CONPASU-tool:
A Concurrent Process Analysis Support tool based on Symbolic Computation

Yoshinao Isobe （磯部 祥尚）

Information Technology Research Institute

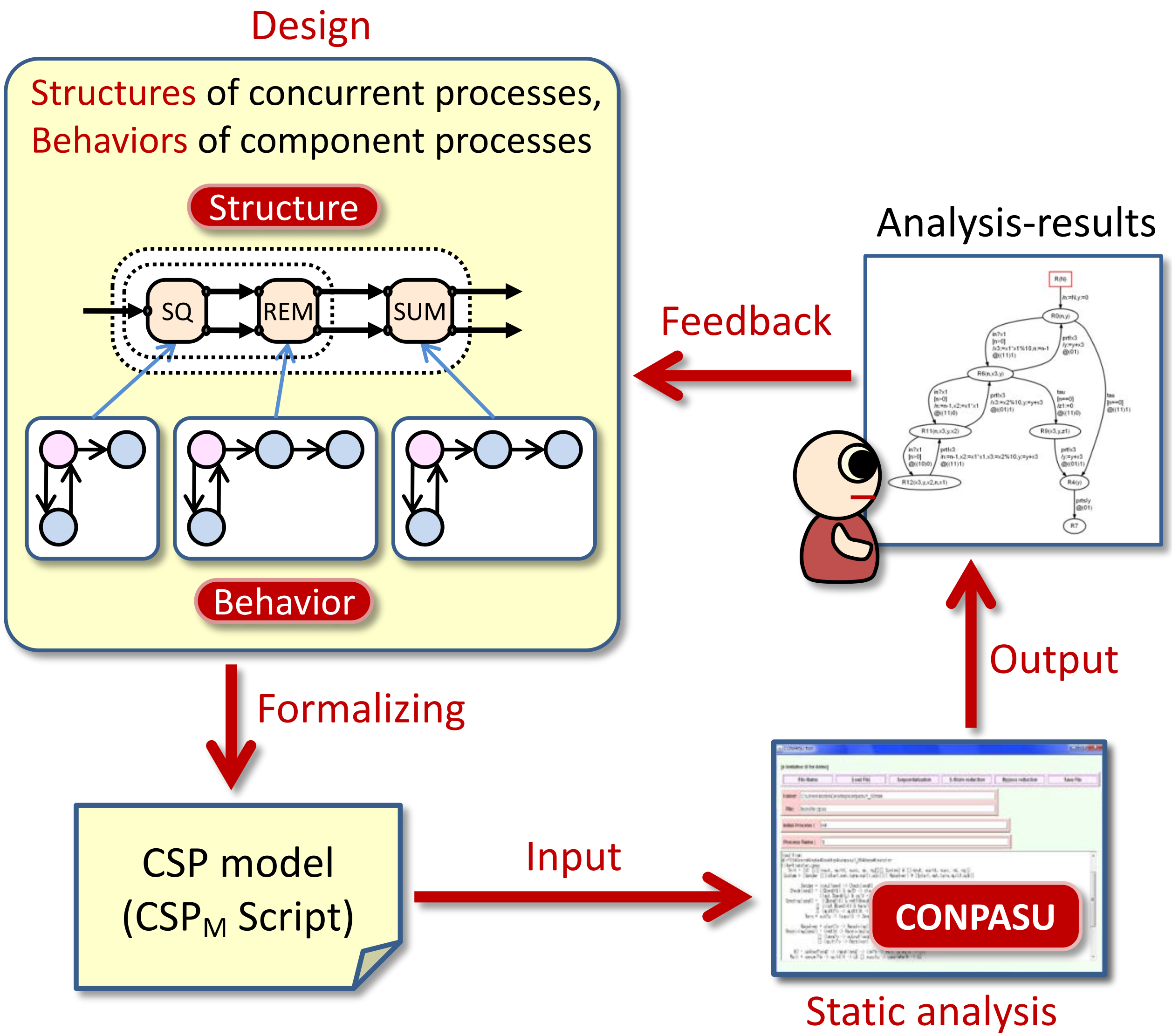
AIST, Japan

CPA 2011 (21 June 2011)

0:30

Contents

CONAPSU is a static analysis tool of concurrent processes.



2:00

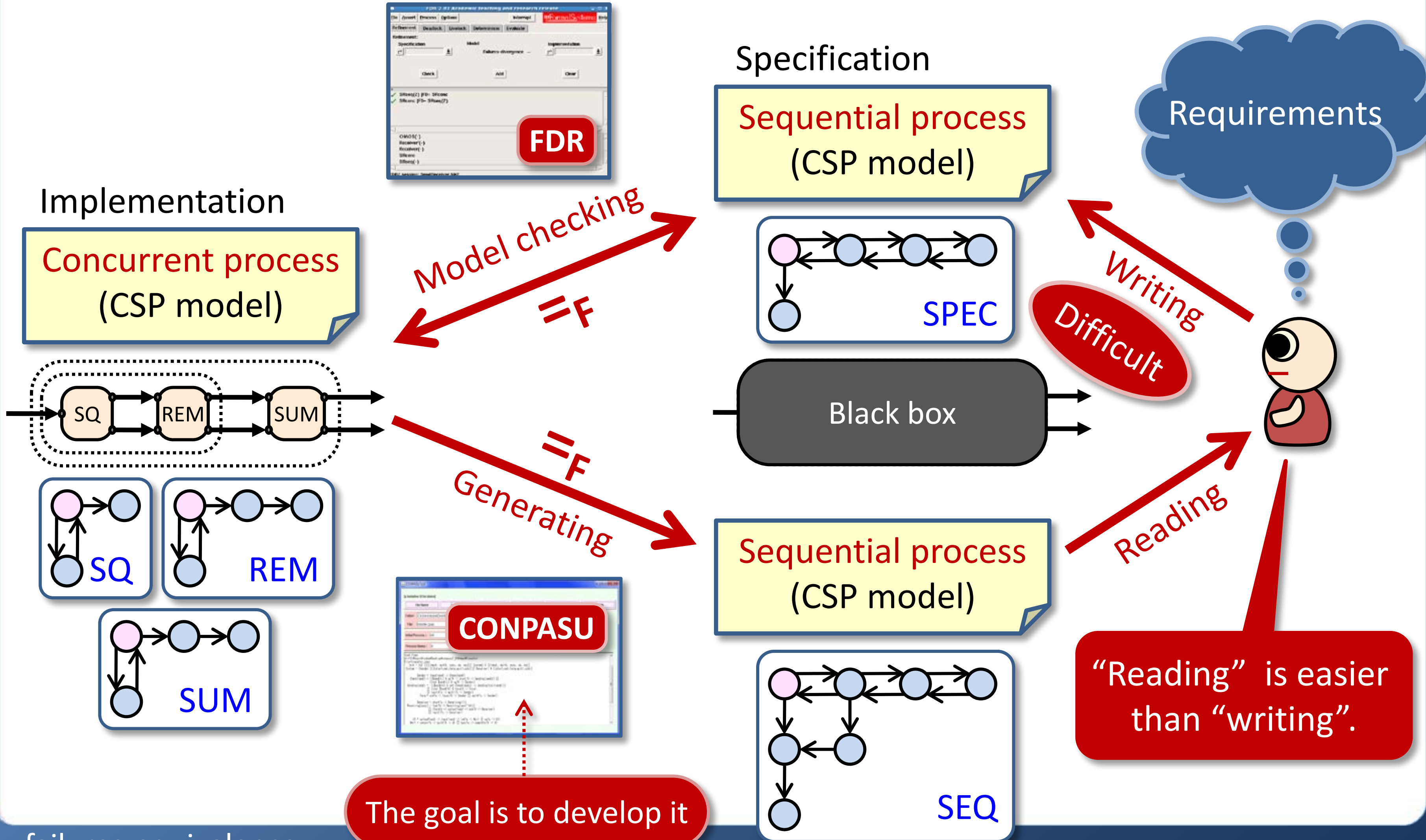
CSP$_M$: the machine readable dialect of CSP used in FDR

Introduction

- Motivation
- CONPASU

Motivation

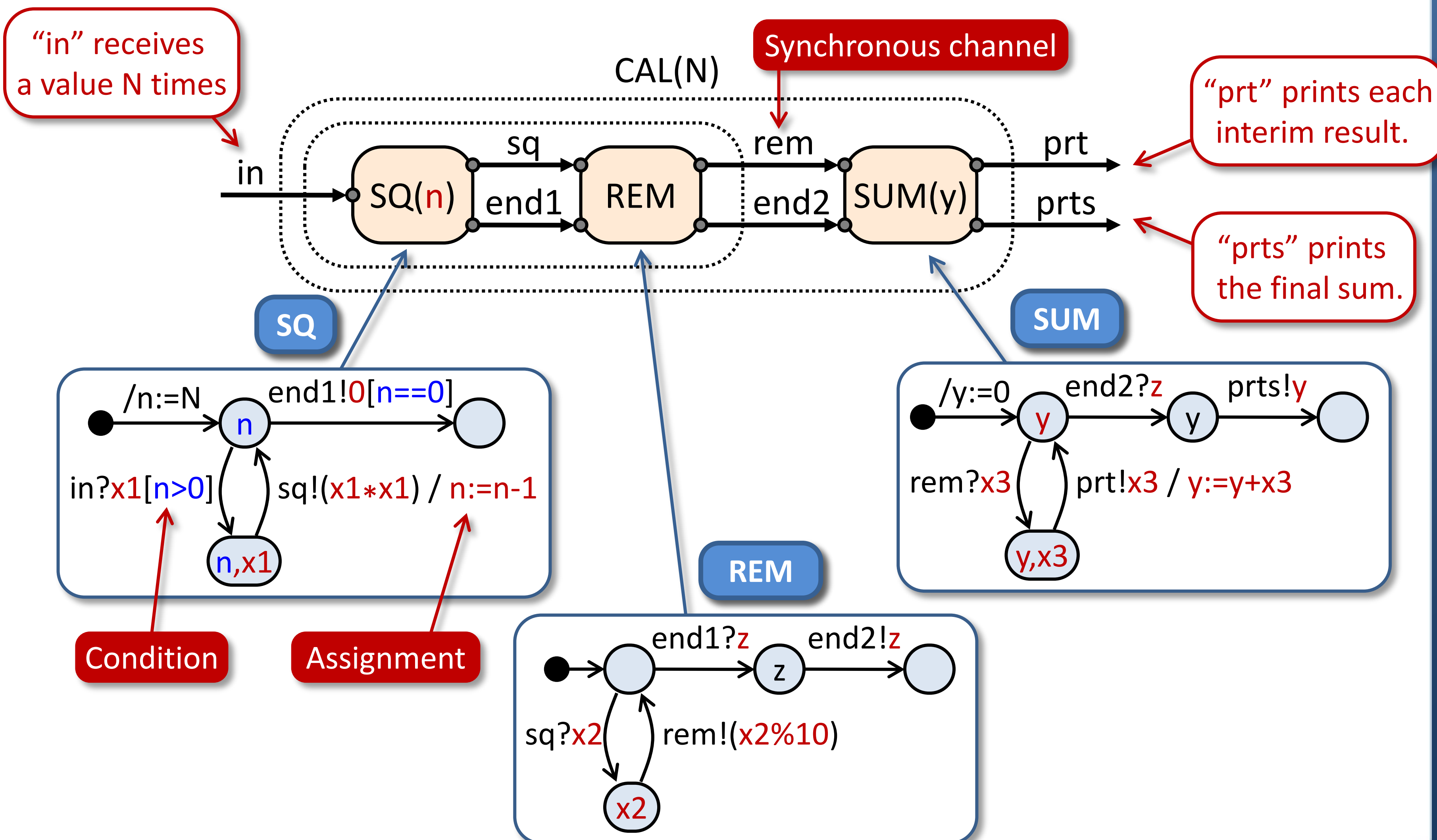
- How can we see behaviors of concurrent processes?



=F : failures-equivalence

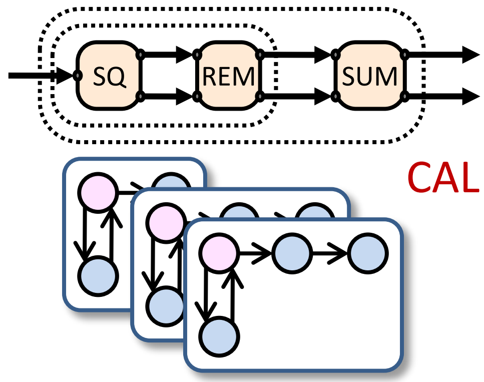
CAL: An example of concurrent process

- CAL: a concurrent process which consists of 3 processes with synchronous channels

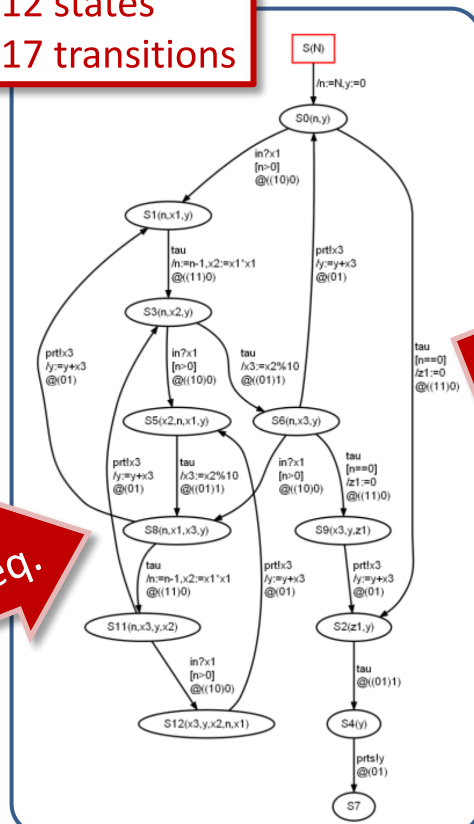


The analysis method of CONPASU (outline)

[step 1] A transition graph is generated from a given CSP model (sequentialization).
[step 2] Needless internal-transitions are bypassed (state-reduction).

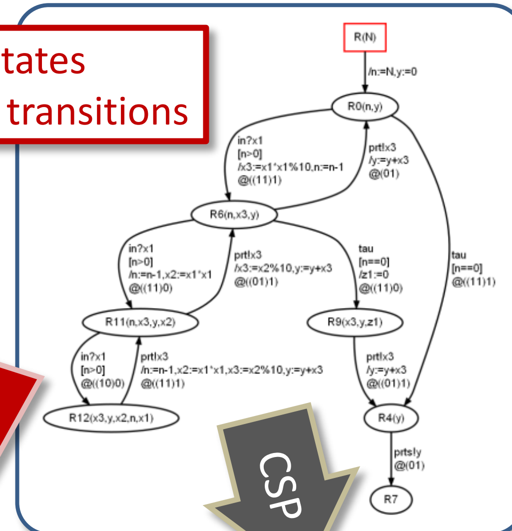


12 states
17 transitions



Red.

7 states
10 transitions



CSP

Structure

CAL(N) = (SQREM(N) [|{|rem,end2|}|] SUM(0))
\ {|rem,end2|}
SQREM(n) = (SQ(n) [|{|sq,end1|}|] REM)
\ {|sq,end1|}

Behavior

SQ(n) = ((n>0) & in?x1 -> sq!(x1*x1) -> SQ(n-1))
[] ((n==0) & end1!0 -> STOP)
REM = sq?x2 -> rem!(x2%10) -> REM
[] end1?z1 -> end2!z1 -> STOP
SUM(y) = rem?x3 -> prt!x3 -> SUM(y+x3)
[] end2?z2 -> prts!y -> STOP

Concurrent process (CSP model)

Seq.

SEQ(N) = SEQ0(N,0) \ {|tmp|}
SEQ0(n,y) = (n>0) & in?x1 -> SEQ6(n-1,x1*x1%10,y)
[] (n==0) & tmp!0 -> SEQ4(y)
SEQ4(y) = prts!(y) -> SEQ7
SEQ6(n,x3,y) = (n>0) & in?x1 -> SEQ11(n-1,x3,y,x1*x1)
[] (n==0) & tmp!0 -> SEQ9(x3,y,0)
[] prt!(x3) -> SEQ0(n,y+x3)
SEQ7 = STOP
SEQ9(x3,y,z1) = prt!(x3) -> SEQ4(y+x3)
SEQ11(n,x3,y,x2) = (n>0) & in?x1 -> SEQ12(x3,y,x2,n,x1)
[] prt!(x3) -> SEQ6(n,x2%10,y+x3)
SEQ12(x3,y,x2,n,x1)
= prt!(x3) -> SEQ11(n-1,x2%10,y+x3,x1*x1)

Sequential process (CSP model)

$=_F$

Graphviz is used for display graphs.

$=_F$: (stable) failures-equivalence

Analysis method

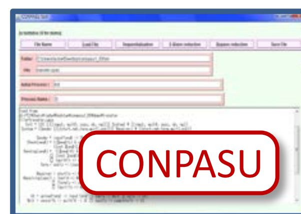
- Sequentialization
- State-reduction
- Abstraction

Sequentialization

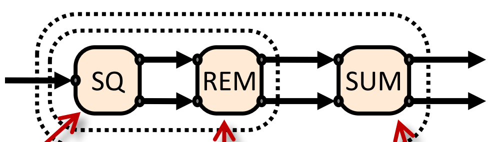
- A **symbolic** operational semantics with data-assignments and locations is used.
- Variables are not instantiated to values in symbolic semantics.
  → Many values can be folded into a variable in symbolic labeled transition graphs.
  → State-minimization is difficult (often undecidable).

The CSP model of CAL(N)

```
   CAL(N) = (SQREM(N) [|{|rem,end2|}|] SUM(0)) ¥ {|rem,end2|}
SQREM(n) = (SQ(n) [|{|sq,end1|}|] REM) ¥ {|sq,end1|}

    SQ(n) = ((n>0) & in?x1 -> sq!(x1*x1) -> SQ(n-1))
            [] ((n==0) & end1!0 -> STOP)
      REM = sq?x2 -> rem!(x2%10) -> REM
            [] end1?z1 -> end2!z1 -> STOP
   SUM(y) = rem?x3 -> prt!x3 -> SUM(y+x3)
            [] end2?z2 -> prts!y -> STOP
```



Symbolic operational semantics



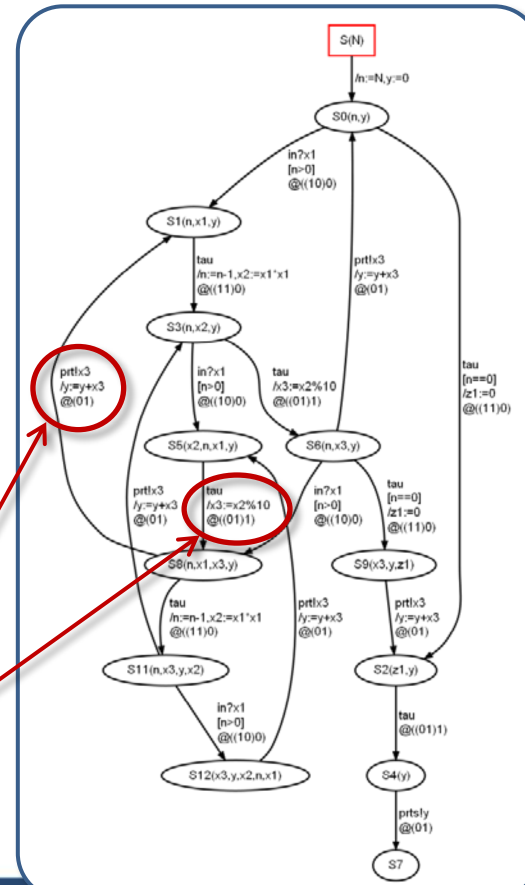
@((10)0)   @((01)0)   @(01)

Locations

prt!x3 / y:=y+x3 @ (01)

τ / x3:=x2%10 @ ((01)1)

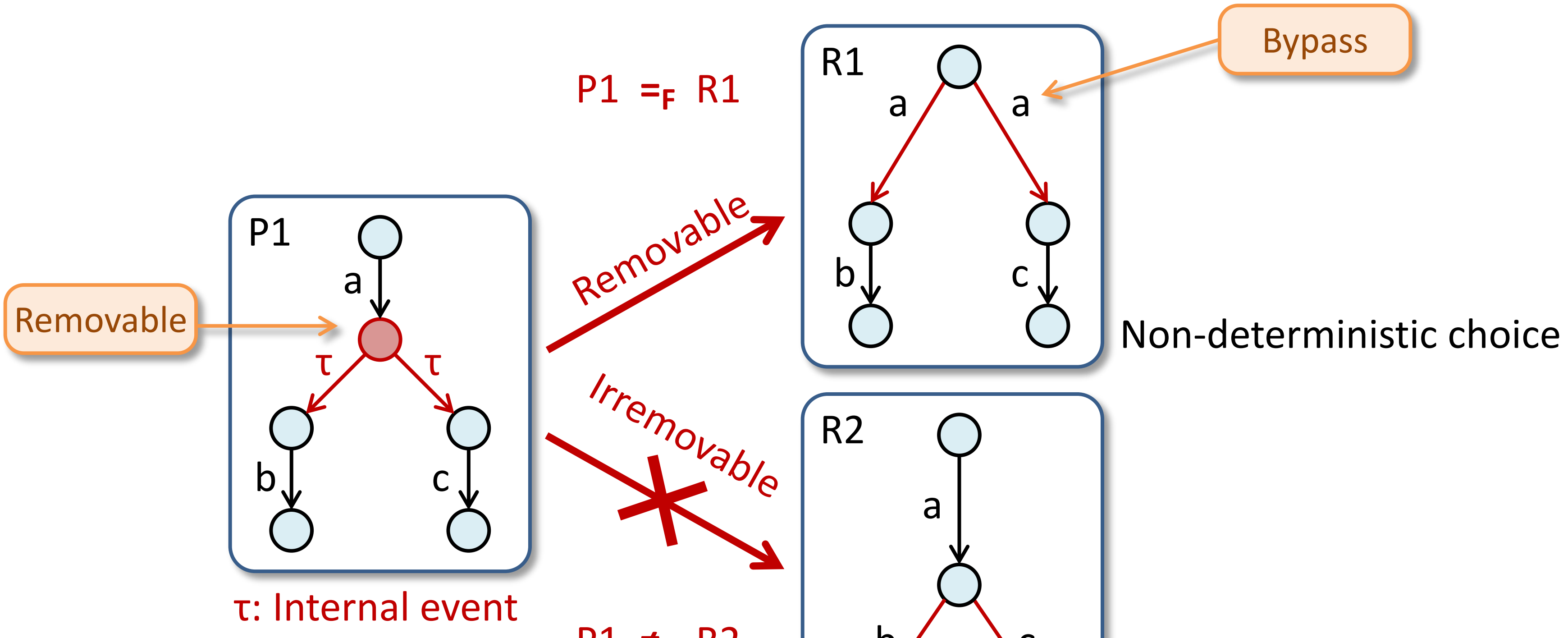
τ: Internal event



State-reduction (internal-choice)

- Needless internal transitions are bypassed with preserving the failures-equivalence $=_F$.

e.g. A removable state with non-deterministic internal transitions.
(in fact, it is more complex because conditions and assignments are considered)

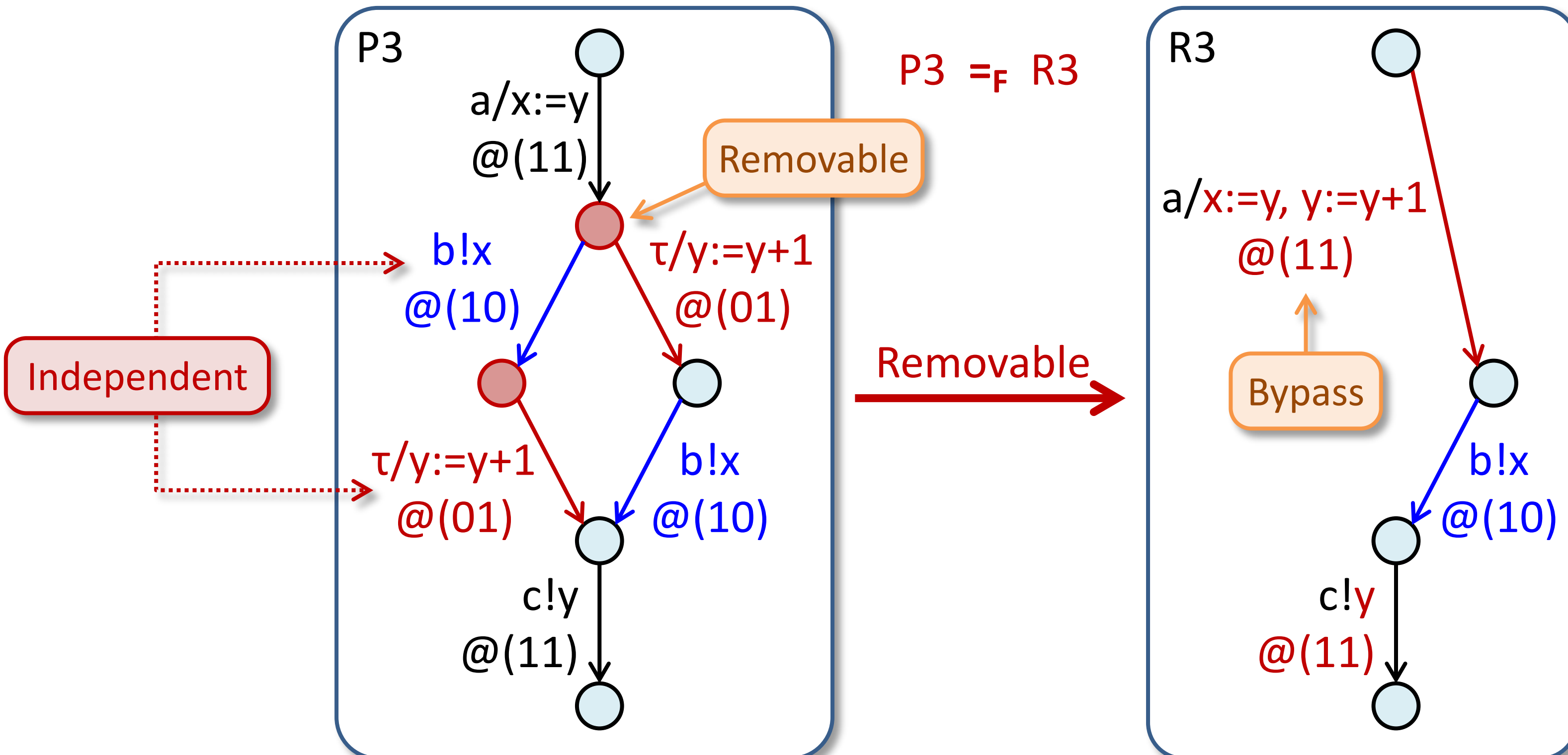


$=_F$: (stable) failures-equivalence

State-reduction (interleaving)

- Needless internal transitions are bypassed with preserving failures-equivalence $=_F$.

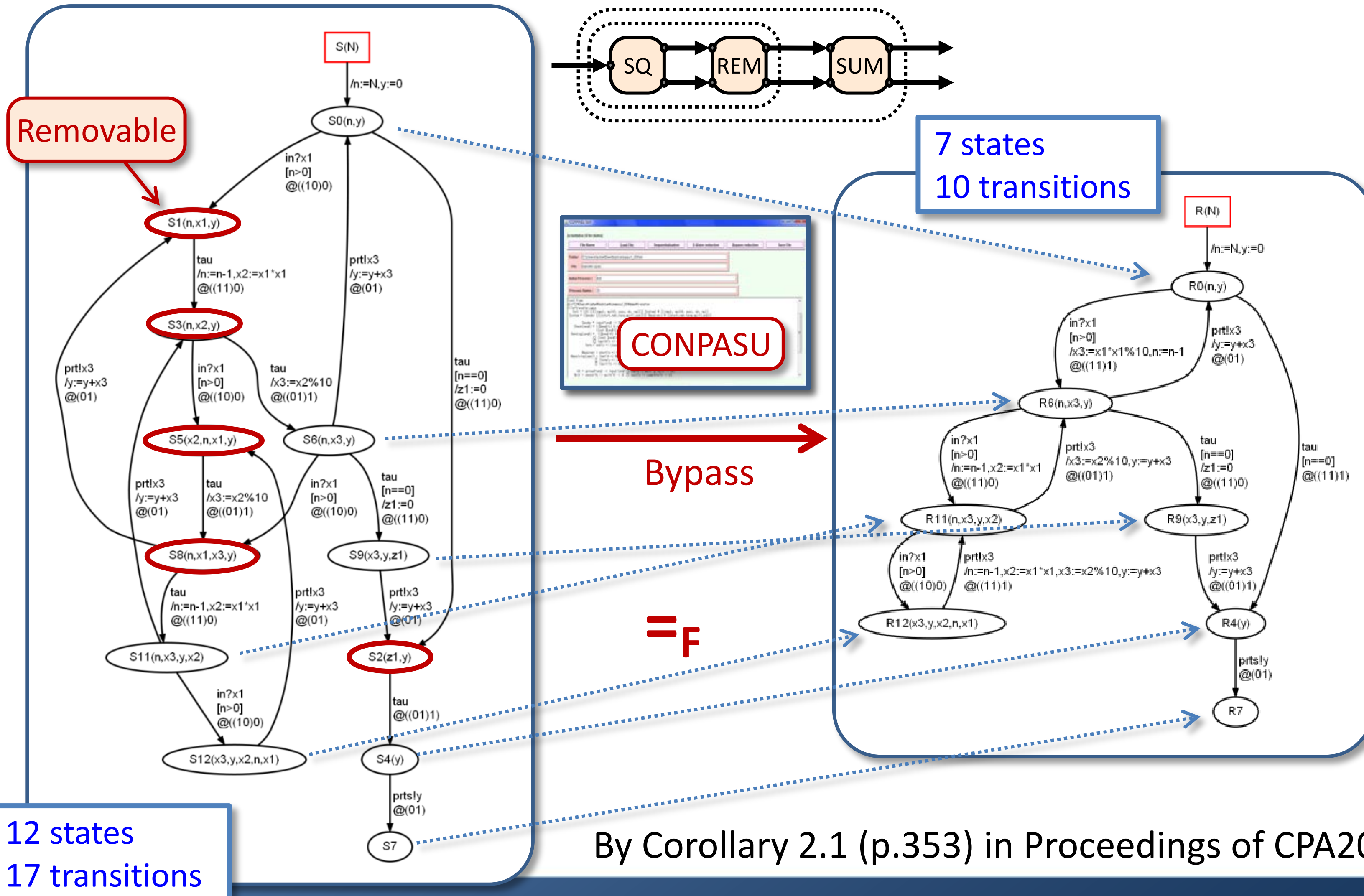
  e.g. Removable states with interleaving.



In CONPASU, locations are used for checking the independency.

State-reduction (an example)

- The removable states in the transition graph of CAL(N) and the reduced graph.

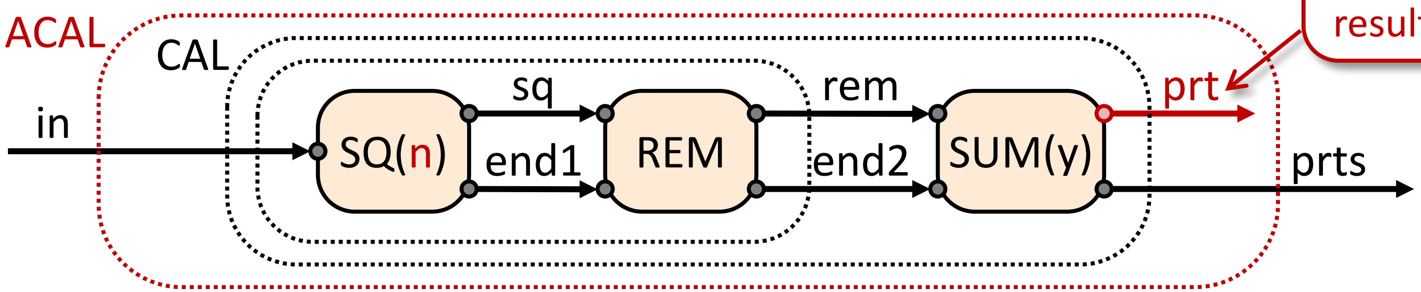


By Corollary 2.1 (p.353) in Proceedings of CPA2011

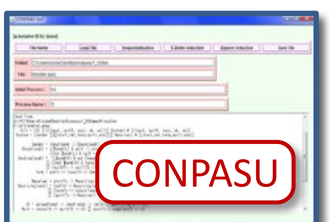
Abstraction

- Analysis by focusing on interesting channels (e.g. in and prts)

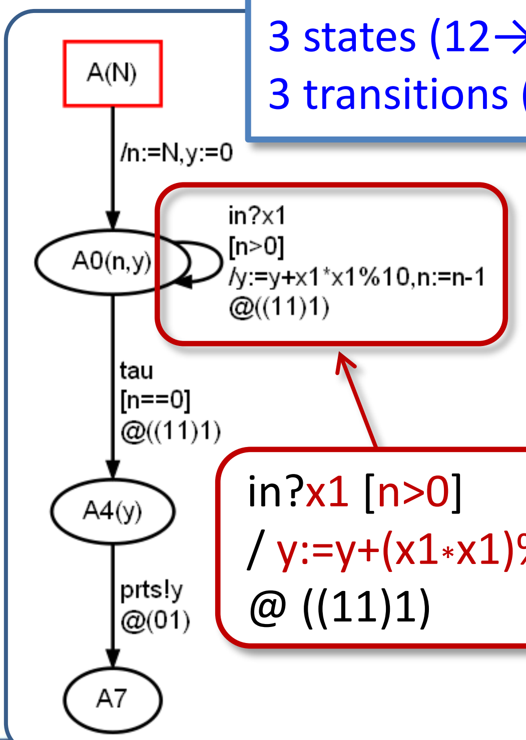
Hiding interim result on prt




```
ACAL(N) = CAL(N) \ {|prt|}
 CAL(N) = (SQREM(N) [|{|rem,end2|}|] SUM(0)) \  {|rem,end2|}
SQREM(n) = (SQ(n) [|{|sq,end1|}|] REM)  \ {|sq,end1|}

  SQ(n) = ((n>0) & in?x1 -> sq!(x1*x1) -> SQ(n-1))
          [] ((n==0) & end1!0 -> STOP)
    REM = sq?x2 -> rem!(x2%10) -> REM
          [] end1?z1 -> end2!z1 -> STOP
  SUM(y) = rem?x3 -> prt!x3 -> SUM(y+x3)
          [] end2?z2 -> prts!y -> STOP
```



Sequentialization,
State-reduction



3 states (12→3)
3 transitions (17→3)

in?x1 [n>0]
/ y:=y+(x1*x1)%10,n:=n-1
@ ((11)1)

Application

- Data-sequence transfer
- Analysis

The CSP model of TransferSys

- TransferSys is a concurrent process that consists of 3 processes: UI, Sender, and Receiver.
- Sender transfers data-sequences from UI to Receiver (it can be cancelled).

The CSP model of TransferSys

TransferSys

```
TransferSys = (UI [|{|input, quit0, succ, ok, ng|}|] Transfer)
                  \ {|input, quit0, succ, ok, ng|}

Transfer = (Sender [|{|start,net,term,quit1,ack|}|] Receiver)
                  \ {|start,net,term,quit1,ack|}
```

Structure

UI

```
UI = upload?ds -> input!ds -> (ok?a -> Wait [] ng?a -> UI)
Wait = cancel?b -> quit0!0 -> UI [] succ?u -> complete!0 -> UI
```

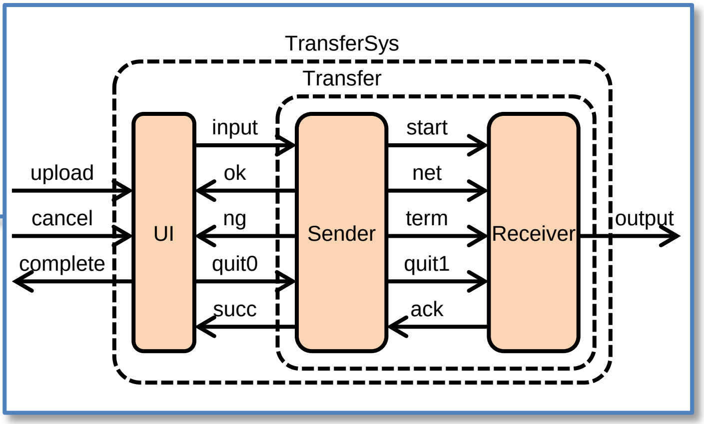
Sender

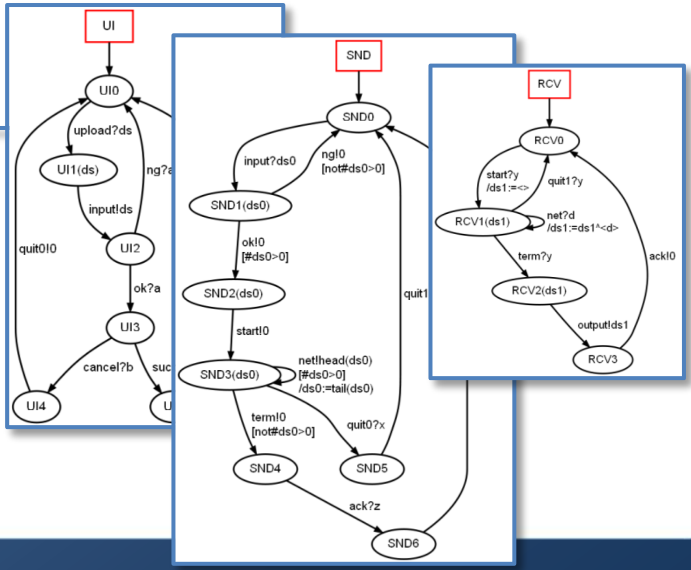
```
Sender = input?ds0 -> Check(ds0)
Check(ds0) = ((#ds0>0) & ok!0 -> start!0 -> Sending(ds0))
             [] ((not #ds0>0) & ng!0 -> Sender)
Sending(ds0) = ((#ds0>0) & net!(head(ds0)) -> Sending(tail(ds0)))
             [] ((not #ds0>0) & term!0 -> Term)
             [] (quit0?x -> quit1!0 -> Sender)
Term = ack?z -> succ!0 -> Sender
```

Receiver

```
Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
                 [] (term?y -> output!ds1 -> ack!0 -> Receiver)
                 [] (quit1?y -> Receiver)
```

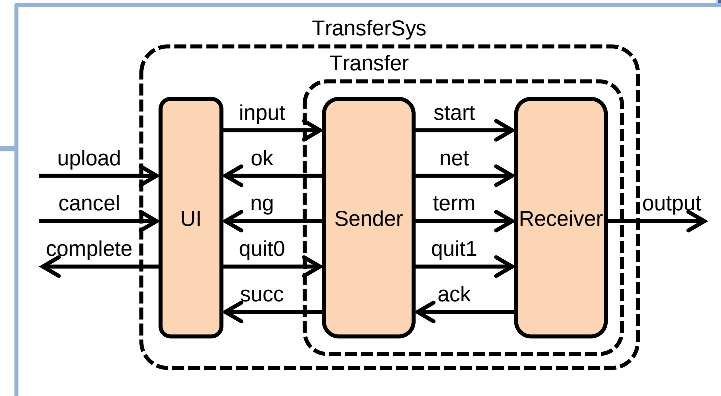
Behavior

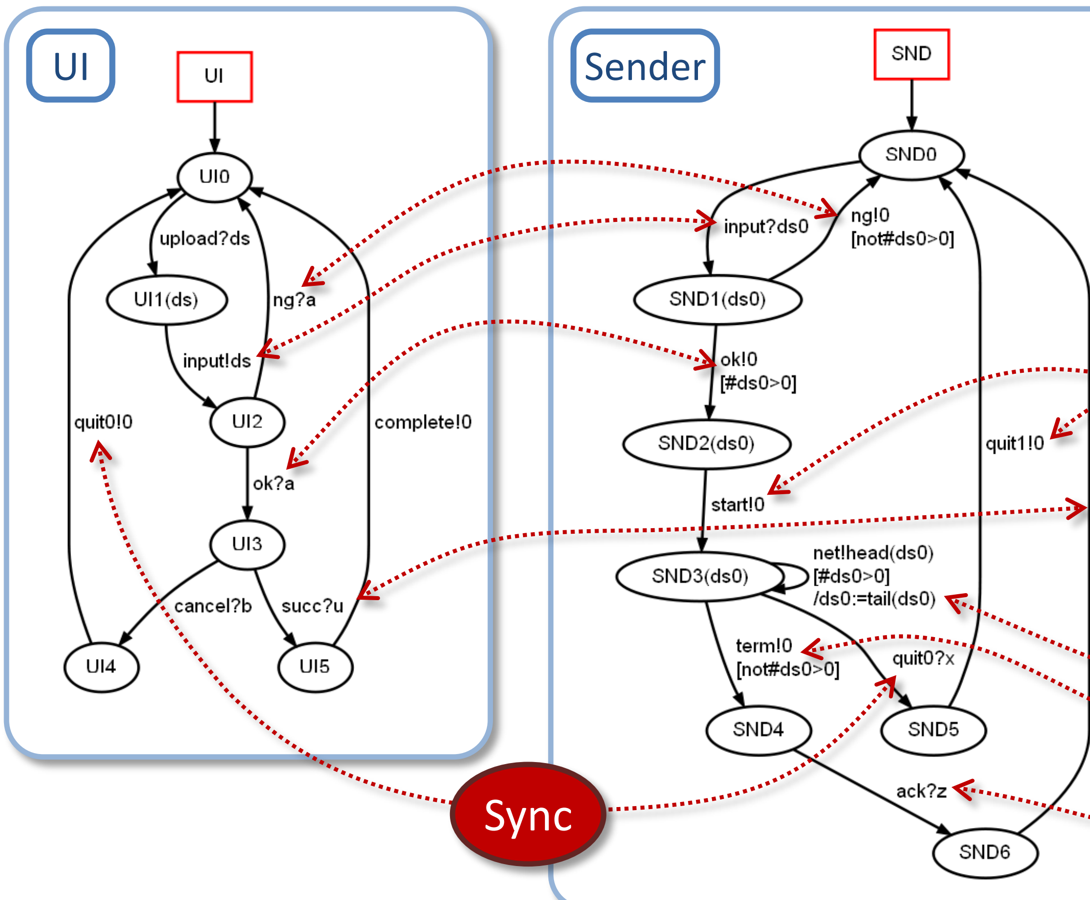


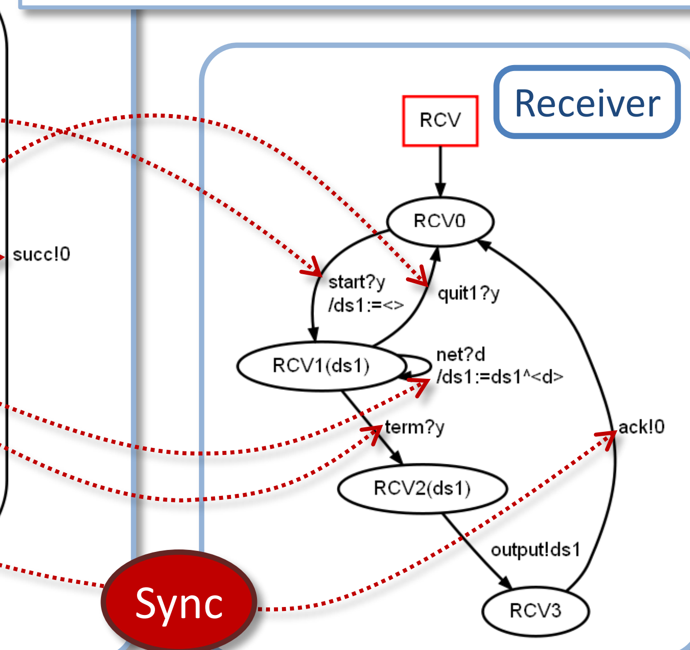


The behaviors of the 3 components

- Sender **synchronously communicates** with UI or Receiver.
  → How does **their composition** behave?







The behavior of TransferSys

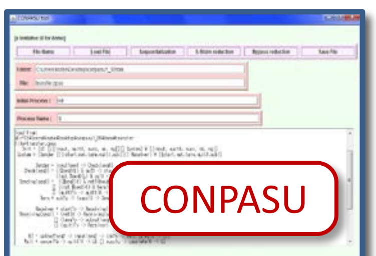
ds0 : the sequence-variable in Sender
ds1 : the sequence-variable in Receiver

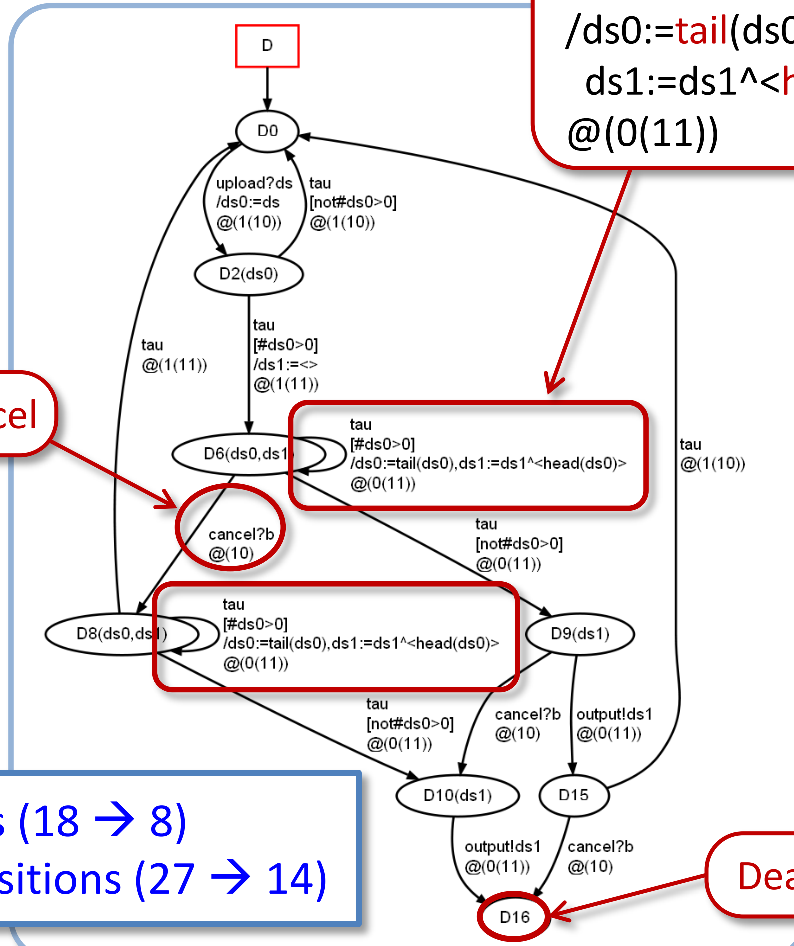
- The symbolic labeled transition graph generated by CONPASU from TransferSys

TransferSys \ {|complete |}

```
TransferSys = (UI [|{|input, quit0, succ, ok, ng|}|] Transfer)
                \ {|input, quit0, succ, ok, ng|}

Transfer = (Sender [|{|start,net,term,quit1,ack|}|] Receiver)
                \ {|start,net,term,quit1,ack|}

UI = upload?ds -> input!ds -> (ok?a -> Wait [] ng?a -> UI)
Wait = cancel?b -> quit0!0 -> UI [] succ?u -> complete!0 -> UI

Sender = input?ds0 -> Check(ds0)
Check(ds0) = ((#ds0>0) & ok!0 -> start!0 -> Sending(ds0))
             [] ((not #ds0>0) & ng!0 -> Sender)
Sending(ds0) = ((#ds0>0) & net!(head(ds0)) -> Sending(tail(ds0)))
             [] ((not #ds0>0) & term!0 -> Term)
             [] (quit0?x -> quit1!0 -> Sender)
Term = ack?z -> succ!0 -> Sender

Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
               [] (term?y -> output!ds1 -> ack!0 -> Receiver)
               [] (quit1?y -> Receiver)
```

Sequentialization,
State-reduction





τ[#ds0>0]
/ds0:=tail(ds0),
ds1:=ds1^<head(ds0)>
@(0(11))

Cancel

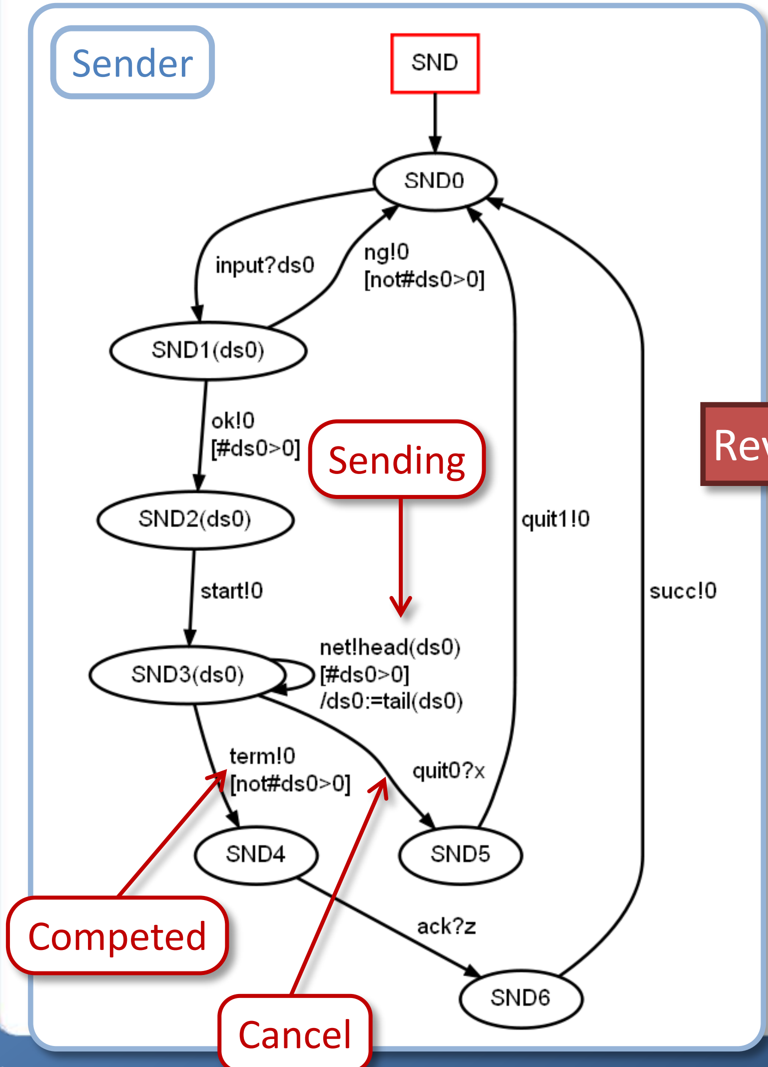
Deadlock !

8 states (18 → 8)
14 transitions (27 → 14)

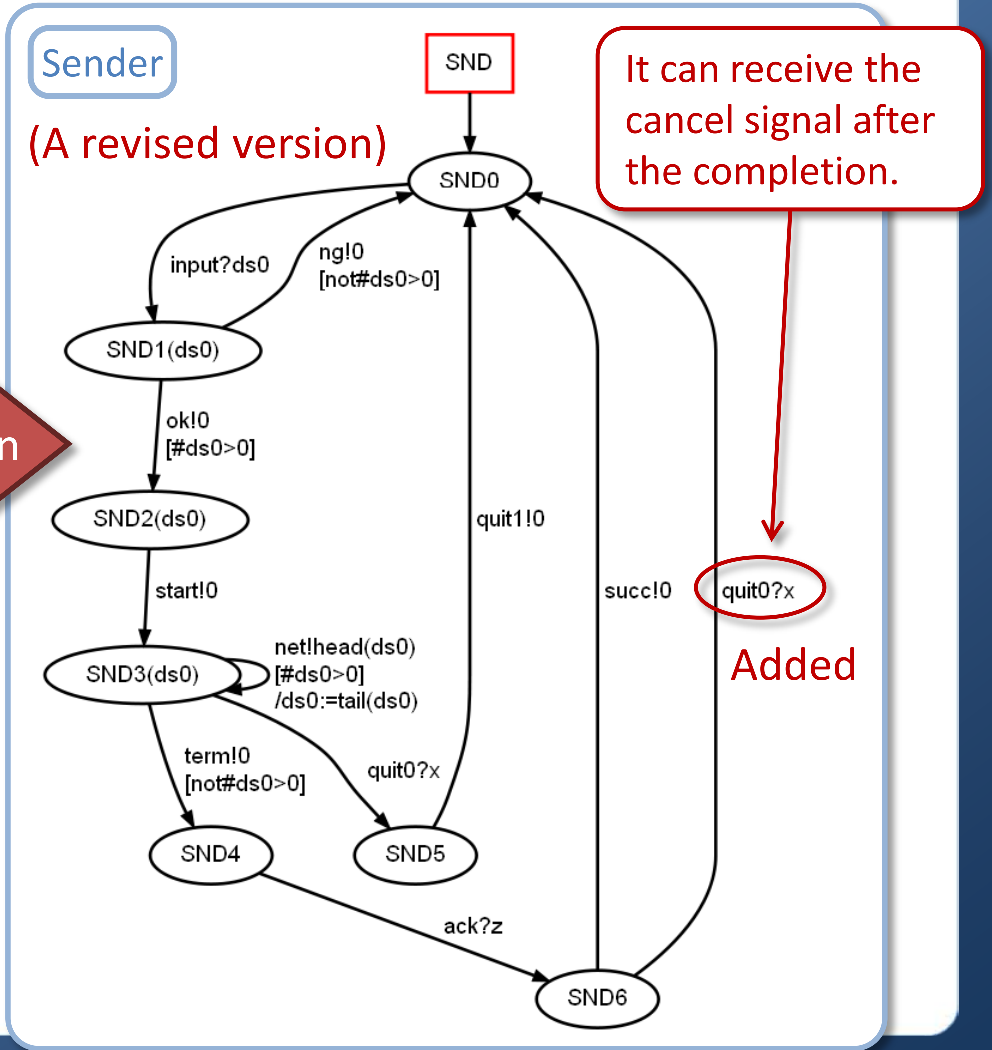
Graphviz is used for display graphs.

A revision of Sender

- A transition is added in Sender for receiving the cancel signal after transfer completion.



Revision



The behavior of the revised TransferSys

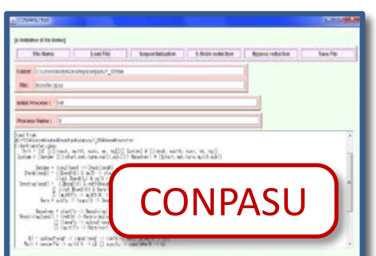
- The transition graph of the revised TransferSys.

TransferSys \ {|complete |}

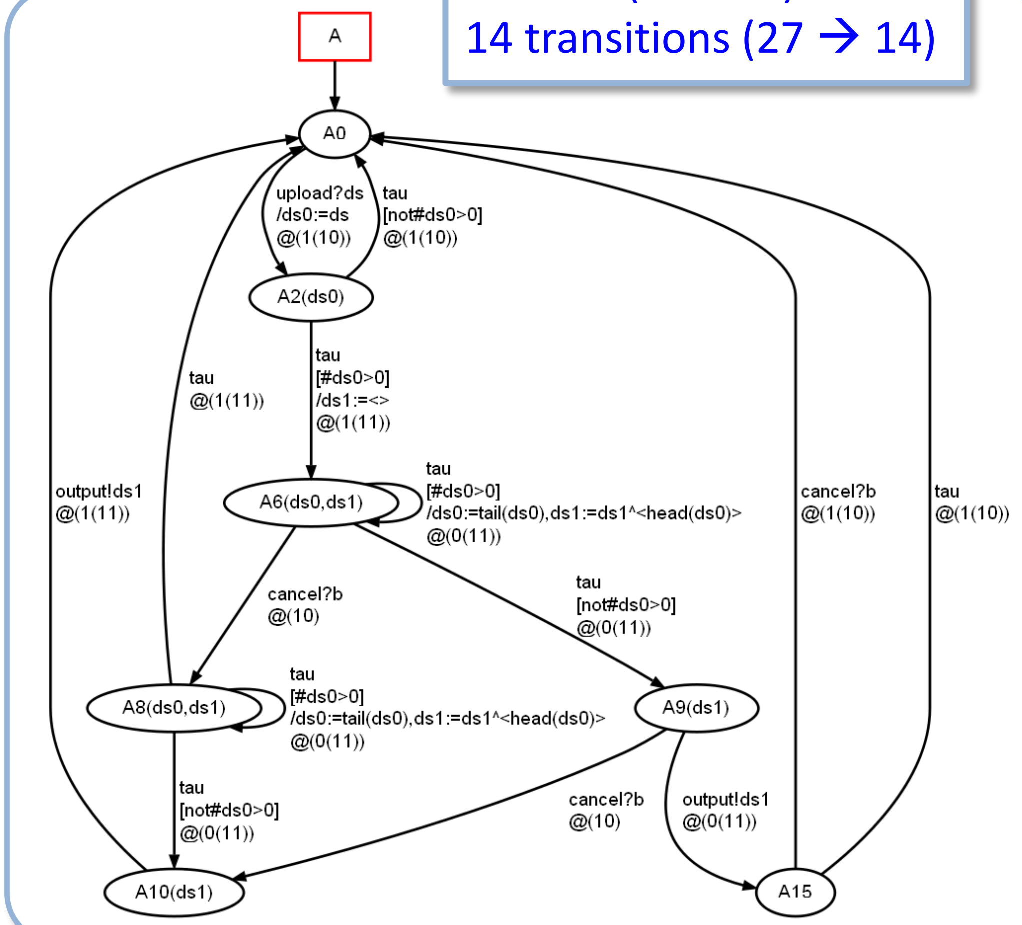
```
TransferSys = (UI [|{|input, quit0, succ, ok, ng|}|] Transfer)
              \ {|input, quit0, succ, ok, ng|}

Transfer = (Sender [|{|start,net,term,quit1,ack|}|] Receiver)
           \ {|start,net,term,quit1,ack|}

UI = upload?ds -> input!ds -> (ok?a -> Wait [] ng?a -> UI)
Wait = cancel?b -> quit0!0 -> UI [] succ?u -> complete!0 -> UI

Sender = input?ds0 -> Check(ds0)
Check(ds0) = ((#ds0>0) & ok!0 -> start!0 -> Sending(ds0))
             [] ((not #ds0>0) & ng!0 -> Sender)
Sending(ds0) = ((#ds0>0) & net!(head(ds0)) -> Sending(tail(ds0)))
             [] ((not #ds0>0) & term!0 -> Term)
             [] (quit0?x -> quit1!0 -> Sender)
Term = ack?z -> (succ!0 -> Sender [] quit0?x -> Sender)

Receiver = start?y -> Receiving(<>)
Receiving(ds1) = (net?d -> Receiving(ds1^<d>))
               [] (term?y -> output!ds1 -> ack!0 -> Receiver)
               [] (quit1?y -> Receiver)
```

Revised

Sequentialization,
State-reduction



7 states (18 → 8)
14 transitions (27 → 14)



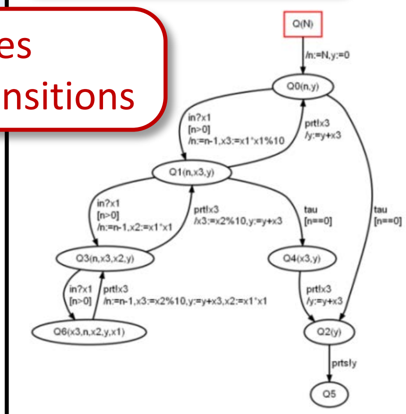
Related works

- PAT
- LTSA

PAT (Process Analysis Toolkit)

cf. CONPASU

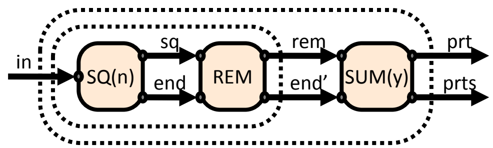
8 states
12 transitions



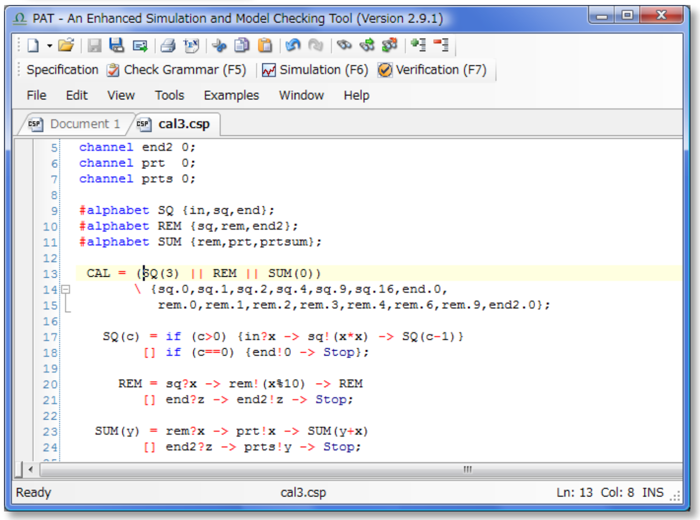
any N and any input

- PAT can display transition graphs of CSP models.
- Standard (non-symbolic) semantics is used.
  (all variables are instantiated to possible values)

CAL



PAT (GUI)

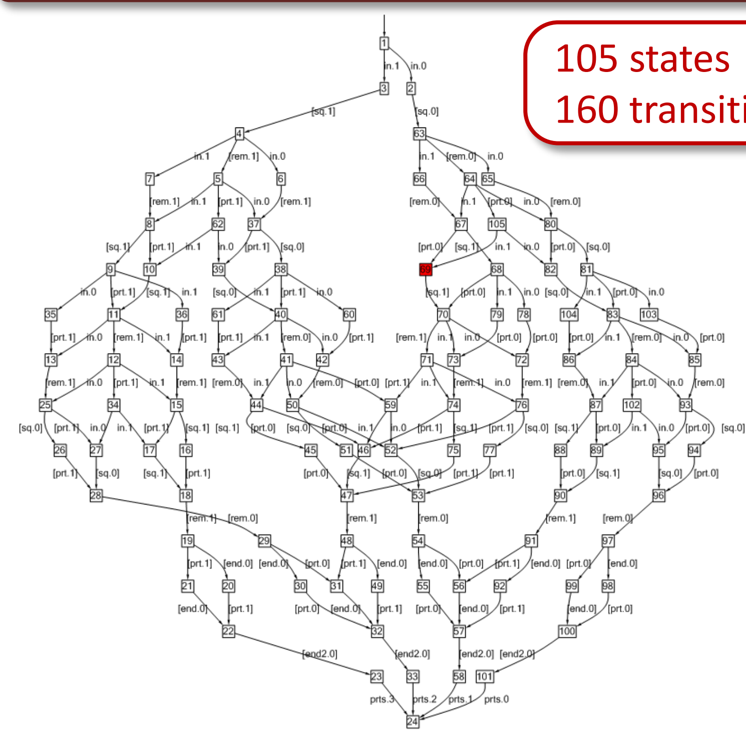


by Simulator

fix N=3 and finitize input in {0,1}

The standard transition graph of CAL(3)

105 states
160 transitions

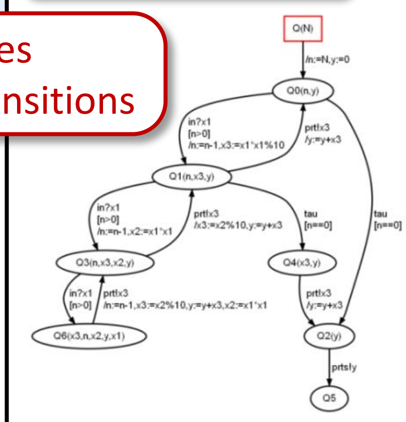


LTSA (LTS analyzer)

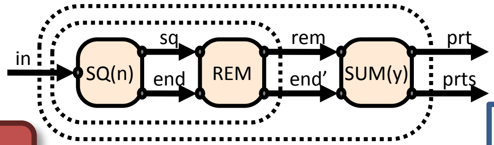
- LTSA can display minimized transition graphs.
- Standard (non-symbolic) semantics is used.
  (all variables are instantiated to possible values)

cf. CONPASU

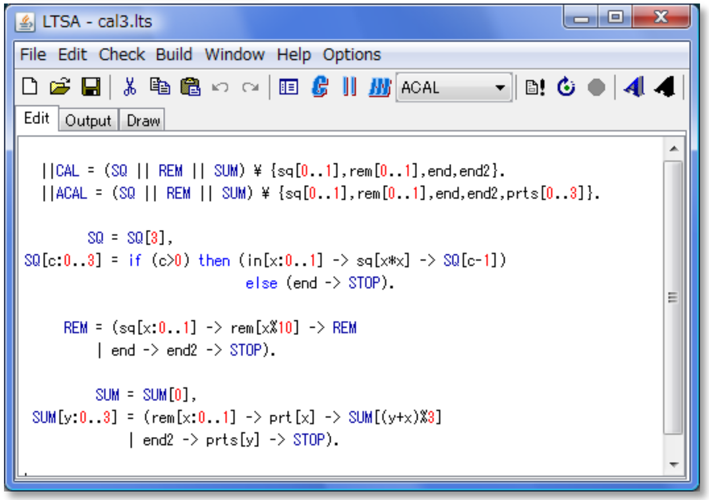
8 states
12 transitions



any N and any input



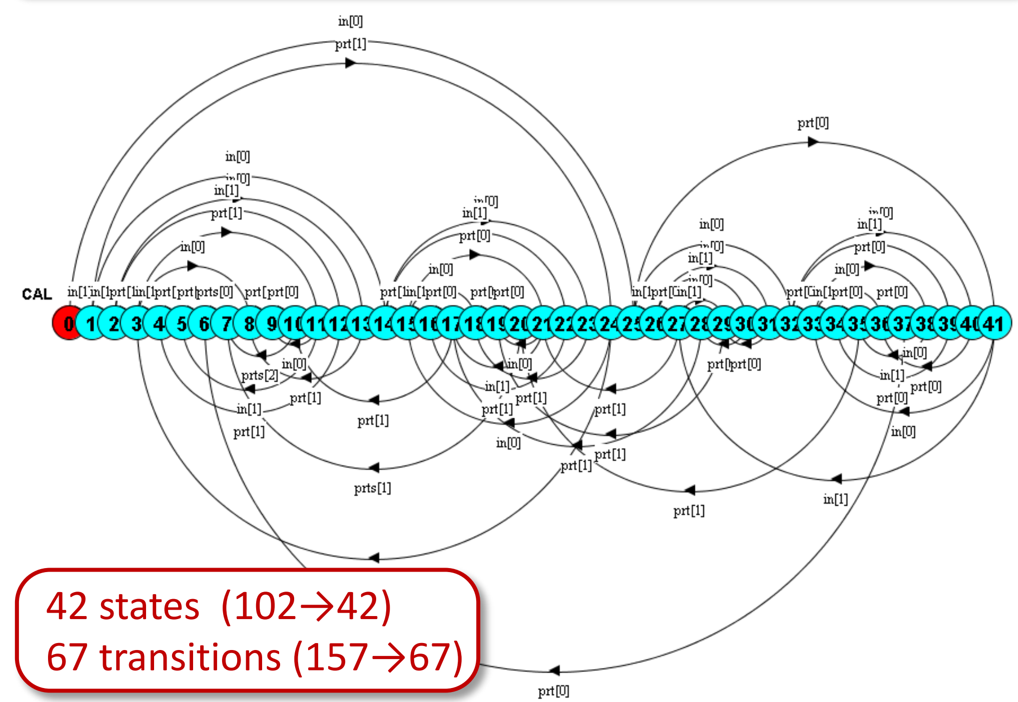
LTSA (GUI)



by Draw

fix N=3 and finitize input in {0,1}

The minimized standard transition graph of CAL(3)



42 states (102→42)
67 transitions (157→67)

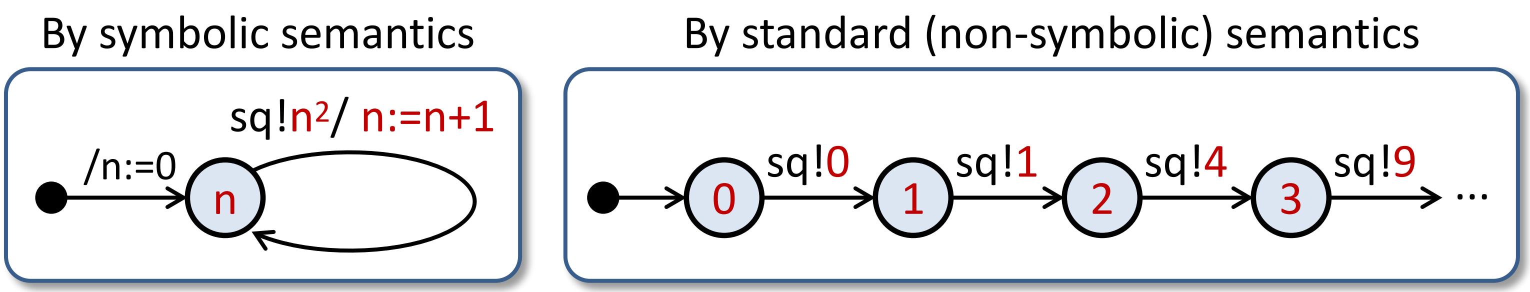
Summary

- Advantages
- Future works

Summary

- A **symbolic analysis method** and its **implementation** CONPASU have been presented.
- **The advantages[A] and disadvantages[D] of CONPASU compared with model-checkers:**
  - [A] Symbolic operational semantics is used (i.e. variables are not instantiated),
  - [A] An equal sequential process (and the graph) can be automatically generated.
  - [D] Symbolic labels are usually more complex than standard (instantiated) labels.
  - [D] Generated sequential processes are not nececssarily optimized (e.g. not minimized).

  → **CONPASU and model checker will complement each other.**



$S(n) = sq!n^2 \rightarrow S(n+1)$

- **Future works:**
  - Careful consideration about livelocks
  - Symbolic computation of data-expressions ($1+2 \neq 2+1$ in the prototype)
  - Improvement of CONPASU (Java, 6,000 lines) and evaluation of performance