topics</u>

1. *CSP* — Communicating Sequential Processes and
   FDR — Failures-Divergence Refinement

   /usr/local/pub-bkb/fdr/fdr.2.25/

2. *Promela* — Process or Protocol Meta Language and
   Spin

   http://netlib.bell-labs.com/netlib/spin/whatispin.html

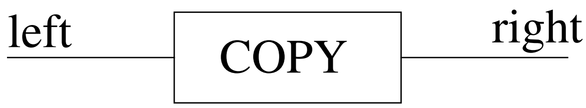
3. *Timed-Automata* and
   Kronos

   http://www-verimag.imag.fr//TEMPORISE/kronos/

1. CSP and FDR

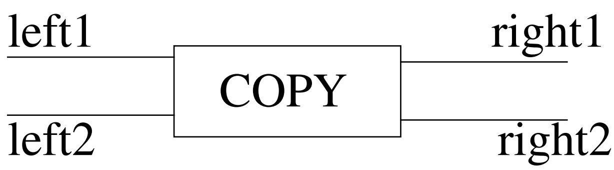
CSP through examples

Example 1: COPY



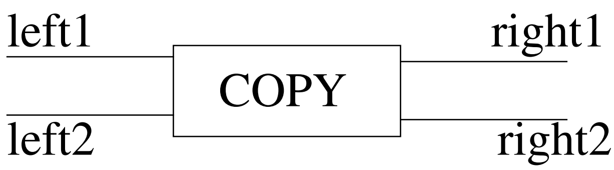
COPY = left $\rightarrow$ right $\rightarrow$ COPY,   or
COPY = left?x $\rightarrow$ right!x $\rightarrow$ COPY

Example 2: COPY2



COPY2 = left1 $\rightarrow$ right1 $\rightarrow$ COPY2
[]
left2 $\rightarrow$ right2 $\rightarrow$ COPY2

Example 3: COPYN

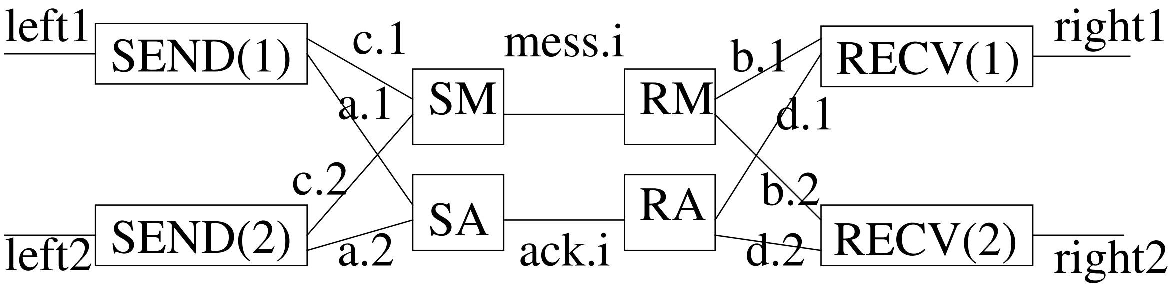


COPYN = left1 $\rightarrow$ right1 $\rightarrow$ COPY2
|~|
left2 $\rightarrow$ right2 $\rightarrow$ COPY2

Example 4: COPYP and SYS

COPY(i) = left.i $\rightarrow$ right.i $\rightarrow$ COPY(i)

COPYP = COPY(1) ||| COPY(2)



SEND(i) = left.i → c.i → a.i → SEND(i)

RECV(i) = b.i → right.i → d.i → RECV(i)

SM = c.1 → mess.1 → SM [] c.2 → mess.2 → SM

RM = mess.1 → b.1 → RM [] mess.2 → b.2 → RM

RA = d.1 → ack.1 → RA [] d.2 → ack.2 → RA

SA = ack.1 → a.1 → SA [] ack.2 → a.2 → SA

```
SYS = (((SEND(1) ||| SEND(2))
            [| {| a, c |} |]
      ((SM [|{|mess|}|] RM) ||| (SA [|{|ack|}|] RA)))
            [|{| b, d |}|]
      (RECV(1) ||| RECV(2)))
    \ {| a, b, c, d, mess, ack |}
```

Model-Checking with FDR

assert COPYP :[deadlock free [F]]
assert COPYP :[livelock free [FD]]

assert SYS :[deadlock free [F]]
assert SYS :[livelock free [FD]]

assert COPY [FD= SYS

2. Promela and Spin

Promela through examples

Example 1: Semaphore

```
#define p 0
#define v 1

chan sema = [0] of { bit };

proctype dijkstra()
{    bit count = 1;
     do
     :: (count==1) ->
          sema!p; count = 0;
     :: (count==0) ->
          sema?v; count = 1
     od
}

proctype user()
{    do
     :: sema?p;
          /* critical section */
          sema!v;
          /* non-critical section */
     od
}

init
{    run dijkstra();
     run user(); run user(); run user()
}
```

Example 2: Peterson's algorithm

```
byte turn, flag[2];
byte ncrit;

active [2] proctype user()
{     assert(_pid == 0 || _pid == 1);
again:
      flag[_pid] = 1;
      turn = _pid;
      (flag[1 - _pid] == 0 || turn == 1 - _pid);
      ncrit++;
      assert(ncrit == 1);      /* critical section */
      ncrit--;
      flag[_pid] = 0;
      goto again
}
```

Model-Checking with Spin

- Simulator

```
shi@triton% spin peterson
```

- Analyzer

```
shi@triton% spin -a peterson
shi@triton% wc pan.[bchmt]
    31     99     579 pan.b
  5071  16577  109079 pan.c
   336   1056    6905 pan.h
    78    337    2485 pan.m
   514   1834   11845 pan.t
  6030  19903  130893 total
```

- Exhaustive Search

    shi@triton% cc -o run pan.c
    shi@triton% run

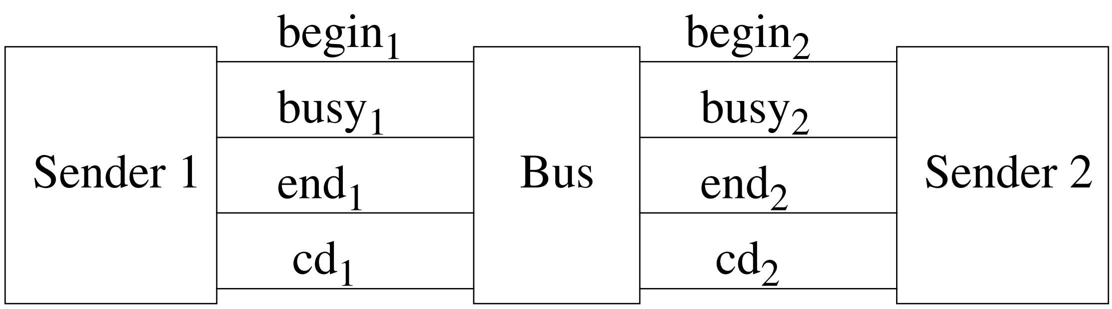
```
hint: this search is more efficient if pan.c is compiled -DSAFETY
(Spin Version 3.2.3 -- 1 August 1998)
        + Partial Order Reduction

Full statespace search for:
        never-claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid endstates       +

State-vector 20 byte, depth reached 32, errors: 0
      55 states, stored
      44 states, matched
      99 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.493   memory usage (Mbyte)

unreached in proctype user
        line 20, state 10, "-end-"
        (1 of 10 states)
```

3. Timed-Automata and Kronos

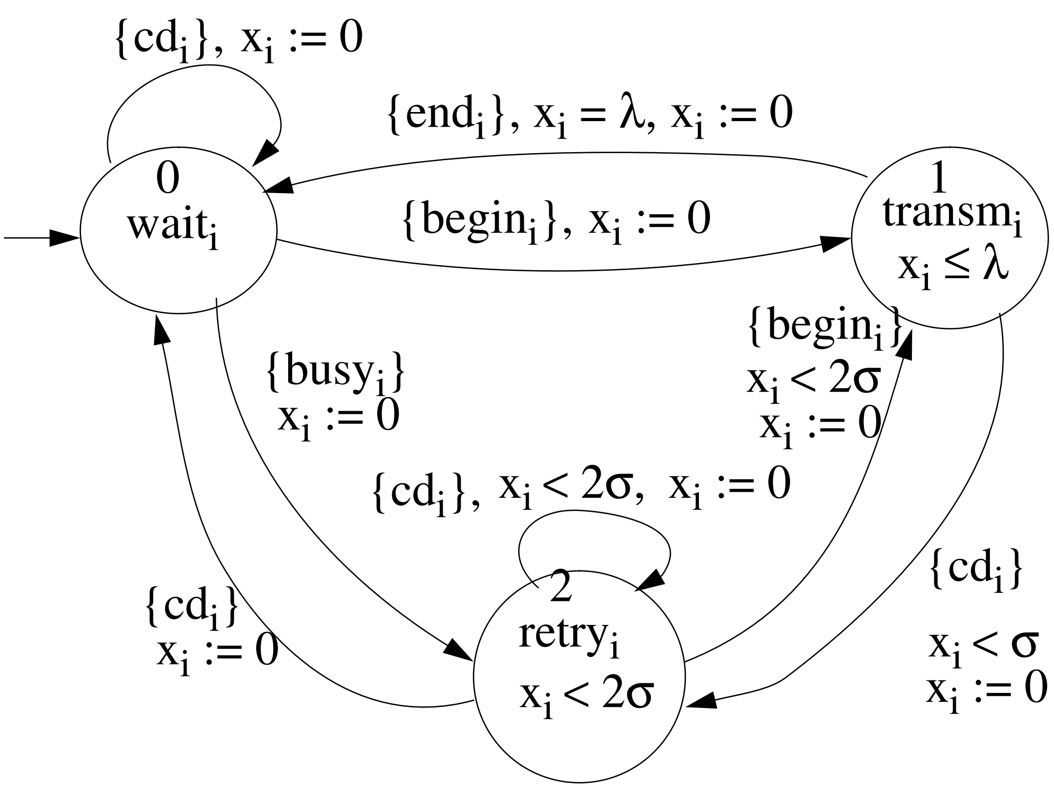
Timed-Automata through an example

Example: CSMA/CD — Carrier Sense, Multiple Access with Collision Detection

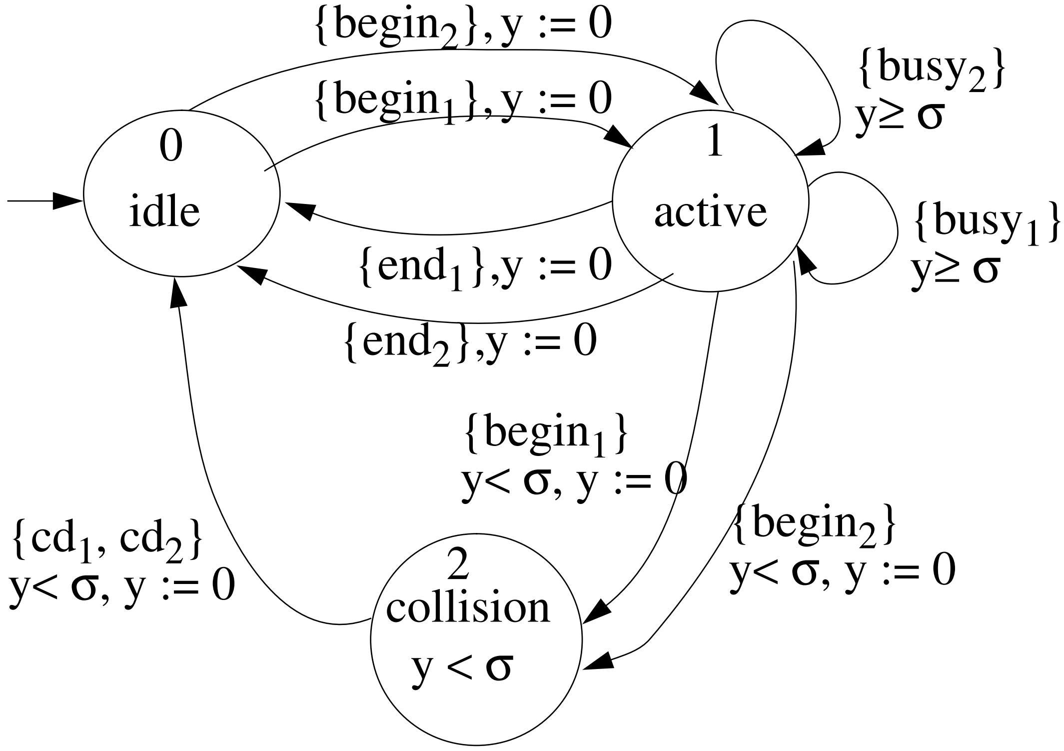
Structure of the system



Timed automata of the senders



Timed automata of the bus



Kronos input files

/* send1.tg */
#locs 3
#trans 9
#clocks X1
#sync BEGIN1 END1 BUSY1 CD1

loc: 0
prop: WAIT1
invar: TRUE
trans:
TRUE => BEGIN1; X1:=0; goto 1
TRUE => BUSY1; X1:=0; goto 2
TRUE => CD1; X1:=0; goto 2
TRUE => CD1; X1:=0; goto 0

```
loc: 1
prop: TRANSM1
invar: X1<=808
trans:
X1=808 => END1; X1:=0; goto 0
X1<26 => CD1; X1:=0; goto 2

loc: 2
prop: RETRY1
invar: X1<=52
trans:
X1<=52 => BEGIN1; X1:=0; goto 1
X1<=52 => BUSY1; X1:=0; goto 2
X1<=52 => CD1; X1:=0; goto 2


/* bus.tg */
#locs 3
#trans 9
#clocks Y
#sync BEGIN1 BEGIN2 END1 END2
          BUSY1 BUSY2 CD1 CD2

loc: 0
prop: IDLE
invar: TRUE
trans:
TRUE => BEGIN1; Y:=0; goto 1
TRUE => BEGIN2; Y:=0; goto 1

loc: 1
prop: ACTIVE
invar: TRUE
trans:
```

```
Y<26 => BEGIN1; Y:=0; goto 2
Y>=26 => BUSY1; ; goto 1
TRUE => END1; Y:=0; goto 0
Y<26 => BEGIN2; Y:=0; goto 2
Y>=26 => BUSY2; ; goto 1
TRUE => END2; Y:=0; goto 0

loc: 2
prop: COLLISION
invar: Y<26
trans:
Y<26 => CD1 CD2; ; goto 0
```

```
/* nz.tctl — specification of non-zenoness property:
      A state is called Non-Zeno if whatever the system does at
      the state, it does not prevent time to diverge. */

      init impl ab ( ed{=1} true )
```

Model-Checking with Kronos

- Product automaton

```
shi@triton% kronos -out csma.tg bus.tg send1.tg send2.tg

kronos: release 2.4.3 (sun5) date Tue Oct  18 15:00:25 MET DST 1999
kronos: file bus.kro already exists
kronos: reading file bus.kro...
kronos: file send1.kro already exists
kronos: reading file send1.kro...
kronos: updating file send2.kro...
kronos: begin syntax analysis of send2.tg...
kronos: end syntax analysis of send2.tg...
kronos: file send2.[kro,lab,prop,reg] updated
kronos: reading file send2.kro...
```

Building synchronization tables...
Generating parallel product graph...
Dumping output... files csma.[tg,kro,lab,prop,reg] created.

- Proof of Non-Zenoness

shi@triton% kronos -v -forw csma.tg nz.tctl

```
kronos: release 2.4.3 (sun5) date Tue Oct  18 15:00:25 MET DST 1999
kronos: updating file csma.kro...
kronos: begin syntax analysis of csma.tg...
kronos: end syntax analysis of csma.tg...
kronos: file csma.[kro,lab,prop,reg] updated
kronos: reading file csma.kro...
kronos: begin evaluation of nz.tctl
>>  init
<<  init
>>  (ED{=1}true)
  >>  true
  <<  true
iter #: 1,2,3,4
<<  (ED{=1}true)
kronos: begin forward analysis
kronos: file _path.reach dumped
kronos: file _trace.reach dumped
kronos: 5 simulation states generated
kronos: 4 simulation transitions generated
kronos: Invariance *** FALSE ***
kronos: end evaluation of nz.tctl
kronos: compacting
-----------------------------------------------------------------------
kronos: fixpoint          : system   0.000s * user   0.020s * #iterations 4
kronos: compact time      : system   0.000s * user   0.000s *
kronos: forward analysis  : system   0.010s * user   0.020s *
kronos: total time        : system   0.010s * user   0.050s *
-----------------------------------------------------------------------
```

- Counter-example

shi@triton% vi _path.reach

```
location: 0
propositions: WAIT2 WAIT1 IDLE
constraint: TRUE
transition: TRUE => BEGIN1 ; RESET{ Y X1 } ; goto 1

location: 1
propositions: WAIT2 TRANSM1 ACTIVE
constraint: X1<=808 and Y=X1 and Y<=X2
transition: Y<26 and X1<=808 => BEGIN2 ; RESET{ Y X2 } ; goto 3

location: 3
propositions: TRANSM1 TRANSM2 COLLISION
constraint: Y<26 and Y<=X1 and X1<Y+26 and Y=X2
```