Separating Functional and Timed Aspects in Transactional Abstraction Levels

Jérôme Cornet, Florence Maraninchi
and Laurent Maillet-Contoz

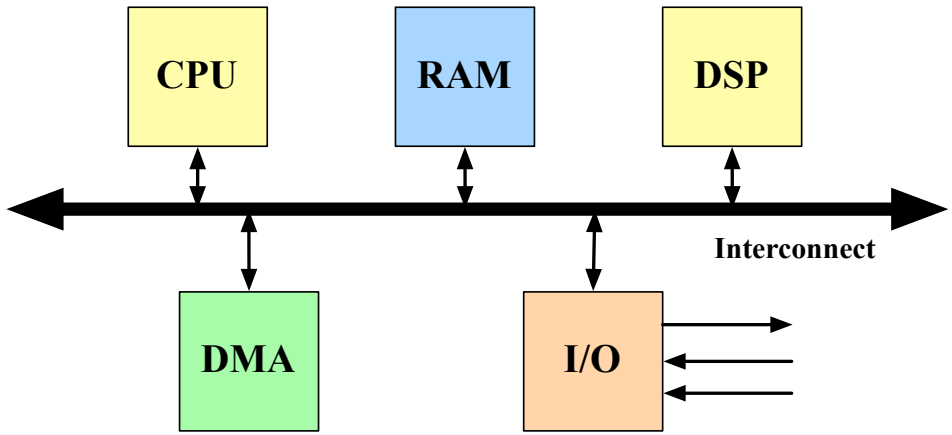
Verimag - CNRS - STMicroelectronics

Outline

Outline

What are Systems on Chip ? (SoC)

- Chips integrating all necessary electronic circuits for a "system"
- Applications : Cell phones, DVD, Set-top boxes, MP3 players, Automotive...
- Key characteristics
  - High level of integration
  - Software and hardware parts
  - Application Specific Integrated Circuits



Systems on Chip's Design

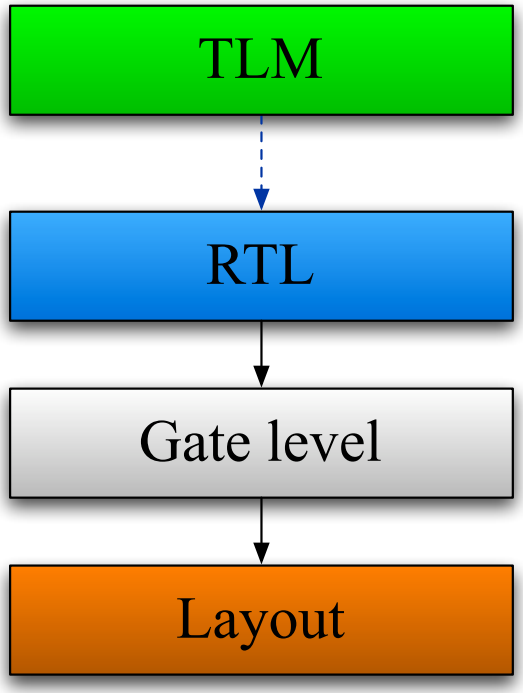
- Software: C/C++ Programming
- Hardware: Component Description at

Register Transfert Level

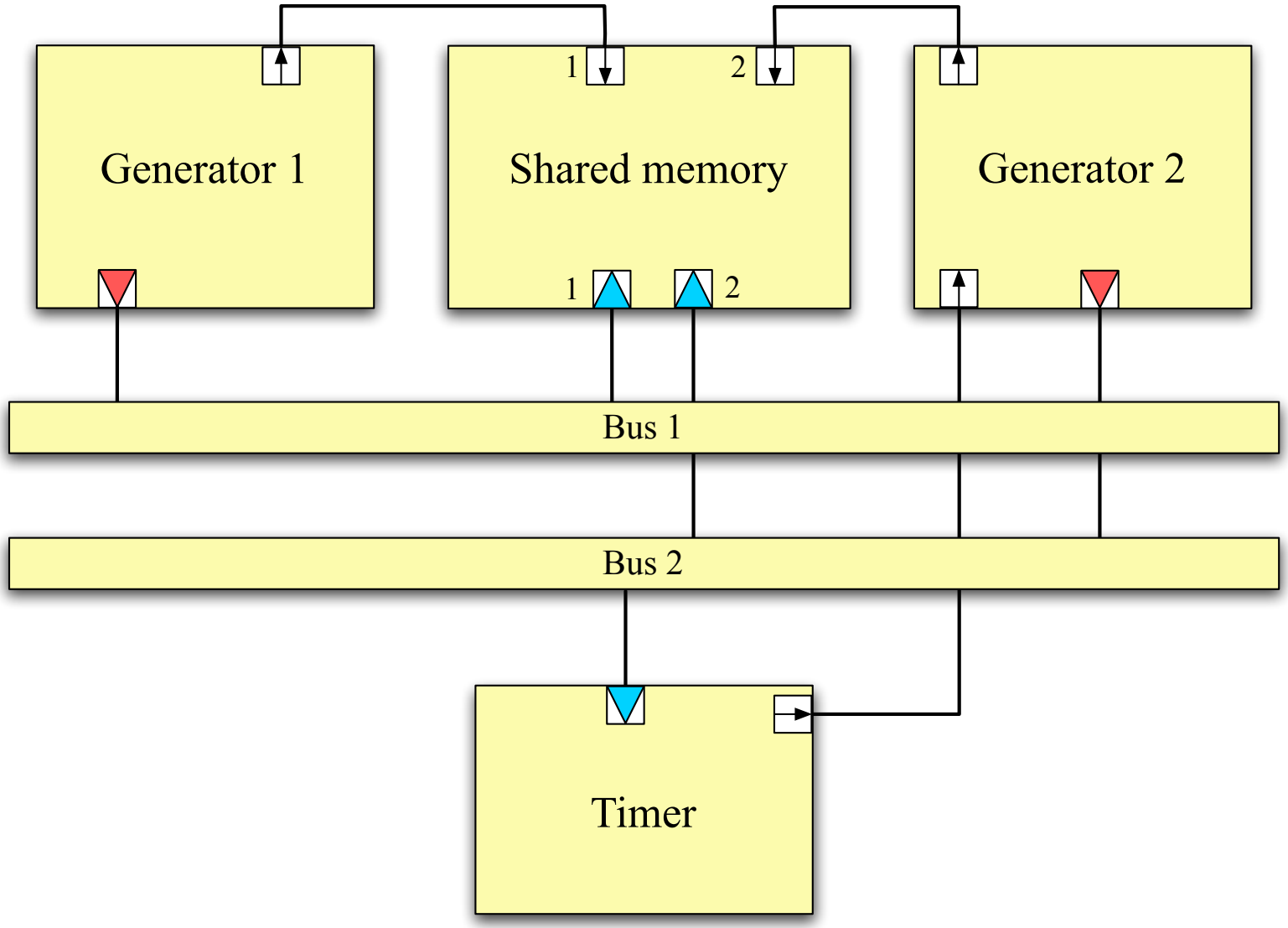
- Synchronous Circuit Description
- Synthesisable
- Usage: manufacture the SoC

Transaction Level Modeling

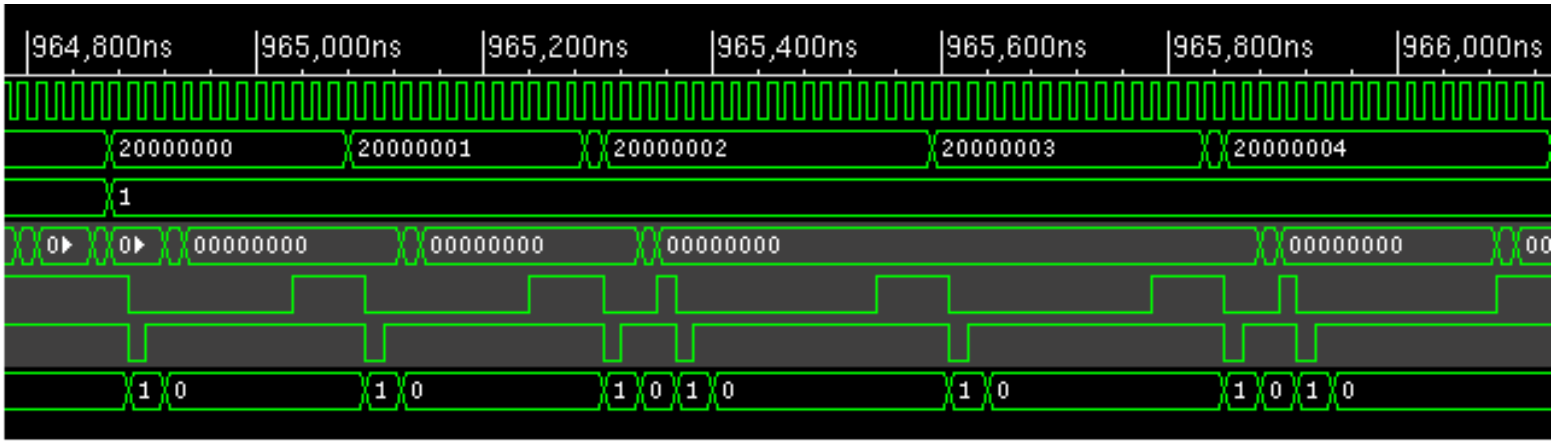
- Clockless
- Explicit system synchronisations
- Usage: Embedded Software development, System Integration, Architecture Evaluation



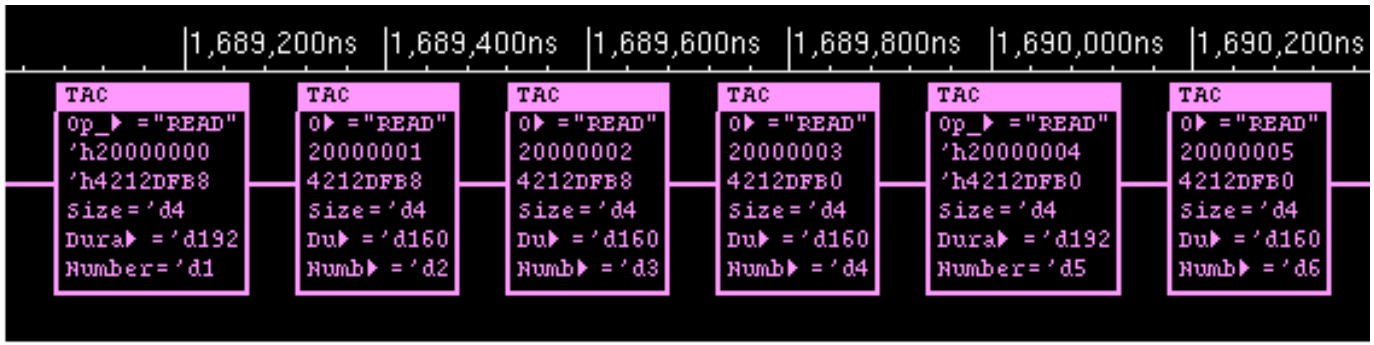
TLM : example



TLM: communications



RTL



TLM

Bus communications abstraction

TLM: implementation

- TL-Models implemented in SystemC
  - C++ Library/“Language”
  - Non-preemptive simulation kernel
  - Standardised (IEEE 1666)
    http://www.systemc.org
- TLM Classes written in SystemC
  - STMicroelectronics’ TAC Protocol
    http://www.greensocs.com/TACPackage
  - Standardisation in progress...
    (OSCI TLM Working Group)

Transactional Levels

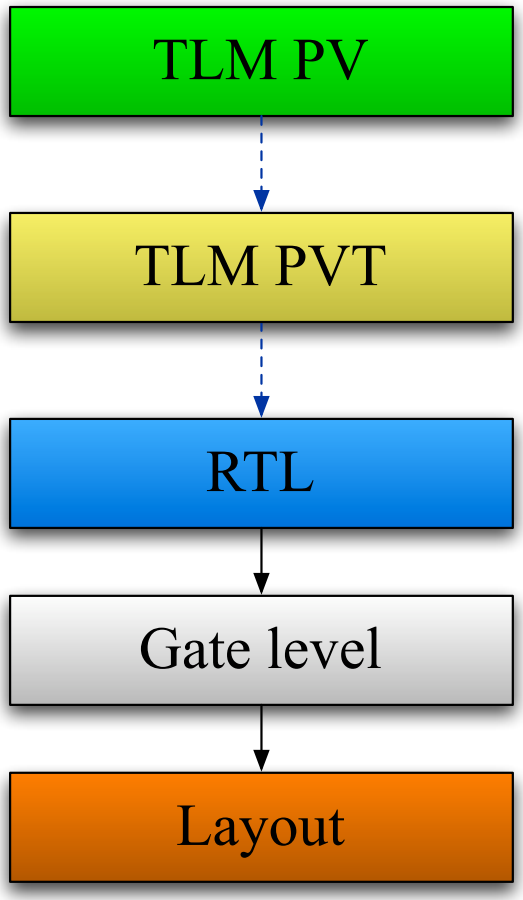
- Conflicting needs at transaction-level
- Timed/Untimed, Granularity...

TLM Programmer's View (PV)

- Time has no meaning
- Coarse communications granularity
- For Embedded Software Development, System Integration

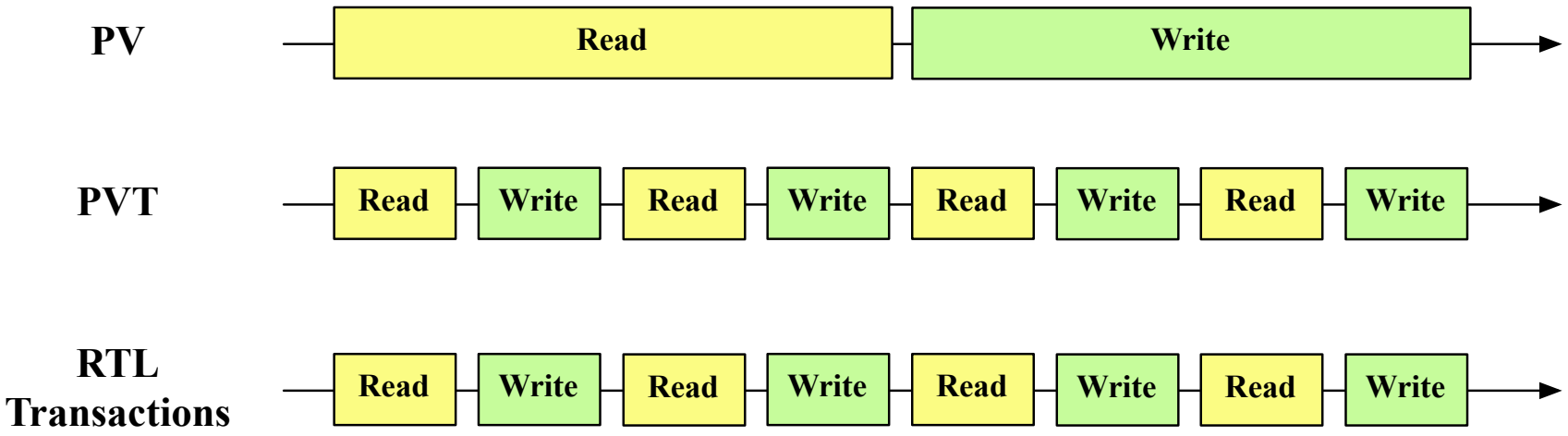
TLM Programmer's View with Time (PVT)

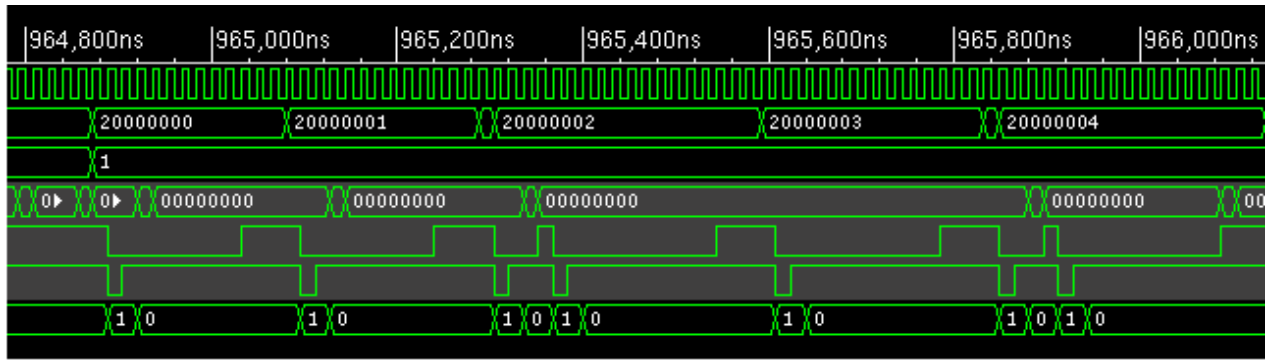
- Precise timings induced by microarchitecture
- Bus communications granularity
- For Architecture Evaluation, Validation of Timed Aspects of Embedded Software



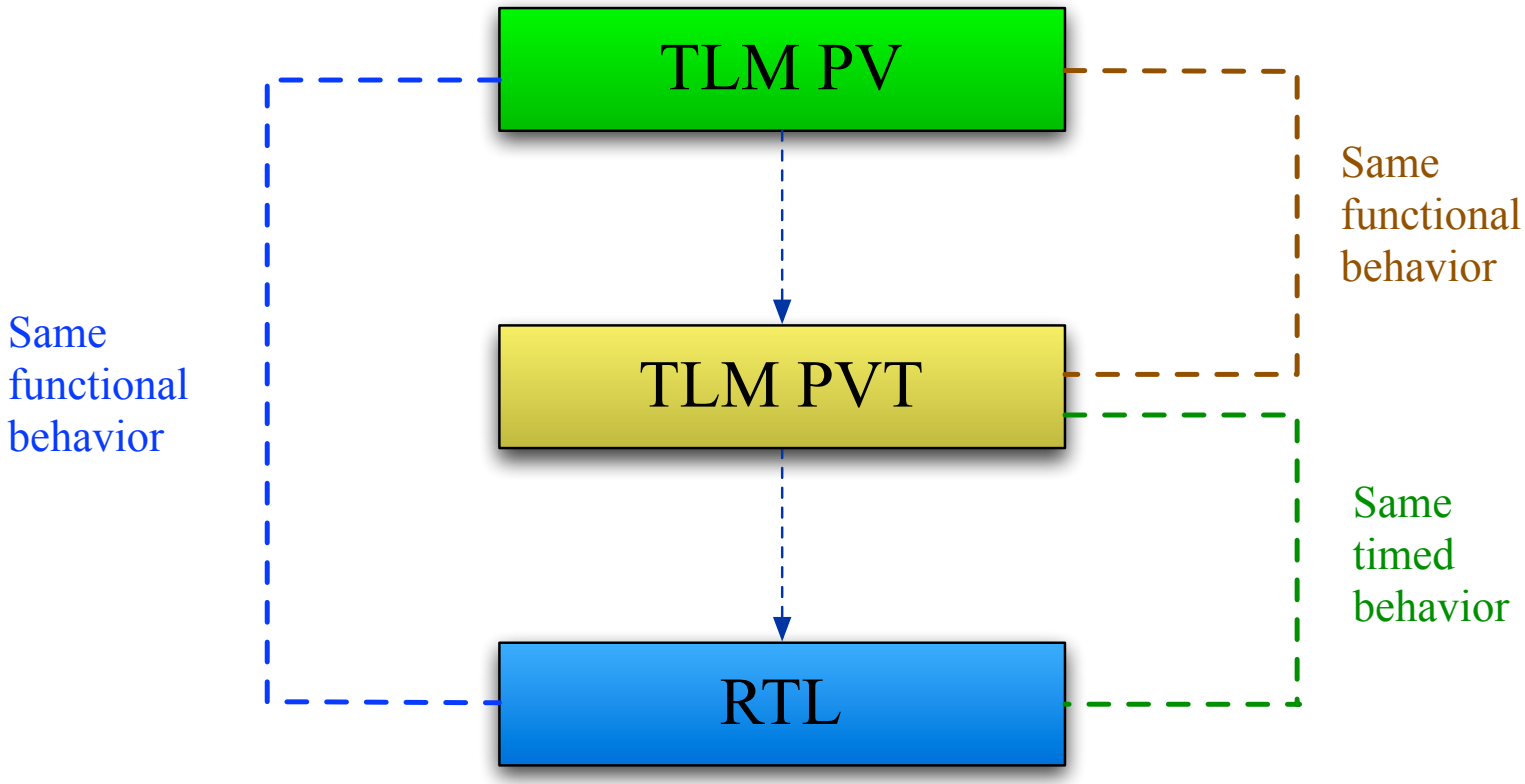
PV/PVT : trace example

- Memory transfer:





PV/PVT : constraints



PV/PVT: approach

For each component:

- Build the PV model
  - Lightweight modeling effort
  - Early availability in the design flow
  - Simulation speed
- Then build the PVT model:
  - With unmodified PV model
  - By adding “T ” informations available later on
- Hope: retain “good” properties of the PV model

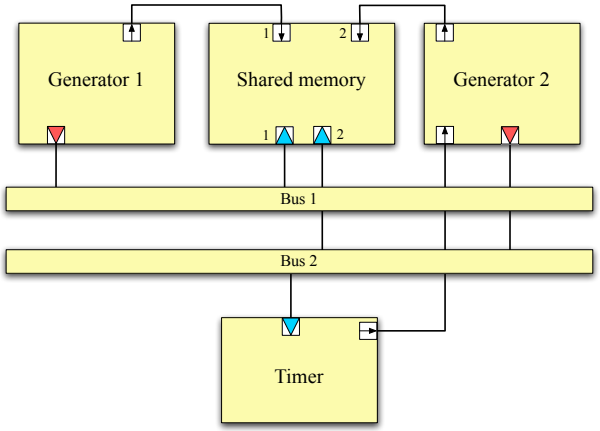
Formalizing the approach (1/3)

- Capture the elements of the equation: PVT $=$ PV $\oplus$ T
  - PV : unmodified PV model of the component
  - T : standalone microarchitecture model
  - $\oplus$: "glue", synchronization between PV and T
- Properties to prove:
  - Functional consistency between PV and PVT
  - Logical consistency betwen PV and T (local and global)
  - Matching between elementary timings and their effective contribution

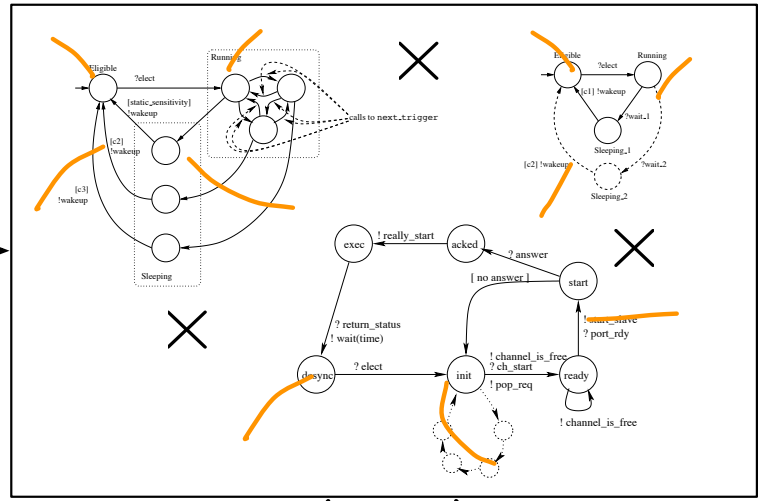
Formalizing the approach (2/3)

- Previous work from Matthieu Moy [MMMC05][Moy05]
  - SystemC semantics using HPIOM (synchronous automata)
  - Connection to model-checkers (Lesar, SMV...): Lussy toolchain
- Our goals here:
  - Define a more direct semantics for SystemC/TLM
  - Not for formal verification...
  - .. but for experiments with manually written models
  - Generic Proof for PV/PVT

Formalizing the approach (3/3)



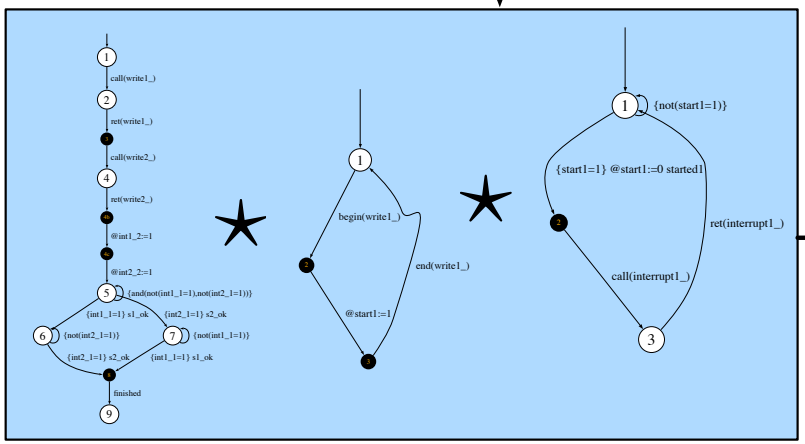
Lussy



SMV

Lesar

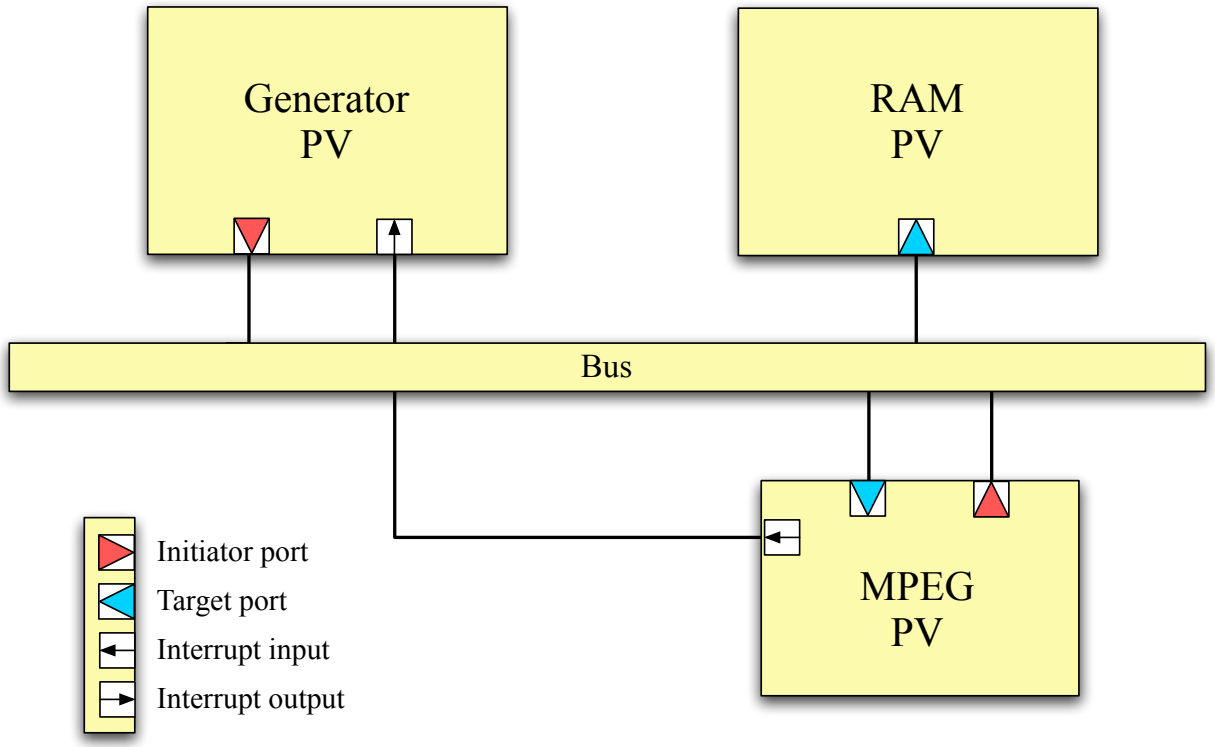
Manual Abstraction and Writing



Results

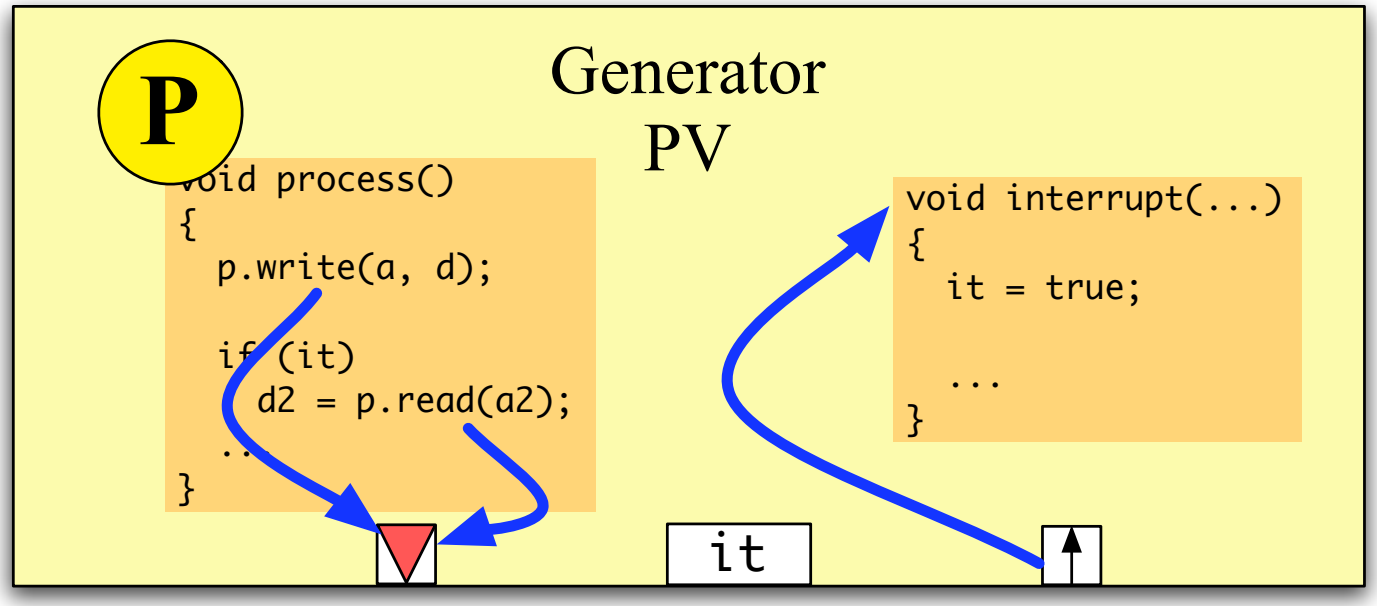
SystemC TLM Modeling (1/3)

- Different modules connected by “wires”
- Communications between modules: functions calls to communications functions



SystemC TLM Modeling (2/3)

- Two kind of code inside modules:
  - Processes (`SC_THREAD`, `SC_METHOD`)
  - Communications functions
- Synchronizations inside a module:
  - Shared variables
  - SystemC's events (`sc_event`)



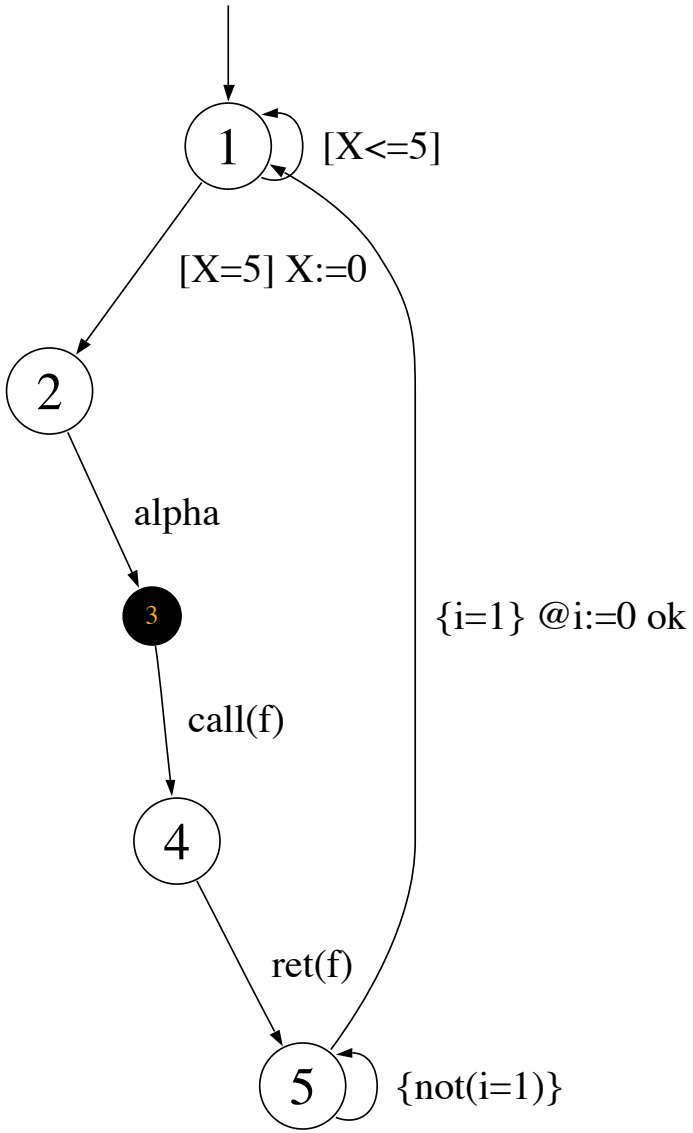
SystemC TLM Modeling (3/3)

- Non-preemptive scheduling of processes:
  - Processes choose when to yield
  - Atomicity
  - Non-determinism
- Yield:
  - Wait for some time: `wait(2, SC_NS);`
  - Wait for a sc_event: `wait(e);`
- SystemC's events are instantaneous

Outline

Micmac Automata (1/2)

- A micmac automaton is a tuple $(Q, q_i, V, \varphi_i, C, L, I, A, T, M)$ where:
  - Q is a set of control points,
  - q_i is the initial control point,
  - V is a set of variables,
  - φ_i is the initial valuation of the variables,
  - C is a set of clocks,
  - L is a set of internal labels,
  - I is a set of function identifiers,
  - A is a set of variable assignments,



Micmac Automata (2/2)

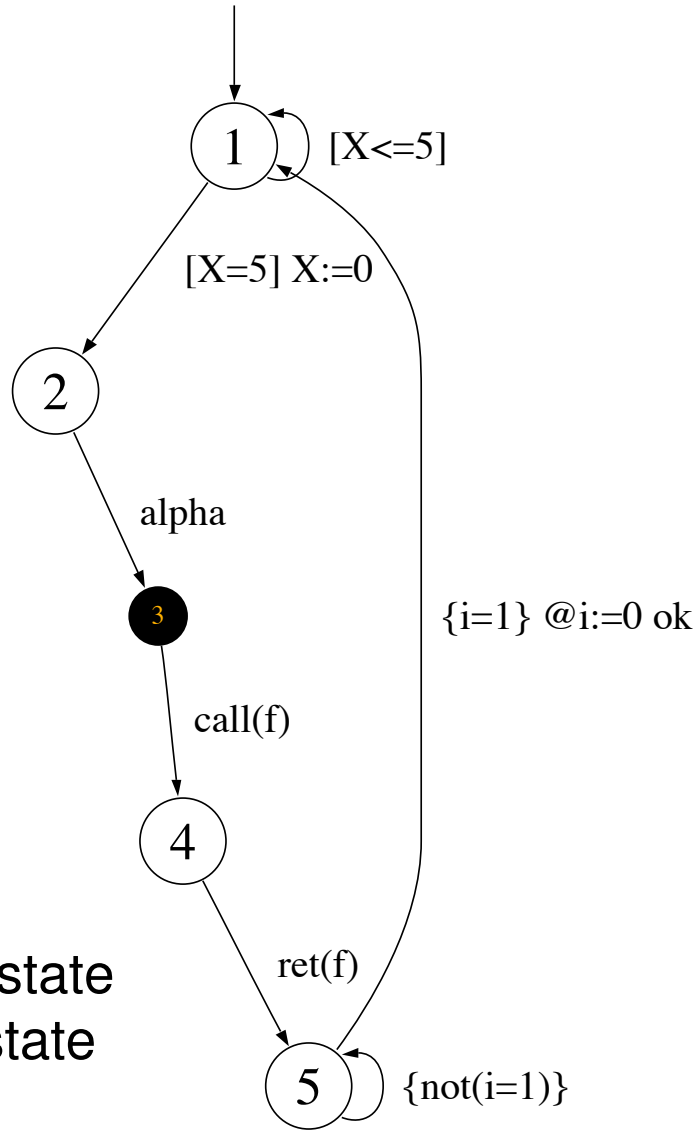
- A micmac automaton is a tuple $(Q, q_i, V, \varphi_i, C, L, I, A, T, M)$ where:
  - T is the transition relation:

$$\begin{aligned} T \subseteq\ & Q \times \\ & \{call, ret, begin, end\} \times I \\ & \cup \mathcal{P}(L \cup \{\varepsilon\}) \\ & \times CG \times G \times \mathcal{P}(A) \times C \\ & \times Q \end{aligned}$$

  - M indicates each state's type:

$$M : \quad Q \rightarrow \{true, false\}$$

$$q \mapsto \begin{cases} true & \text{if } q \text{ is a Macro-state} \\ false & \text{if } q \text{ is a micro-state} \end{cases}$$



Micmac Product (without clocks)

Binary product $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$

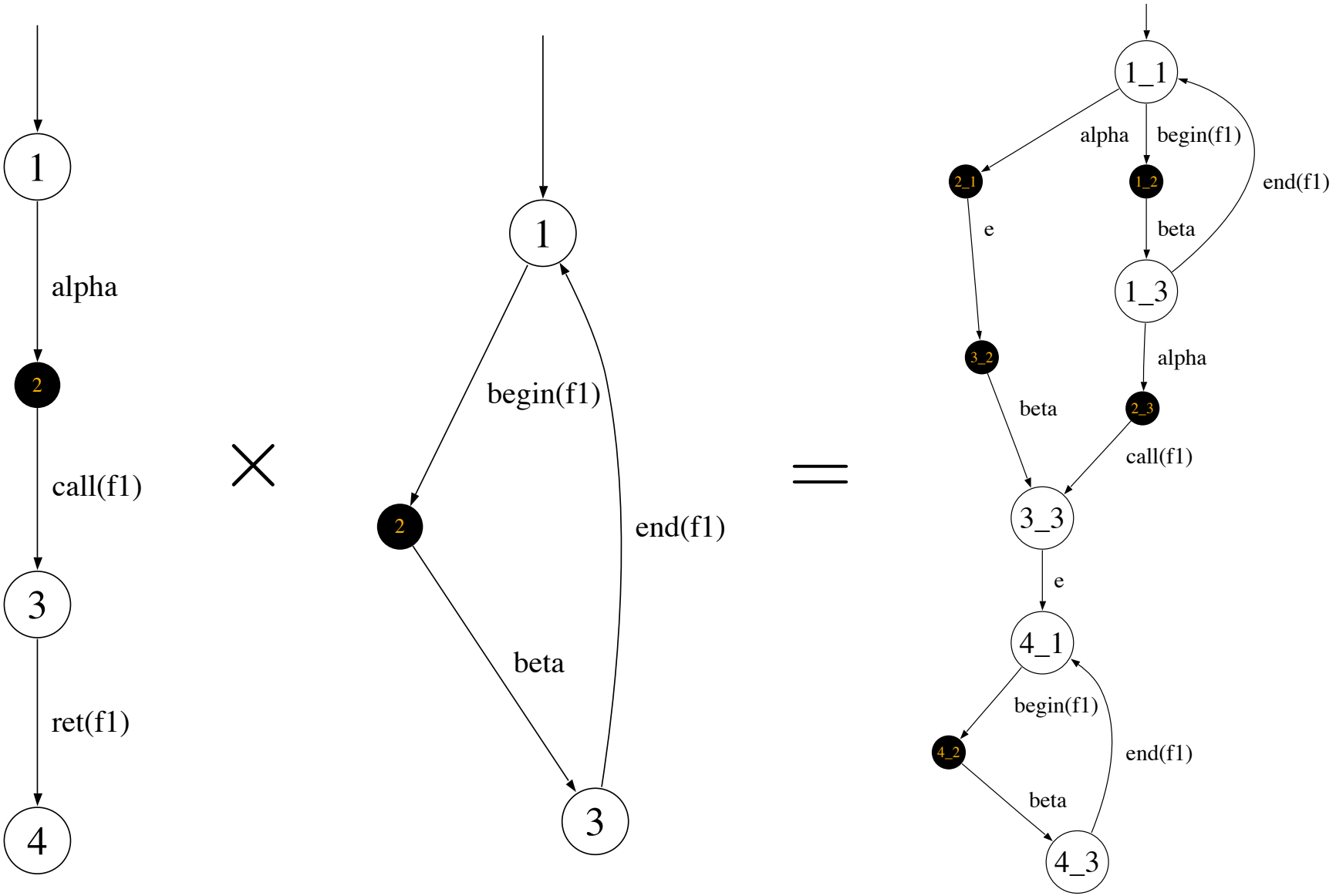
- For each state $q \subseteq Q^1 \times Q^2$:
  - $M(q) = M^1(q^1) \wedge M^2(q^2)$
  - If $M^1(q^1)$, each transition t^1 of q^1 belongs to q
  - If $M^2(q^2)$, each transition t^2 of q^2 belongs to q
  - Function call semantics:

$$q^1 \xrightarrow{call(f)} q'^1 \in \mathcal{A}^1,\ q^2 \xrightarrow{begin(f)} q'^2 \in \mathcal{A}^2 \Rightarrow (q^1, q^2) \xrightarrow{\varepsilon} (q'^1, q'^2) \in \mathcal{A}$$

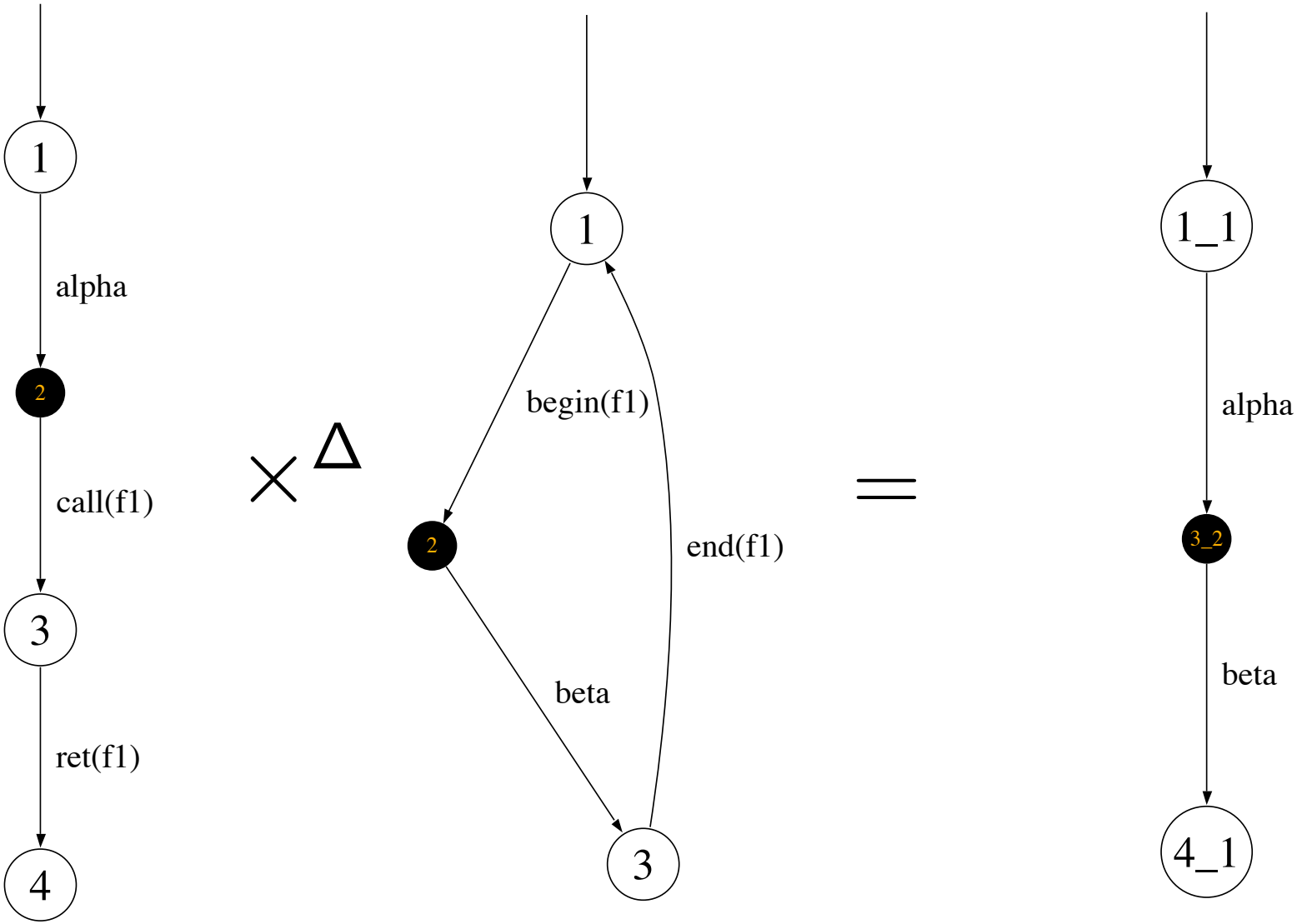
$$q^1 \xrightarrow{ret(f)} q'^1 \in \mathcal{A}^1,\ q^2 \xrightarrow{end(f)} q'^2 \in \mathcal{A}^2 \Rightarrow (q^1, q^2) \xrightarrow{\varepsilon} (q'^1, q'^2) \in \mathcal{A}$$

- Cutting: for f function identifier, remove every transition holding $call(f)$ or $ret(f)$ or $begin(f)$ or $end(f)$

Example: product



Example: product with cutting



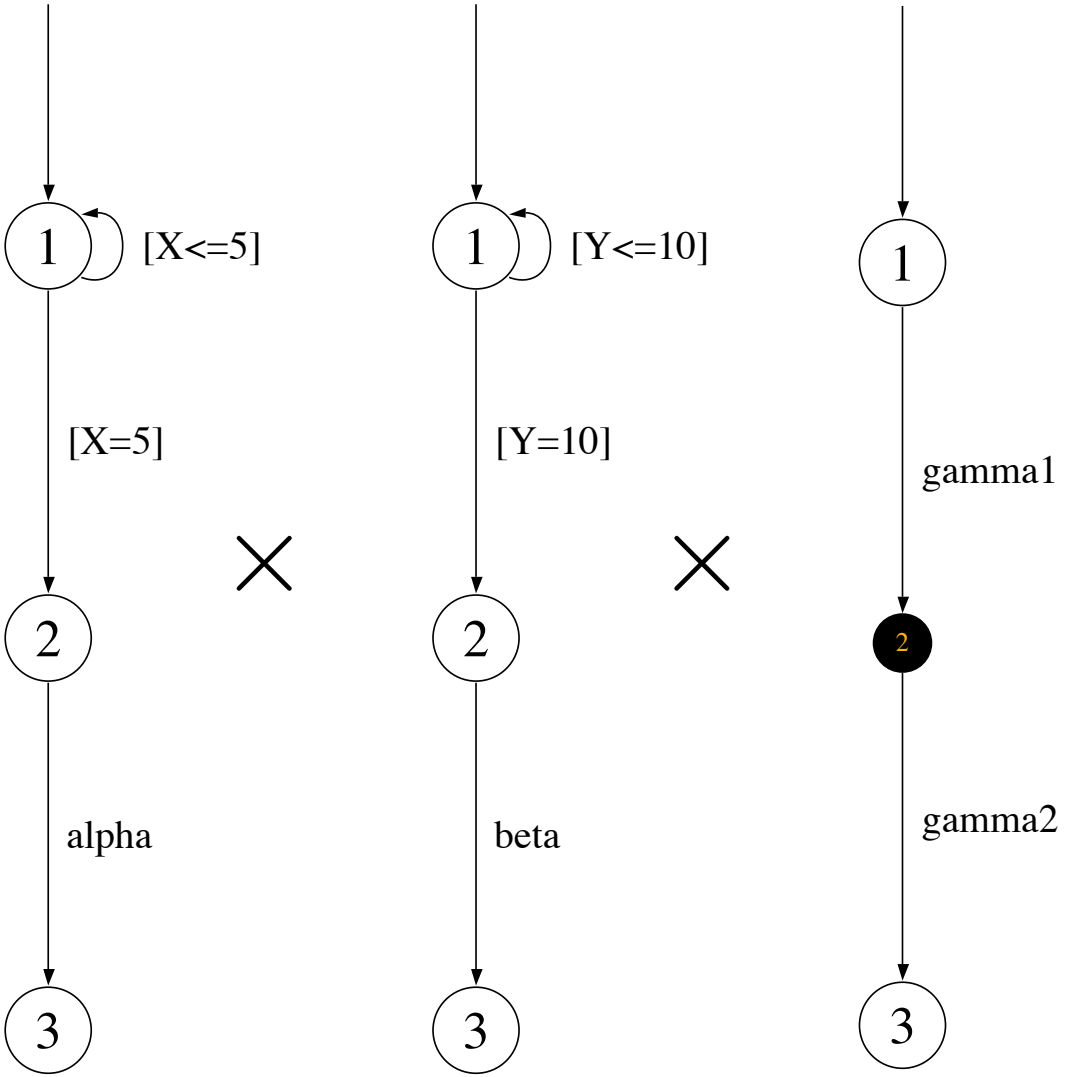
Micmac Product (with clocks)

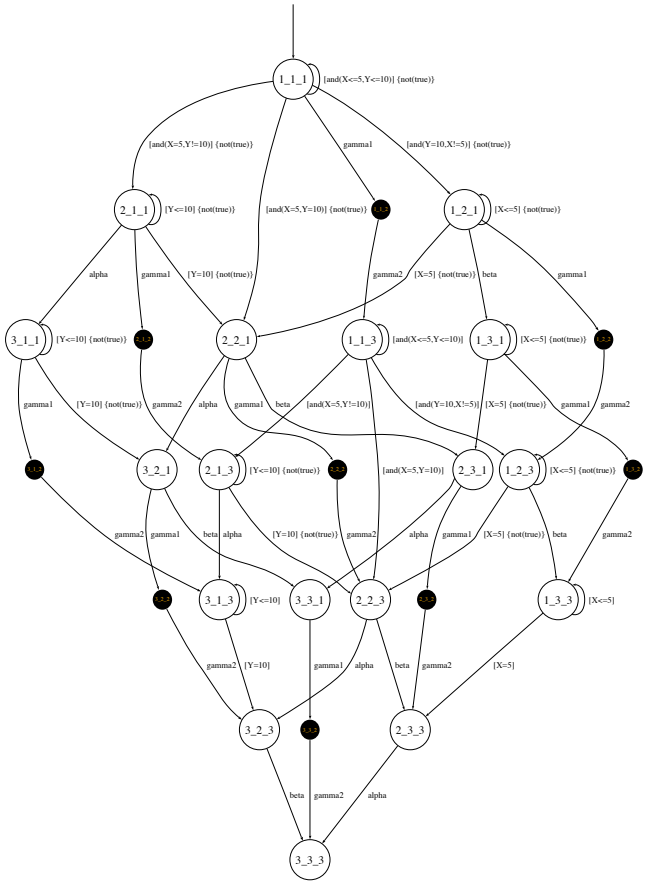
- Two “kinds” of transition:
  - Transition holding [*true*]: untimed transitions
  - Transition with non-trivial clock guards: timed transitions
- Untimed transitions: apply previous product
- For each timed transitions
  $q^1 \xrightarrow{[cg^1]\ \{g^1\}} q'^1 \in \mathcal{A}^1$, $q^2 \xrightarrow{[cg^2]\ \{g^2\}} q'^2 \in \mathcal{A}^2$:
  - $(q^1, q^2) \xrightarrow{[cg^1 \wedge \neg \Pi^2]\ \{g^1 \wedge \Omega^1 \wedge \Omega^2\}} (q'^1, q^2) \in \mathcal{A}$
  - $(q^1, q^2) \xrightarrow{[cg^1 \wedge cg^2]\ \{g^1 \wedge g^2 \wedge \Omega^1 \wedge \Omega^2\}} (q'^1, q'^2) \in \mathcal{A}$
  - $(q^1, q^2) \xrightarrow{[cg^2 \wedge \neg \Pi^1]\ \{g^2 \wedge \Omega^1 \wedge \Omega^2\}} (q^1, q'^2) \in \mathcal{A}$

Micmac Product (with clocks)

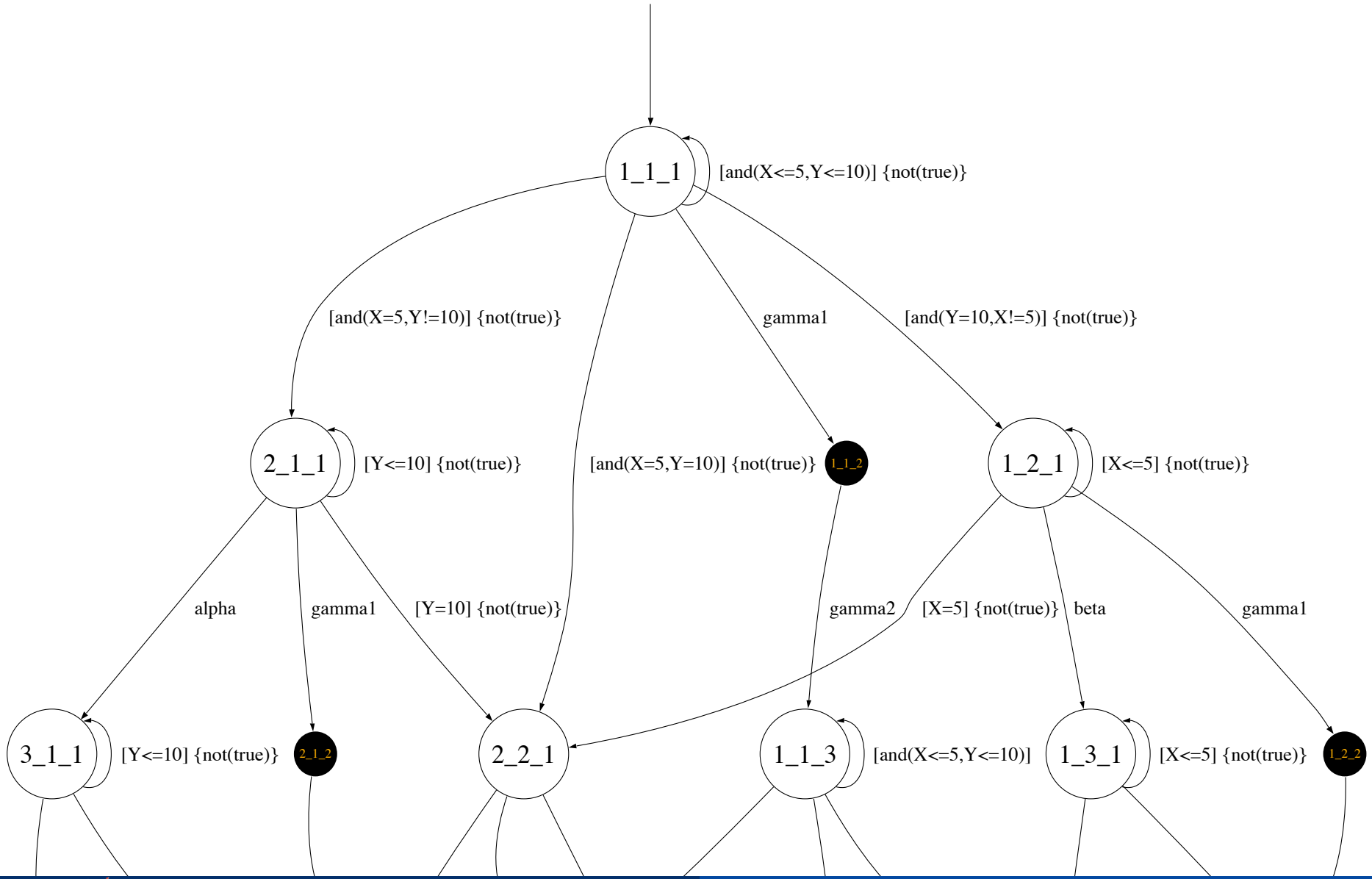
- For each timed transitions $q^1 \xrightarrow{[cg^1]\ \{g^1\}} q'^1 \in \mathcal{A}^1$, $q^2 \xrightarrow{[cg^2]\ \{g^2\}} q'^2 \in \mathcal{A}^2$:
  - $(q^1, q^2) \xrightarrow{[cg^1 \wedge \neg \Pi^2]\ \{g^1 \wedge \Omega^1 \wedge \Omega^2\}} (q'^1, q^2) \in \mathcal{A}$
  - $(q^1, q^2) \xrightarrow{[cg^1 \wedge cg^2]\ \{g^1 \wedge g^2 \wedge \Omega^1 \wedge \Omega^2\}} (q'^1, q'^2) \in \mathcal{A}$
  - $(q^1, q^2) \xrightarrow{[cg^2 \wedge \neg \Pi^1]\ \{g^2 \wedge \Omega^1 \wedge \Omega^2\}} (q^1, q'^2) \in \mathcal{A}$
- $\Pi^1 = \bigwedge_{cg \in t^1 \text{ from } q^1} cg \qquad \Pi^2 = \bigwedge_{cg \in t^2 \text{ from } q^2} cg$
- $\Omega^1 = \bigwedge_{g \in t^1 \text{ from } q^1} \neg g \qquad \Omega^2 = \bigwedge_{g \in t^2 \text{ from } q^2} \neg g$

Example: product with clockguards

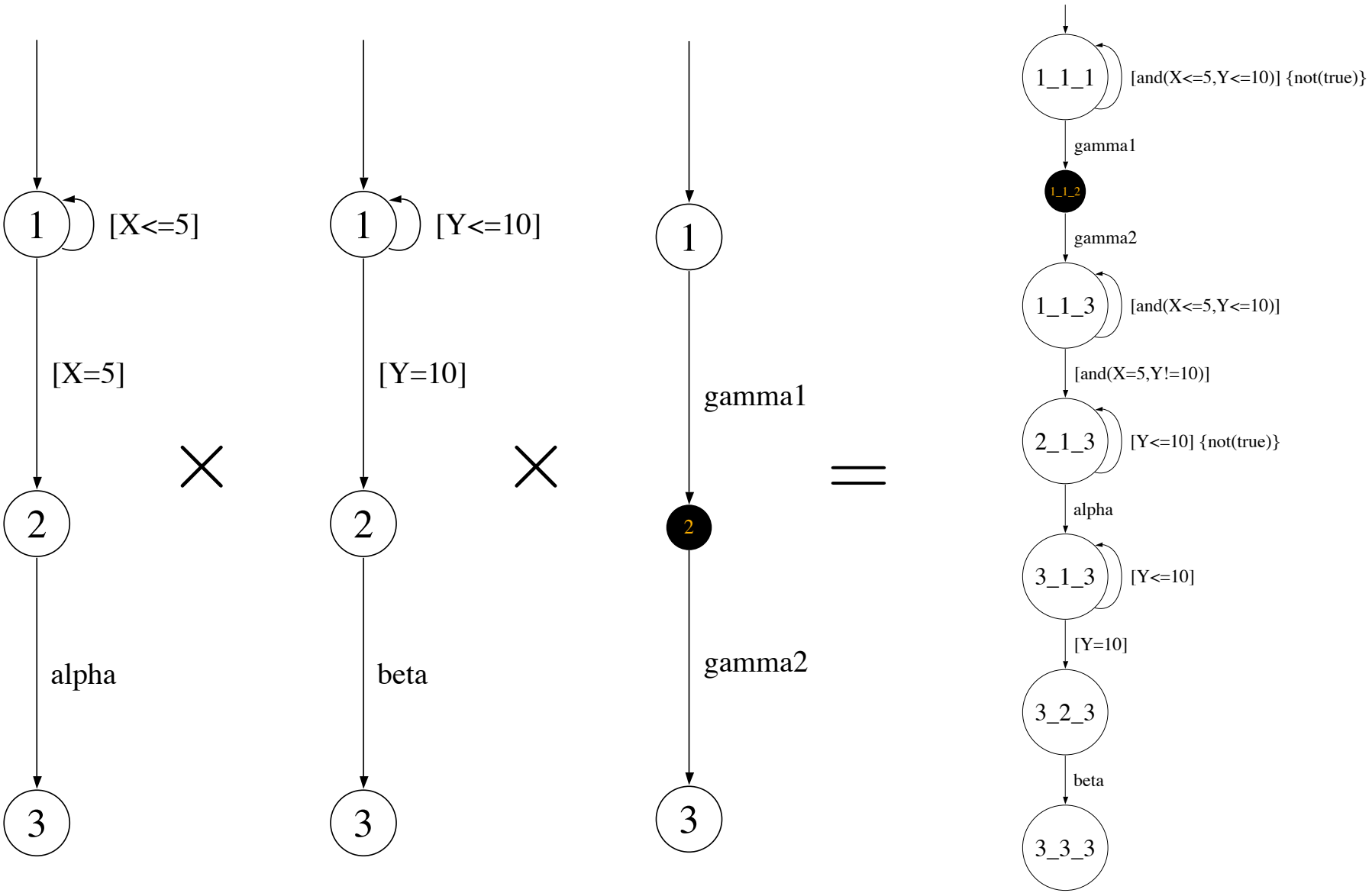




Example: product with clockguards



Example: product with clockguards



Logical Constraints on Micmac automata

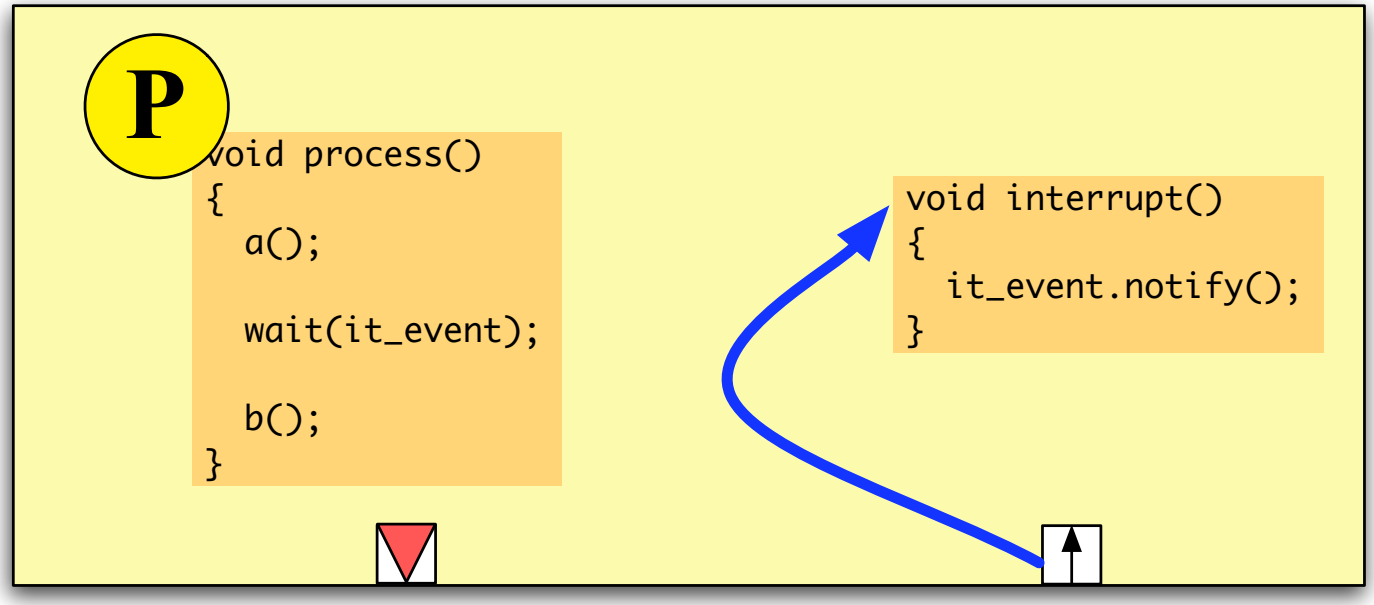
The following states are Macro states:

- The initial state
- A “final” state
- A state with at least one timed transition
- A state pointed to by a timed transition

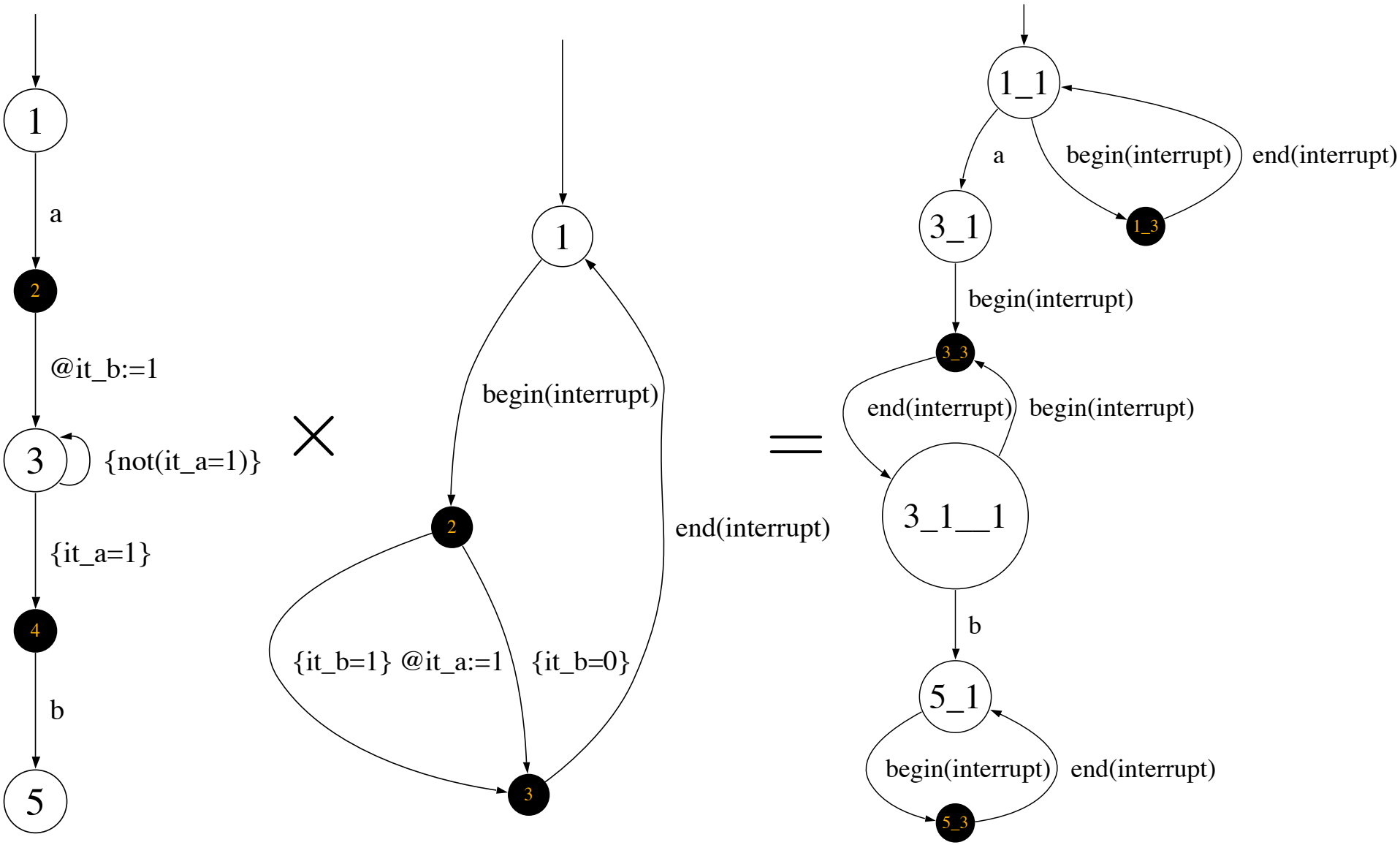
SystemC TLM Modeling with Micmac automata

- One micmac automaton per:
  - Process
  - Communication function
- Function calls model inter-modules communications
- Macro states represent “yielding points”
- Shared variables between automata of the same module
  - Model classical shared variables
  - Model SystemC’s events

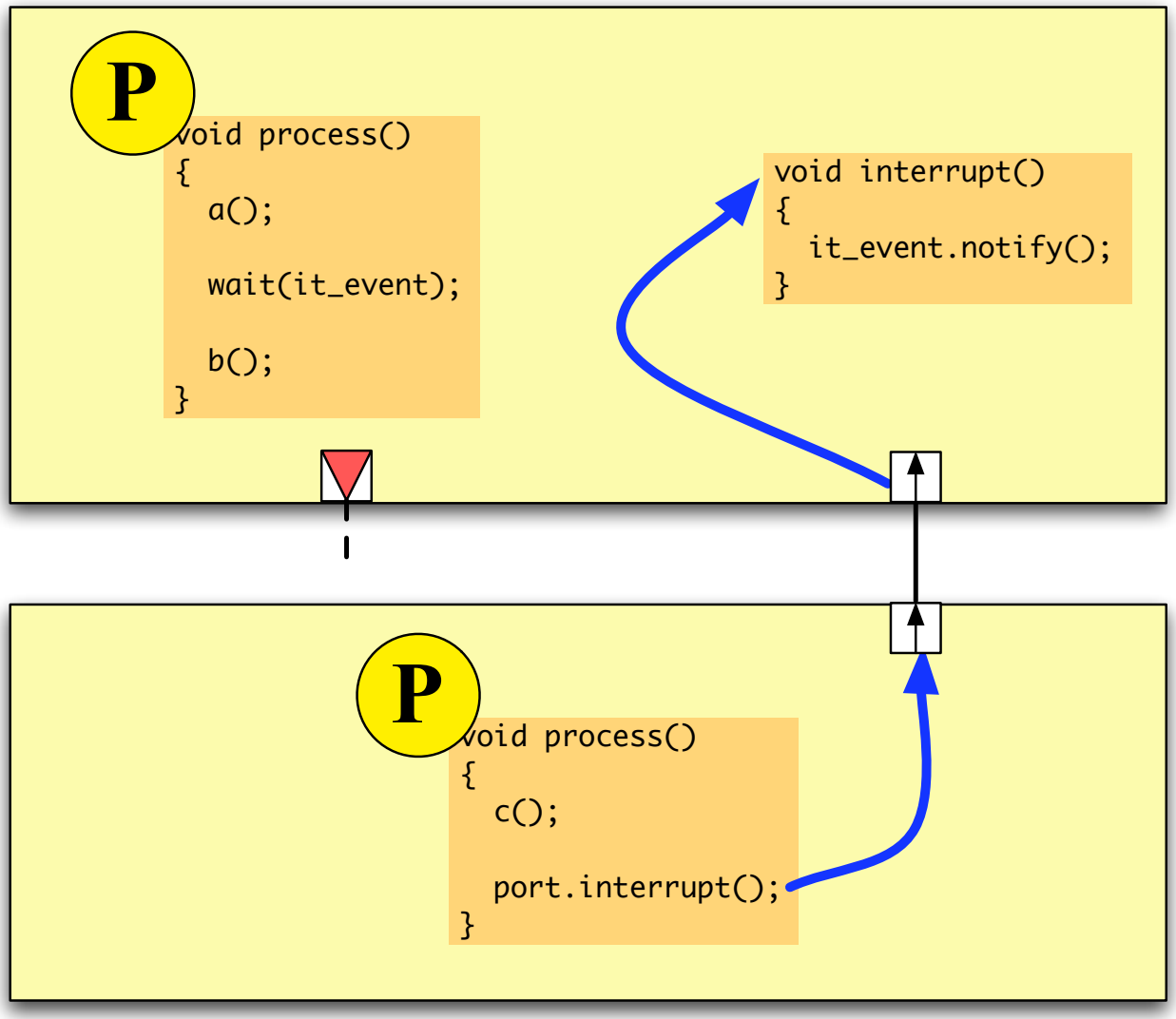
Example


P
void process()
{
a();
wait(it_event);
b();
}
void interrupt()
{
it_event.notify();
}

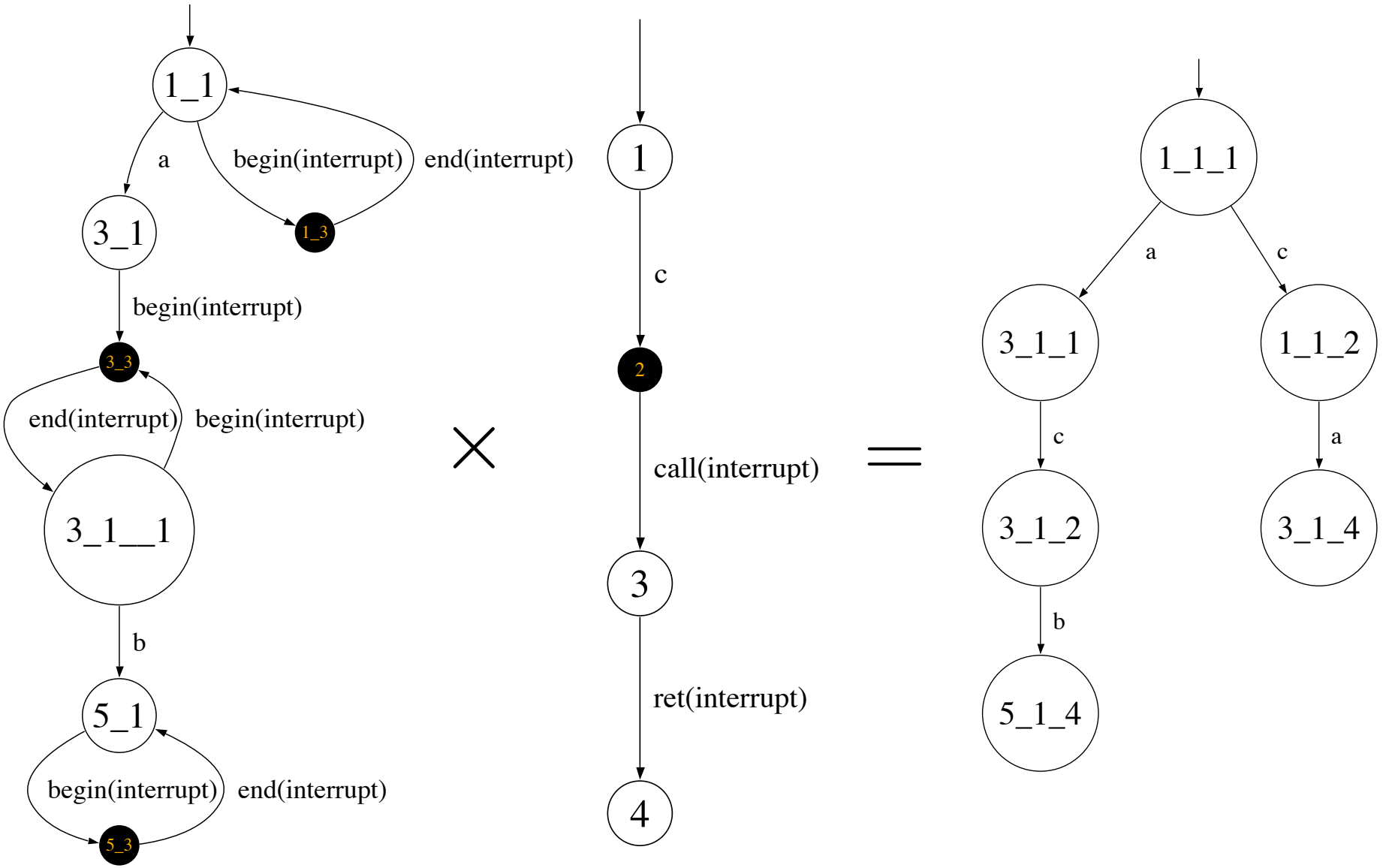
Example: product



Example

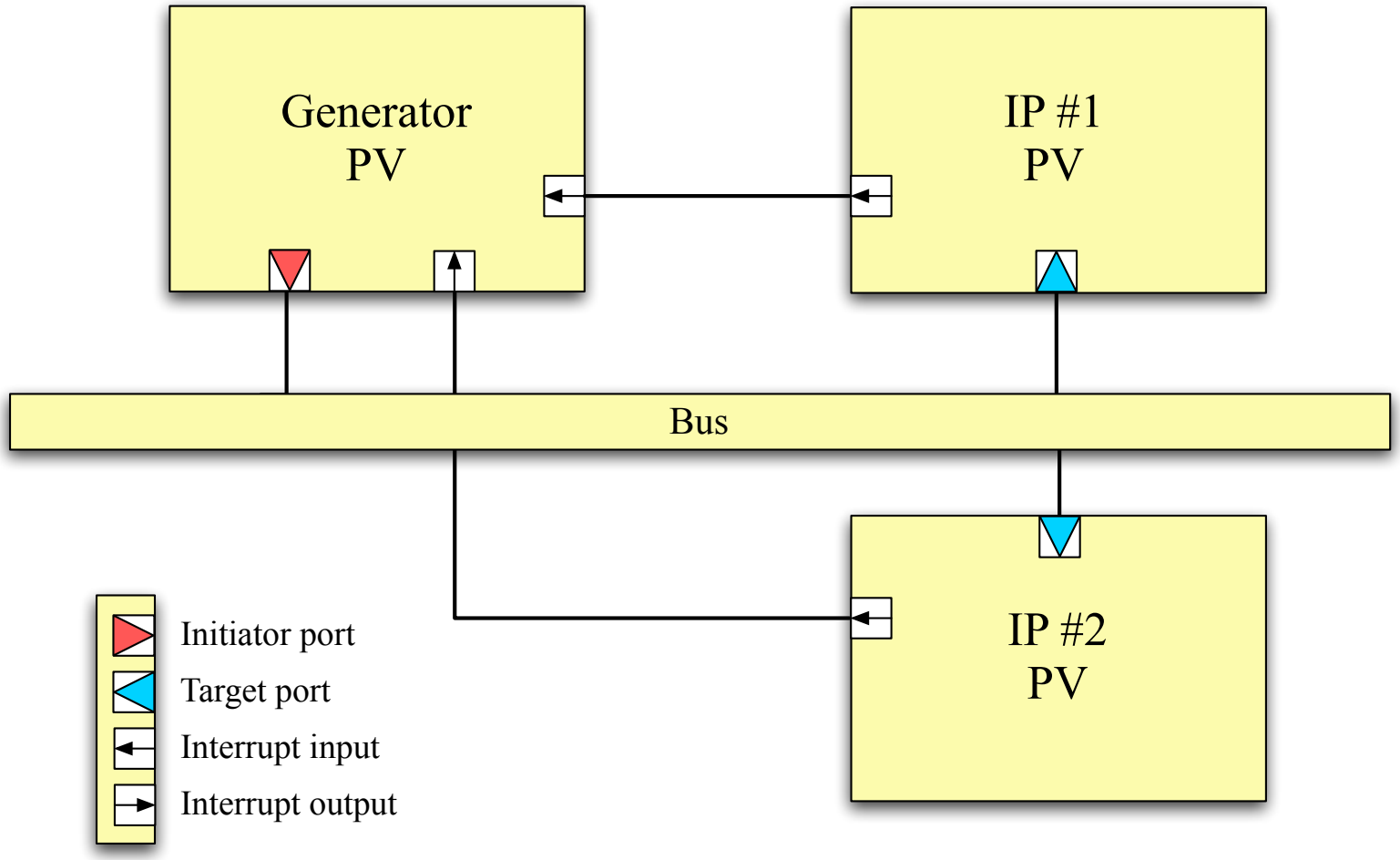

P
void process()
{
a();
wait(it_event);
b();
}
void interrupt()
{
it_event.notify();
}
P
void process()
{
c();
port.interrupt();
}

Example: product

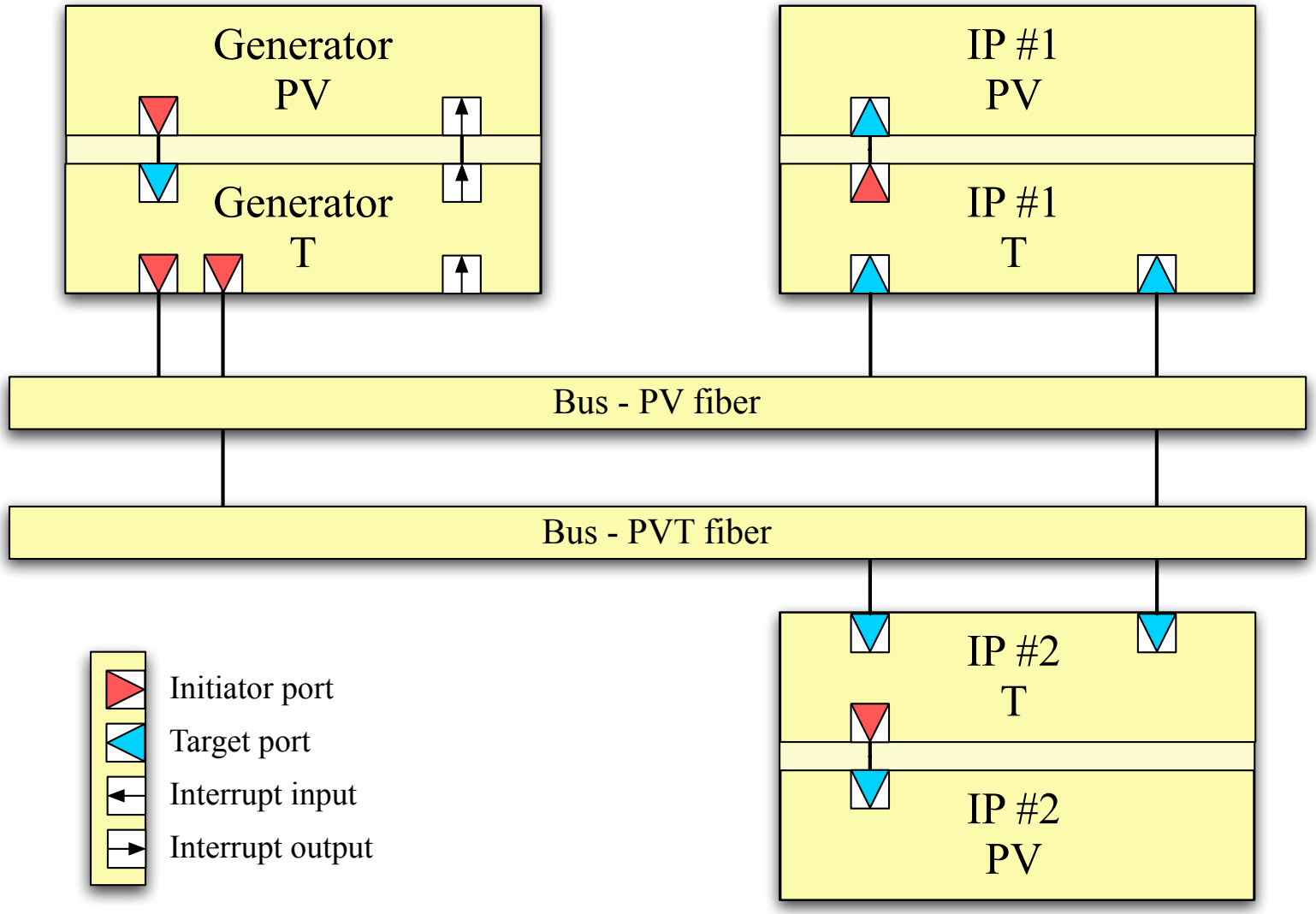


Outline

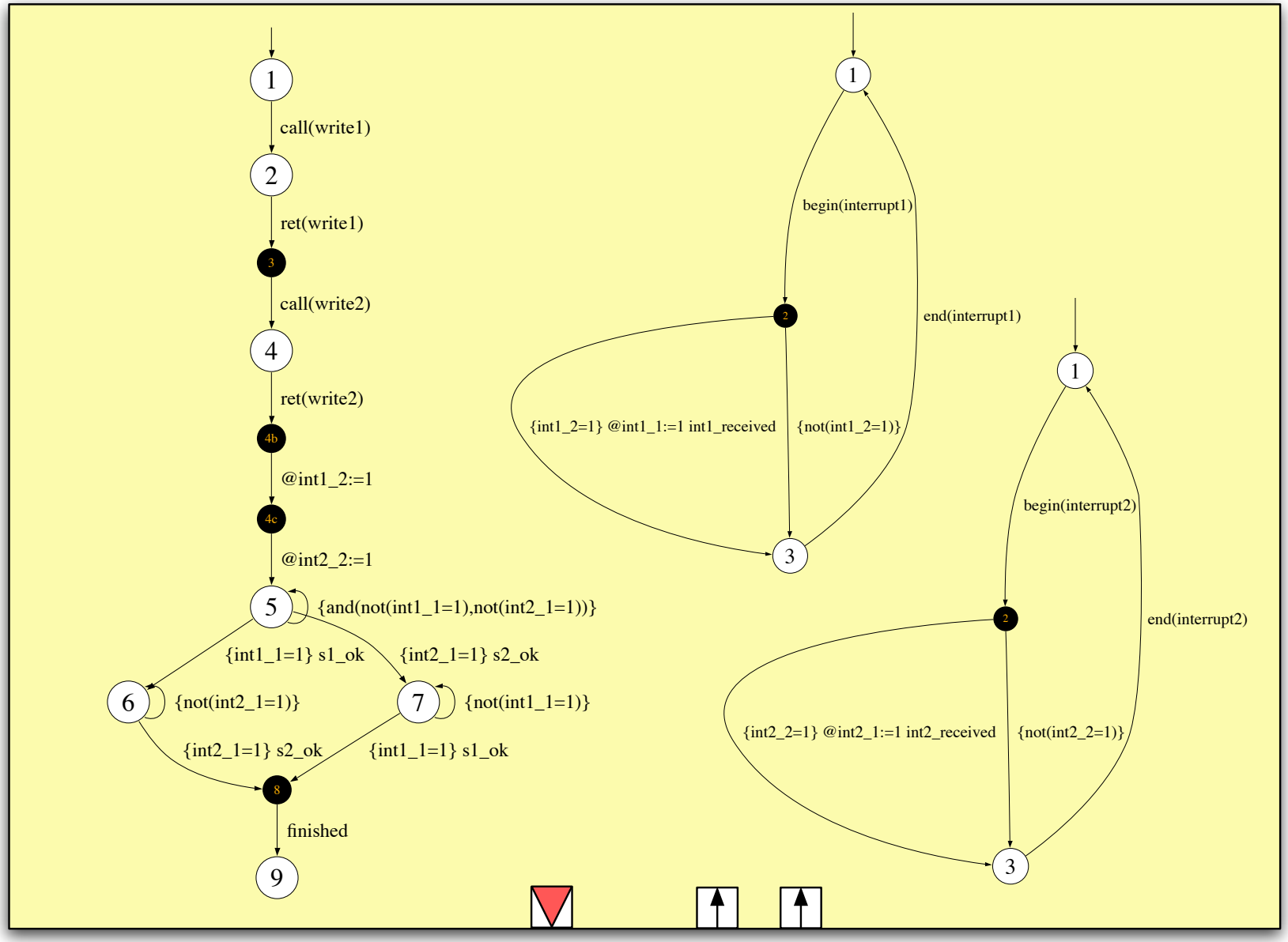
Example PV


Generator
PV
IP #1
PV
Bus
IP #2
PV
Initiator port
Target port
Interrupt input
Interrupt output

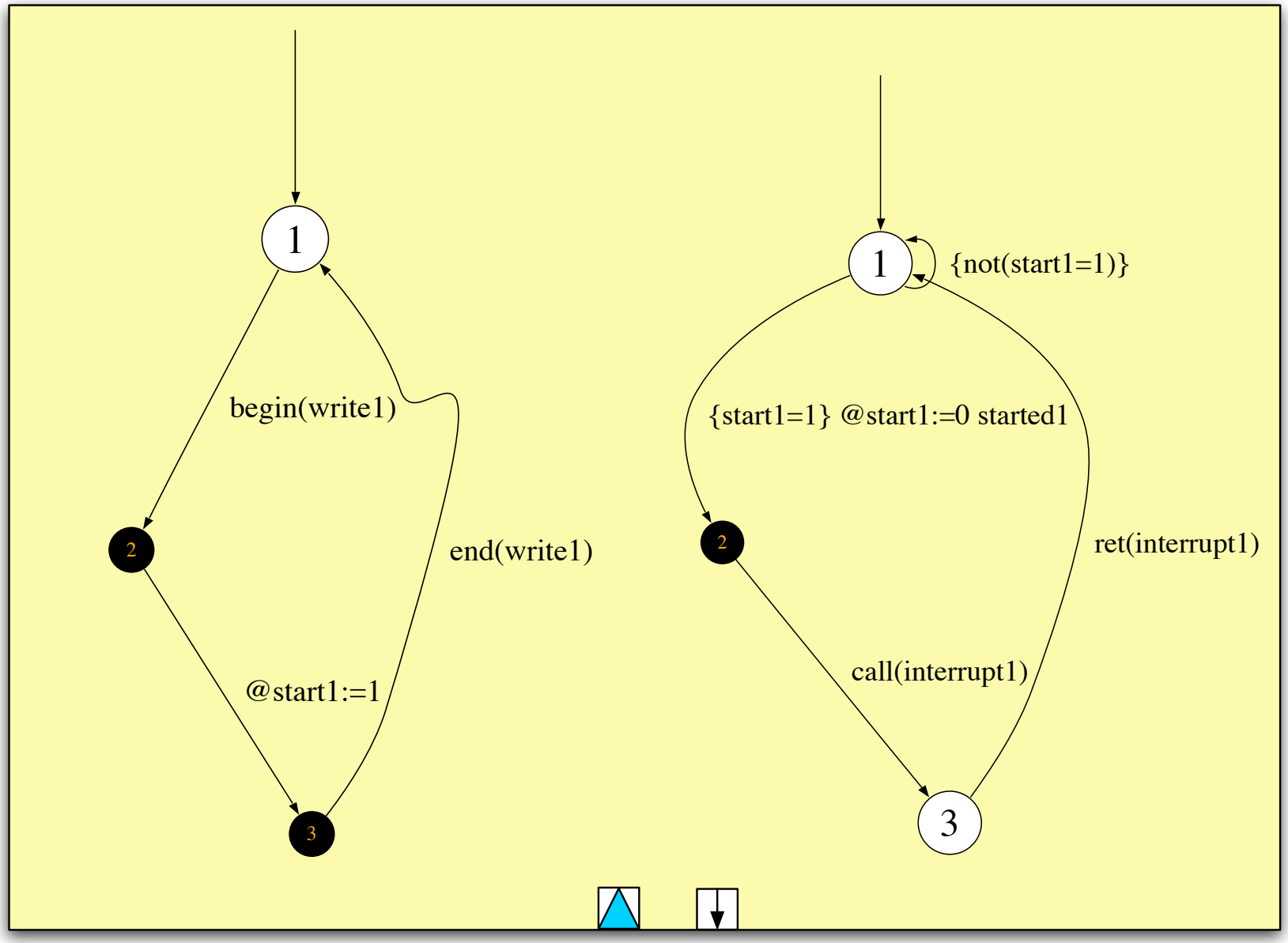
Example PVT


Generator
PV
Generator
T
IP #1
PV
IP #1
T
Bus - PV fiber
Bus - PVT fiber
IP #2
T
IP #2
PV
Initiator port
Target port
Interrupt input
Interrupt output

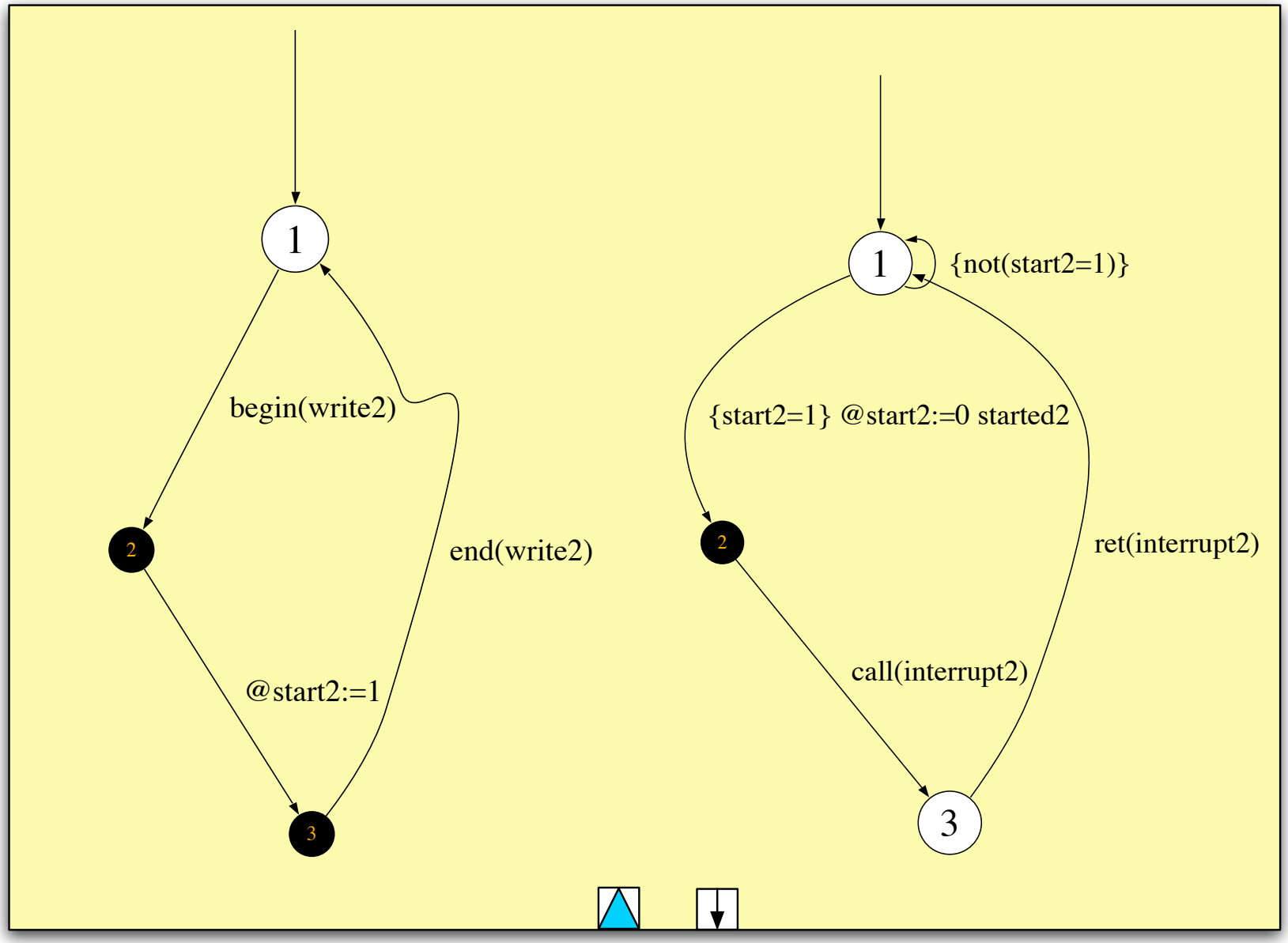
Generator PV



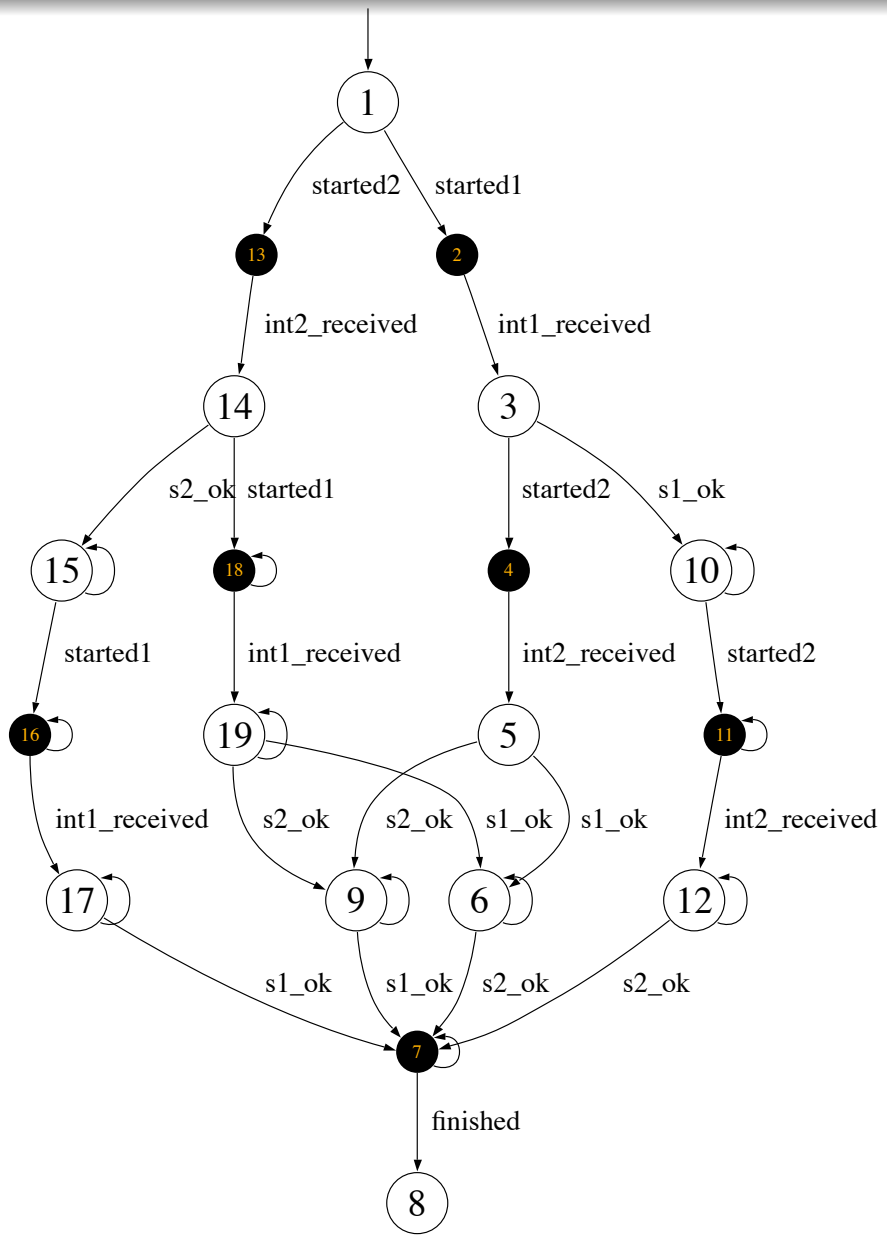
IP #1 PV


1
begin(write1)
2
end(write1)
@start1:=1
3
1
{not(start1=1)}
{start1=1} @start1:=0 started1
2
ret(interrupt1)
call(interrupt1)
3

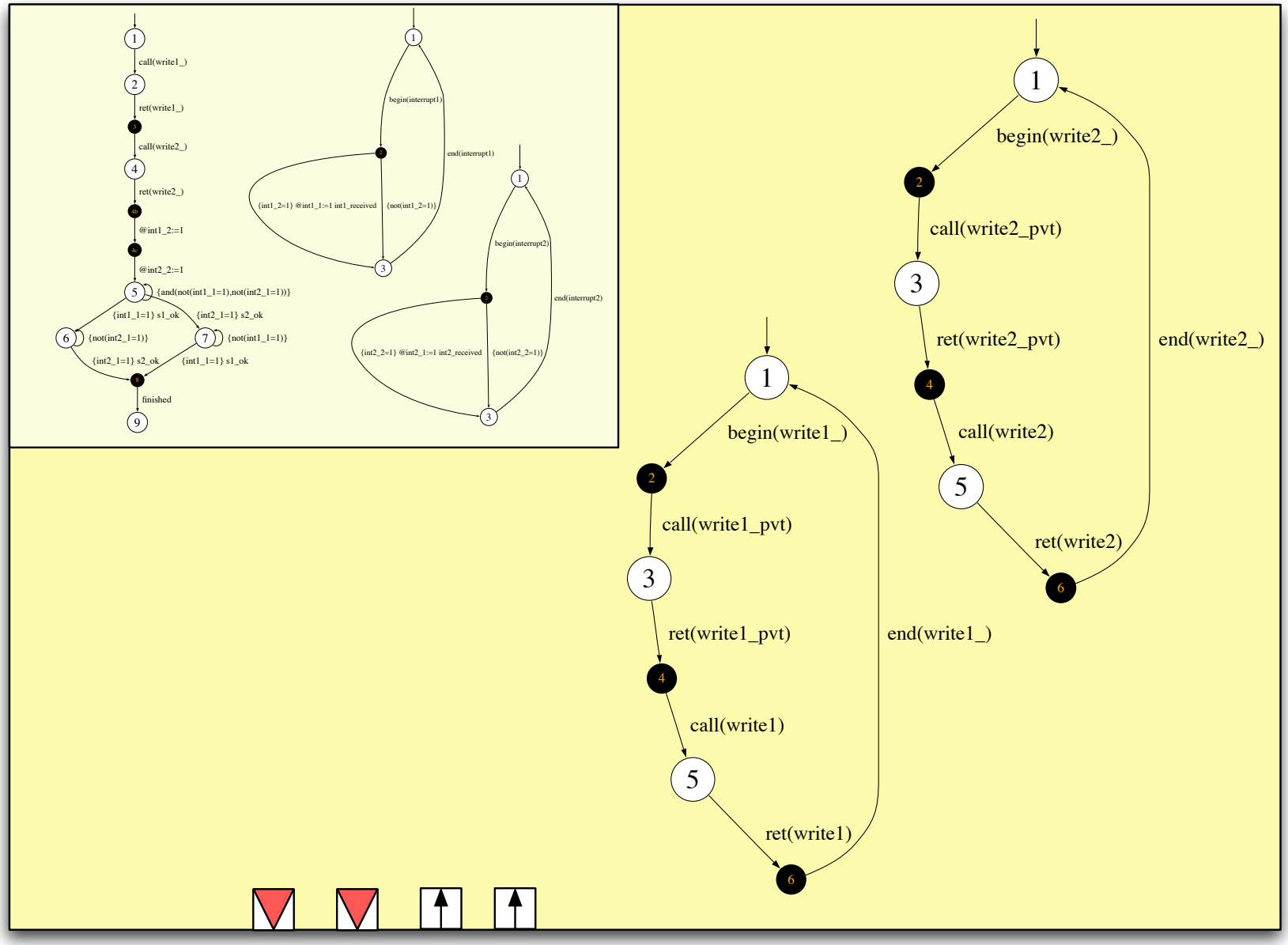
IP #2 PV


1
begin(write2)
2
end(write2)
@start2:=1
3
1
{not(start2=1)}
{start2=1} @start2:=0 started2
2
ret(interrupt2)
call(interrupt2)
3

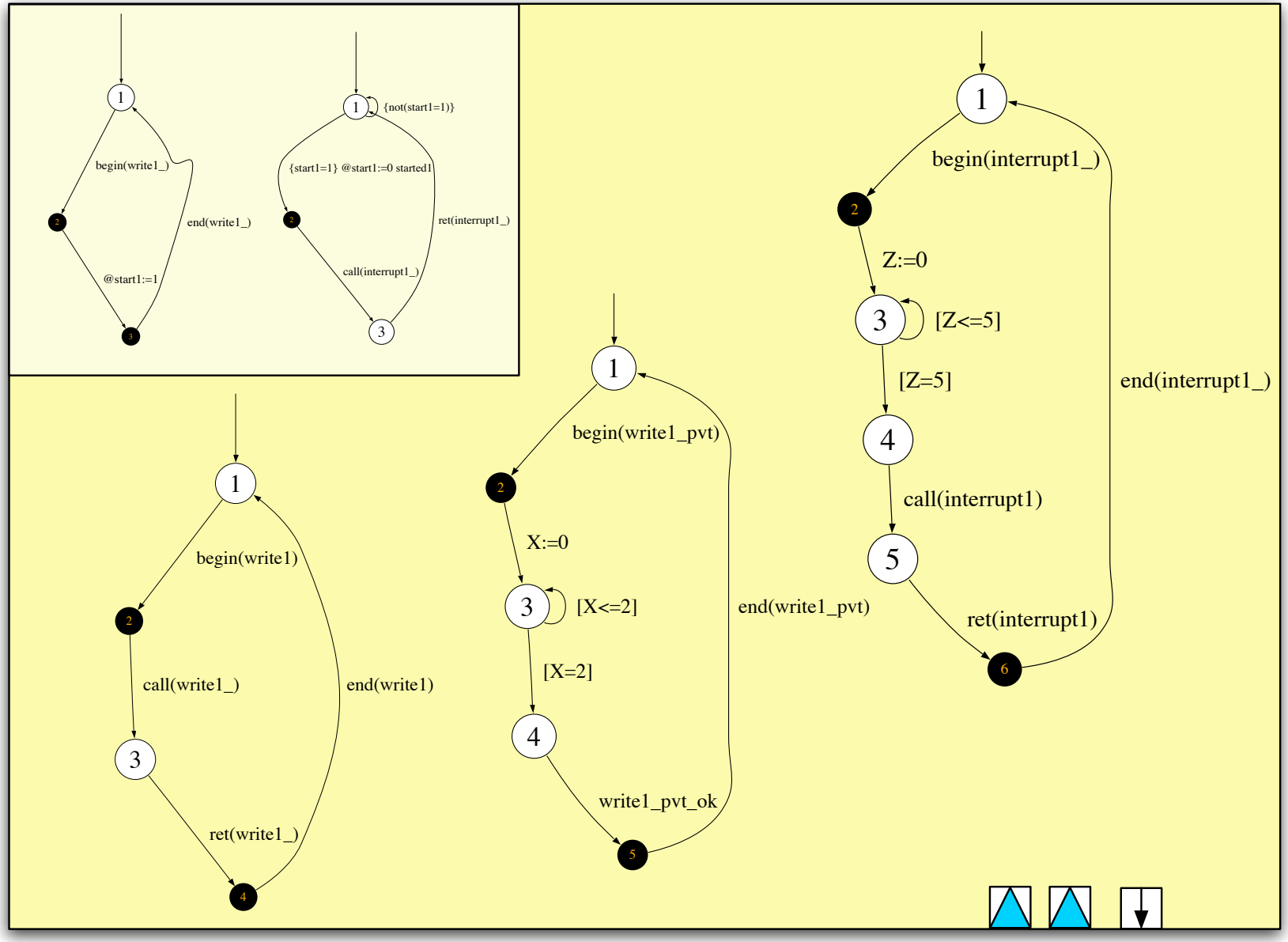
PV Platform's behavior



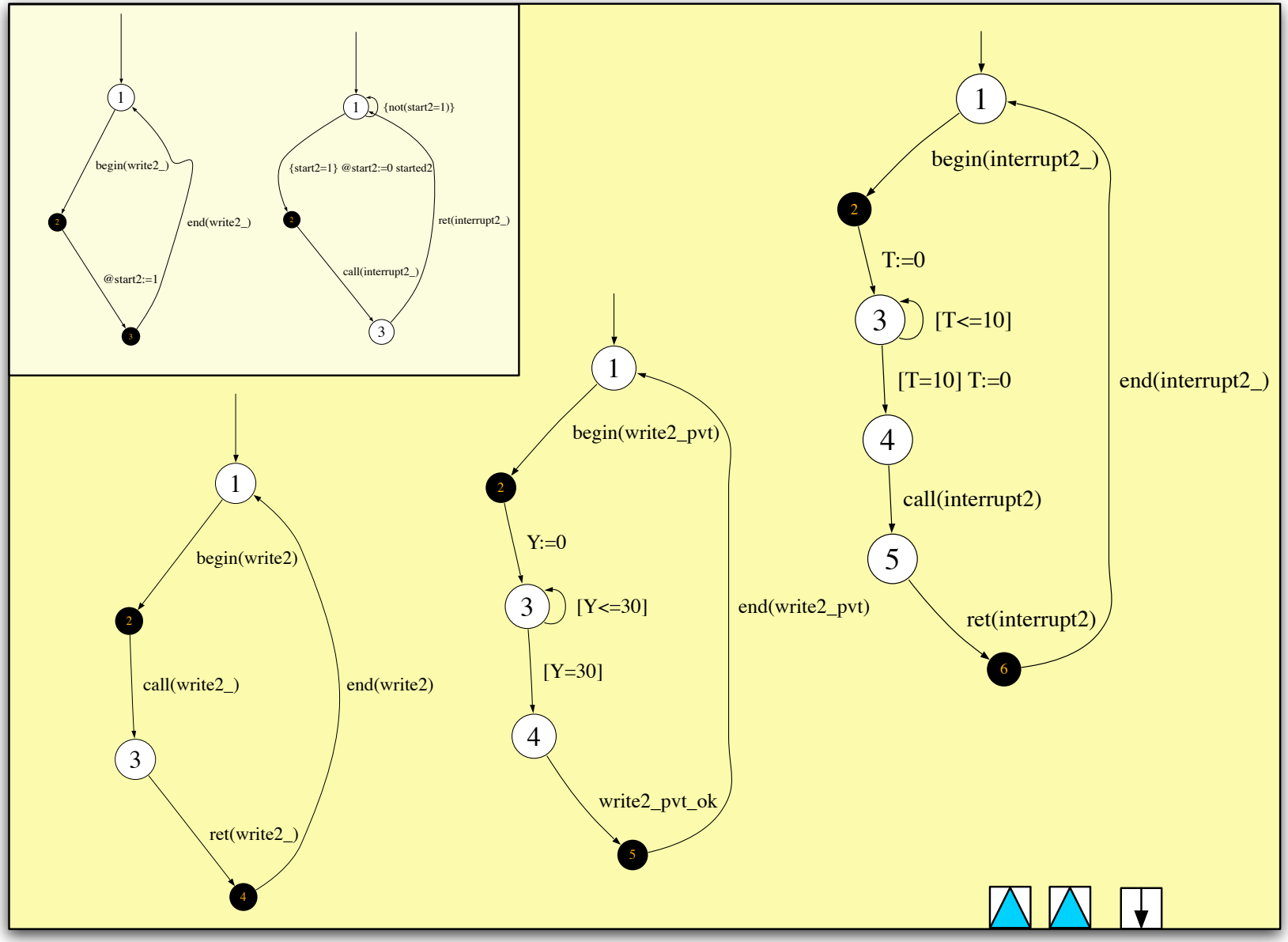
Generator PVT



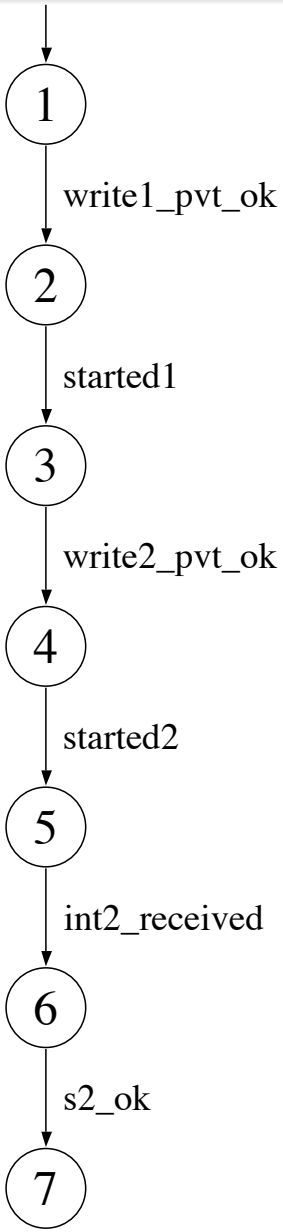
IP #1 PVT


1
begin(write1_)
2
end(write1_)
@start1:=1
3
1
{not(start1=1)}
{start1=1} @start1:=0 started1
2
ret(interrupt1_)
call(interrupt1_)
3
1
begin(write1)
2
call(write1_)
end(write1)
3
ret(write1_)
4
1
begin(write1_pvt)
2
X:=0
3
[X<=2]
end(write1_pvt)
[X=2]
4
write1_pvt_ok
5
1
begin(interrupt1_)
2
Z:=0
3
[Z<=5]
[Z=5]
end(interrupt1_)
4
call(interrupt1)
5
ret(interrupt1)
6

IP #2 PVT


1
begin(write2_)
2
end(write2_)
@start2:=1
3
1
{not(start2=1)}
{start2=1} @start2:=0 started2
2
ret(interrupt2_)
call(interrupt2_)
3
1
begin(write2)
2
call(write2_)
end(write2)
3
ret(write2_)
4
1
begin(write2_pvt)
2
Y:=0
3
[Y<=30]
end(write2_pvt)
[Y=30]
4
write2_pvt_ok
5
1
begin(interrupt2_)
2
T:=0
3
[T<=10]
[T=10] T:=0
end(interrupt2_)
4
call(interrupt2)
5
ret(interrupt2)
6

PVT Platform's behavior



Comparison

- Functional consistency between PV and PVT ?
- Comparison done at platform level
- Formally:

Global functional consistency

PV: micmac automata of PV Platform
PVT: micmac automaton of PVT Platform

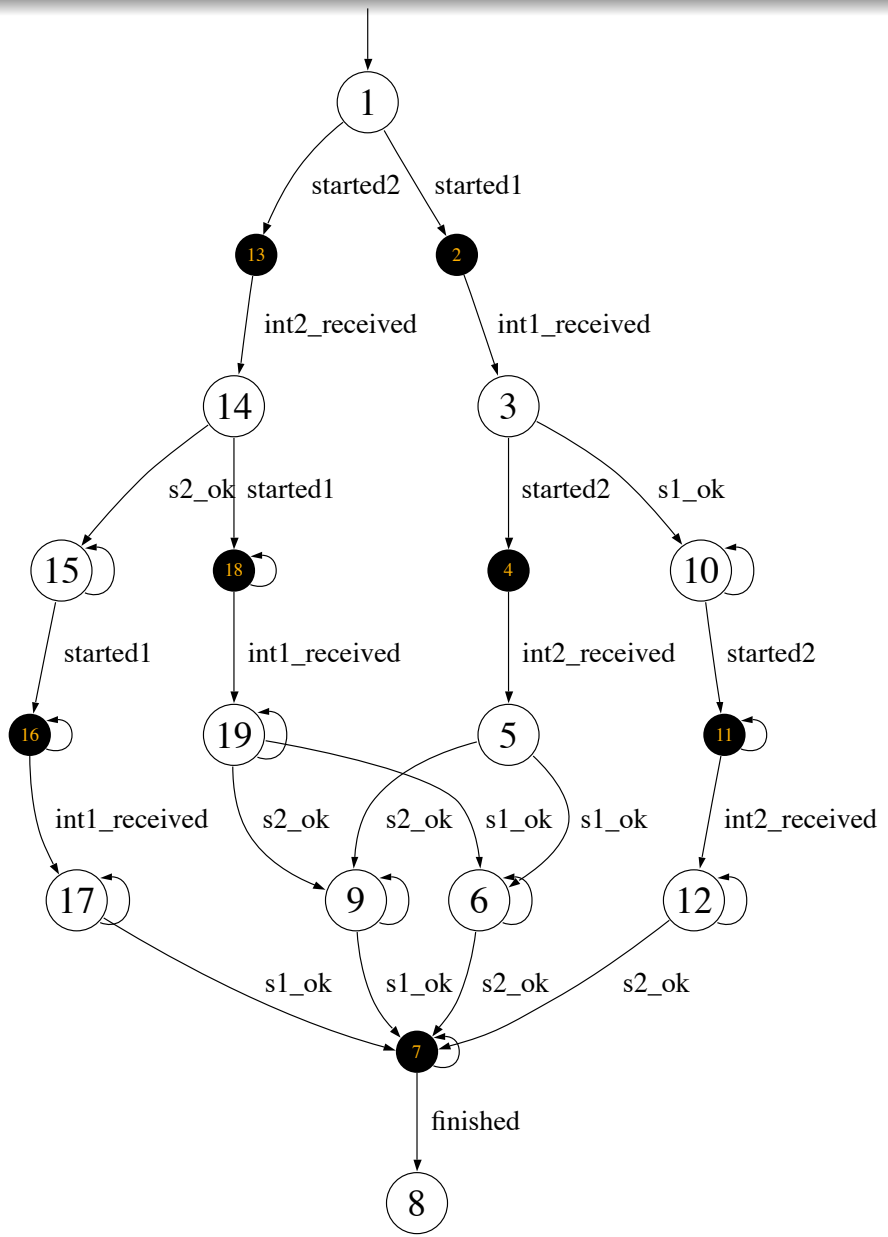
- Using traces:

$$tr(PVT)\,|_{PV} \subseteq tr(PV)$$

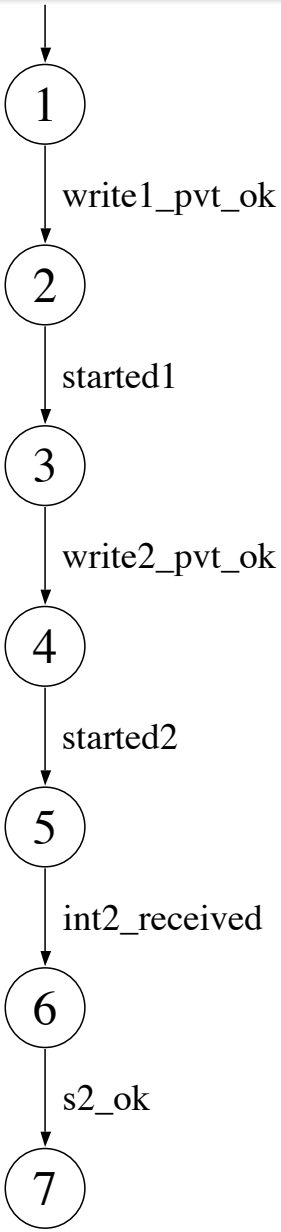
- On automata:

$$PVT|_{PV} \prec PV$$

Comparison

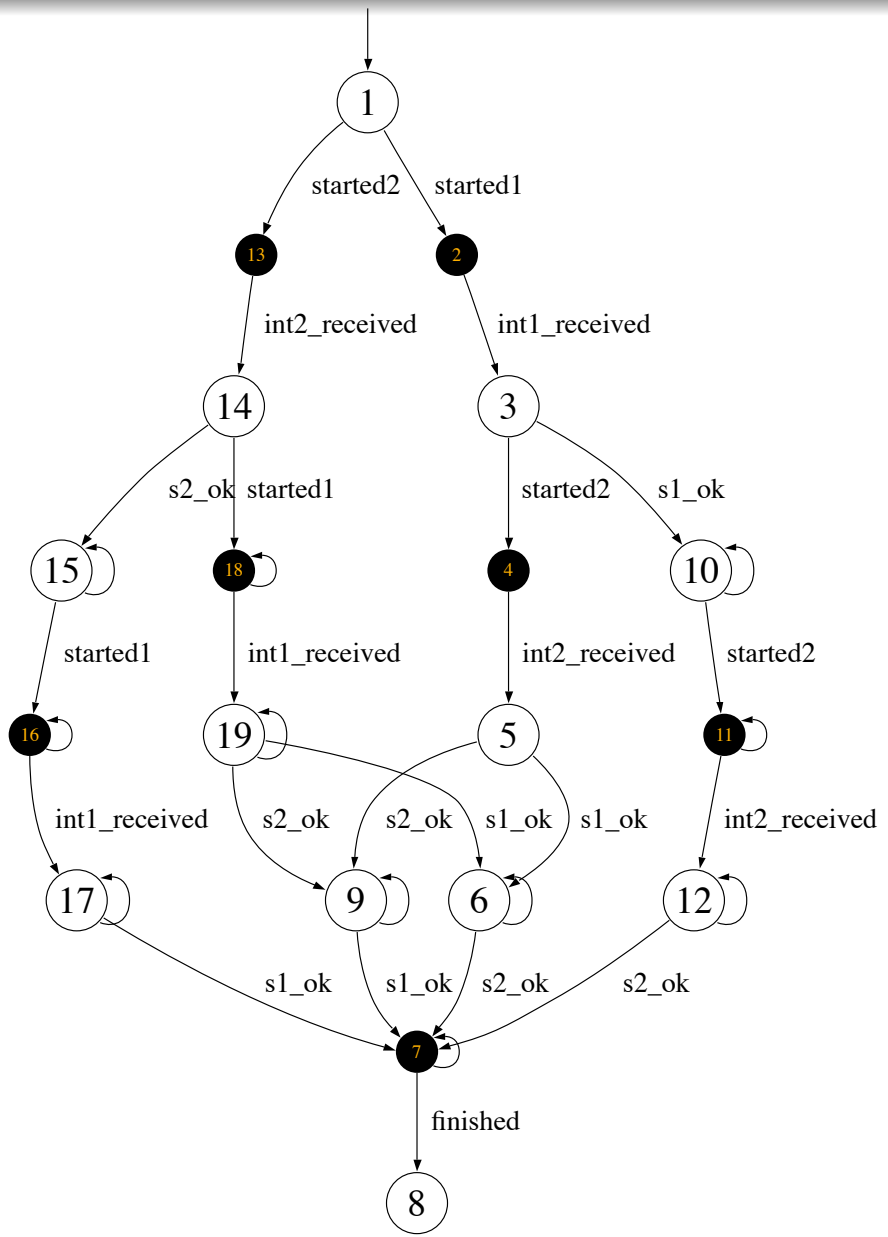


VS

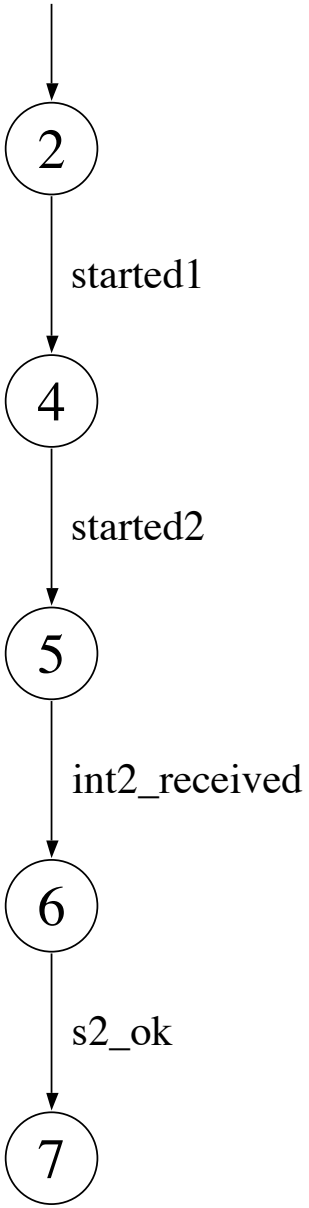


?

Comparison

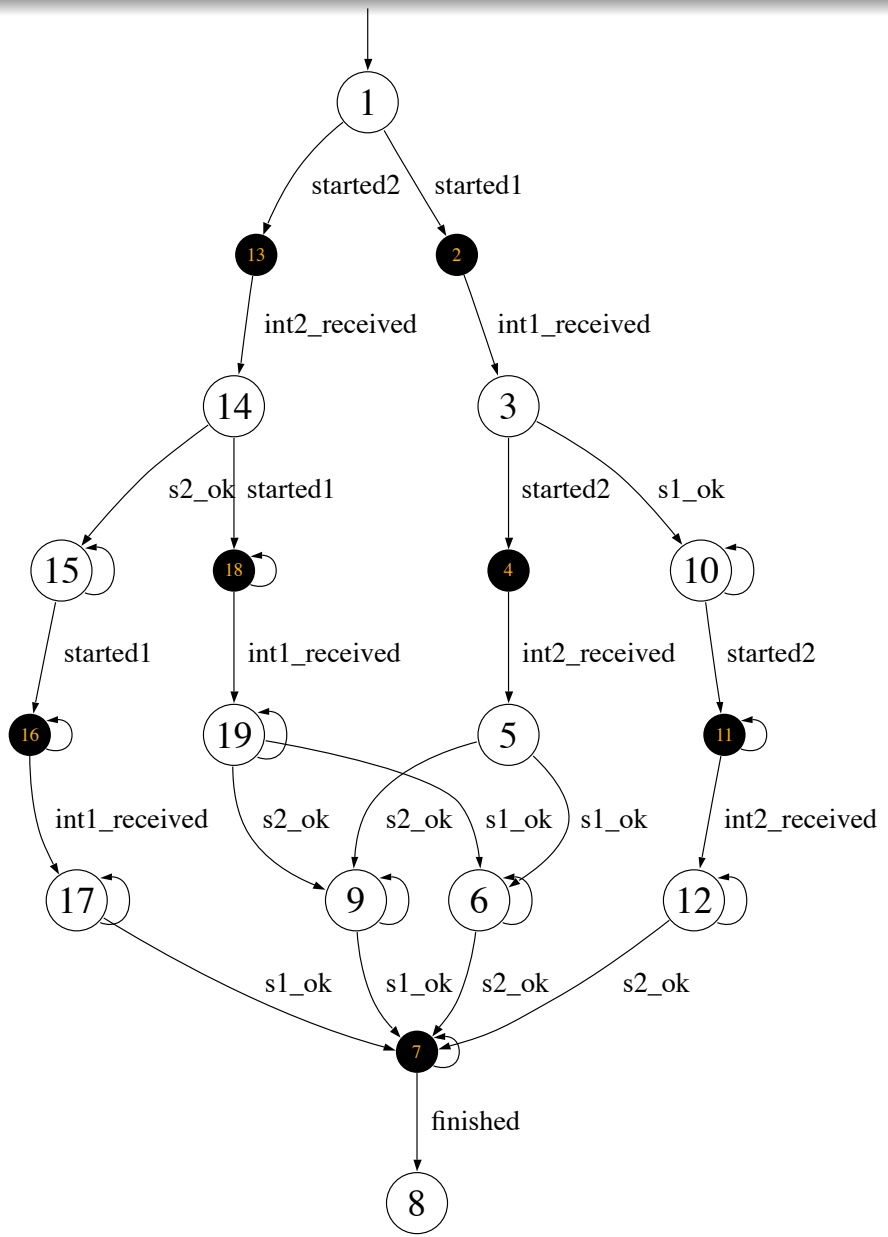

1
started2
started1
13
2
int2_received
int1_received
14
3
s2_ok
started1
started2
s1_ok
15
18
4
10
started1
int1_received
int2_received
started2
16
19
5
11
int1_received
s2_ok
s2_ok
s1_ok
s1_ok
int2_received
17
9
6
12
s1_ok
s1_ok
s2_ok
s2_ok
7
finished
8

≻

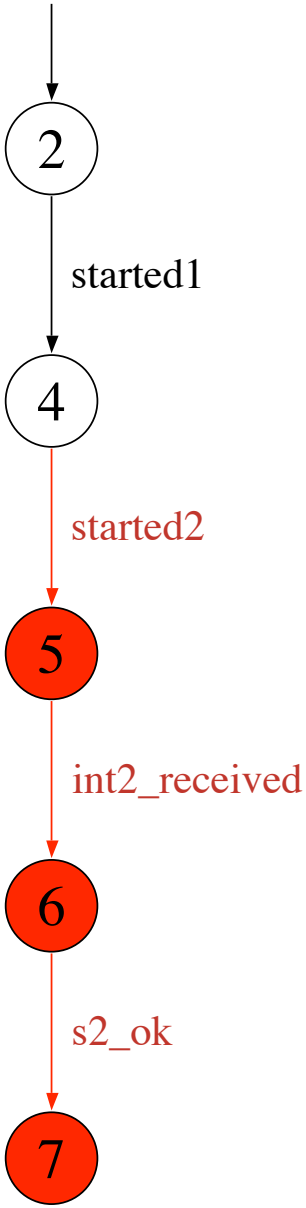

2
started1
4
started2
5
int2_received
6
s2_ok
7

?

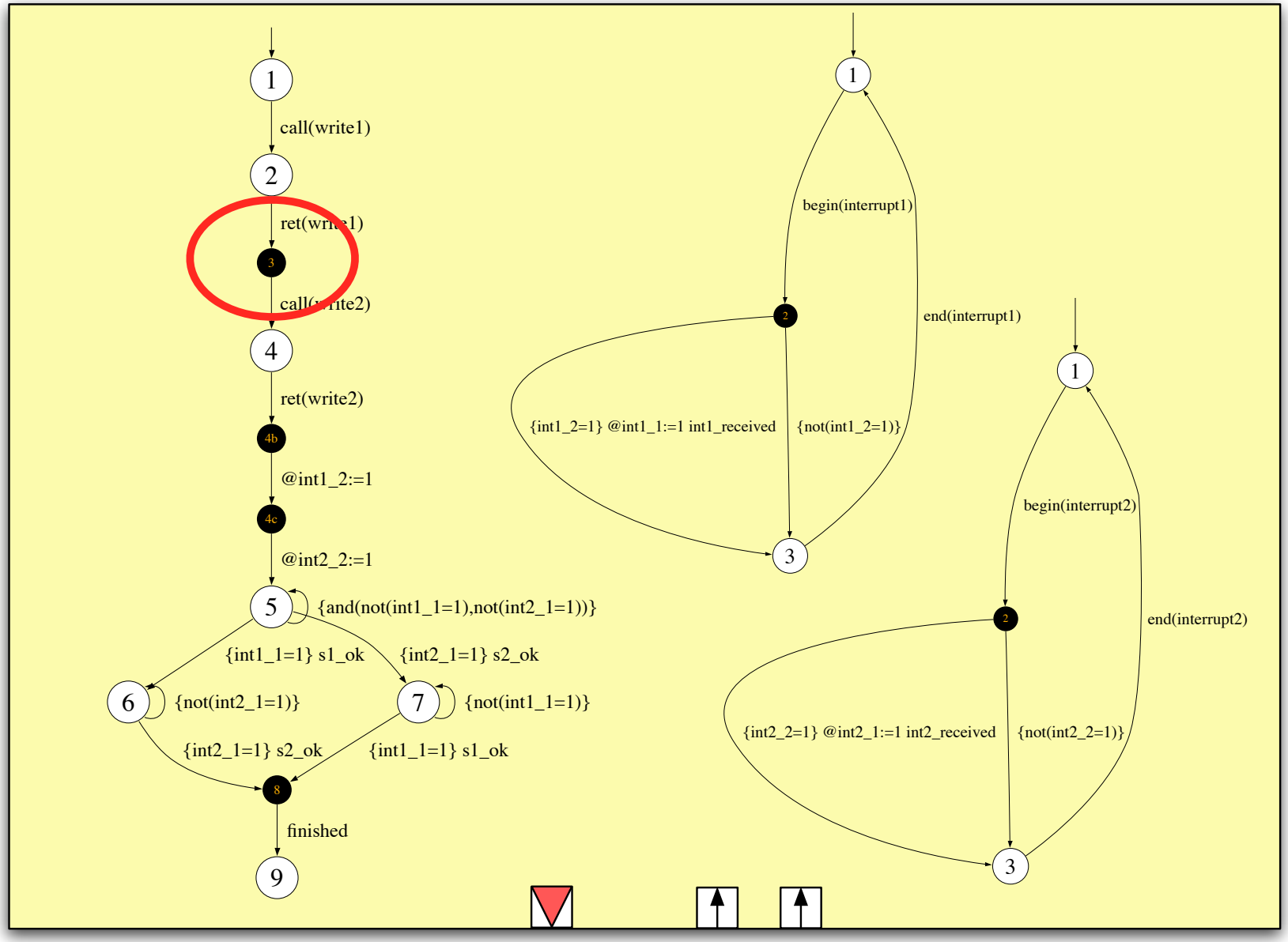
Comparison


1
started2
started1
13
2
int2_received
int1_received
14
3
s2_ok
started1
started2
s1_ok
15
18
4
10
started1
int1_received
int2_received
started2
16
19
5
11
int1_received
s2_ok
s2_ok
s1_ok
s1_ok
int2_received
17
9
6
12
s1_ok
s1_ok
s2_ok
s2_ok
7
finished
8

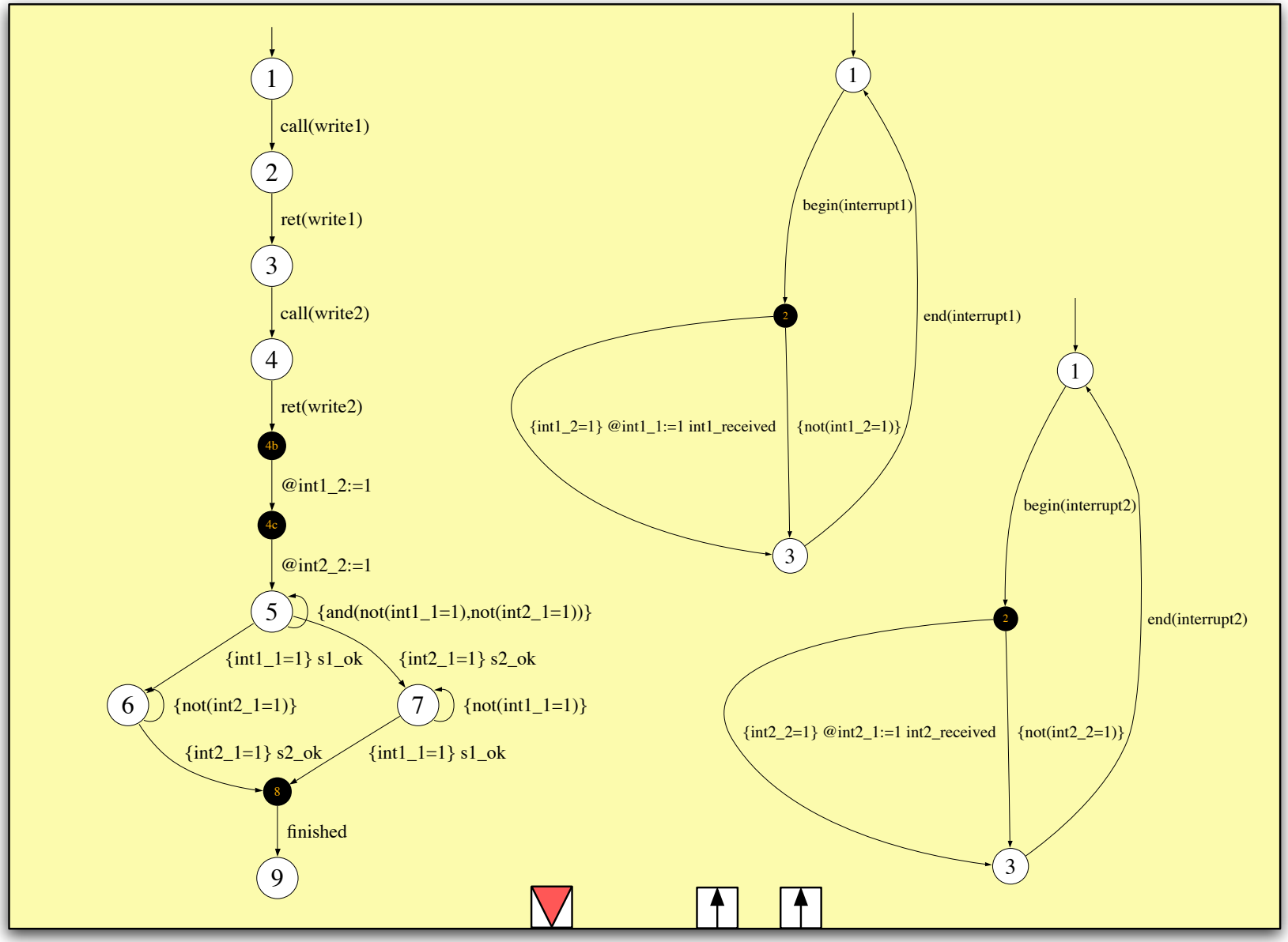
≠


2
started1
4
started2
5
int2_received
6
s2_ok
7

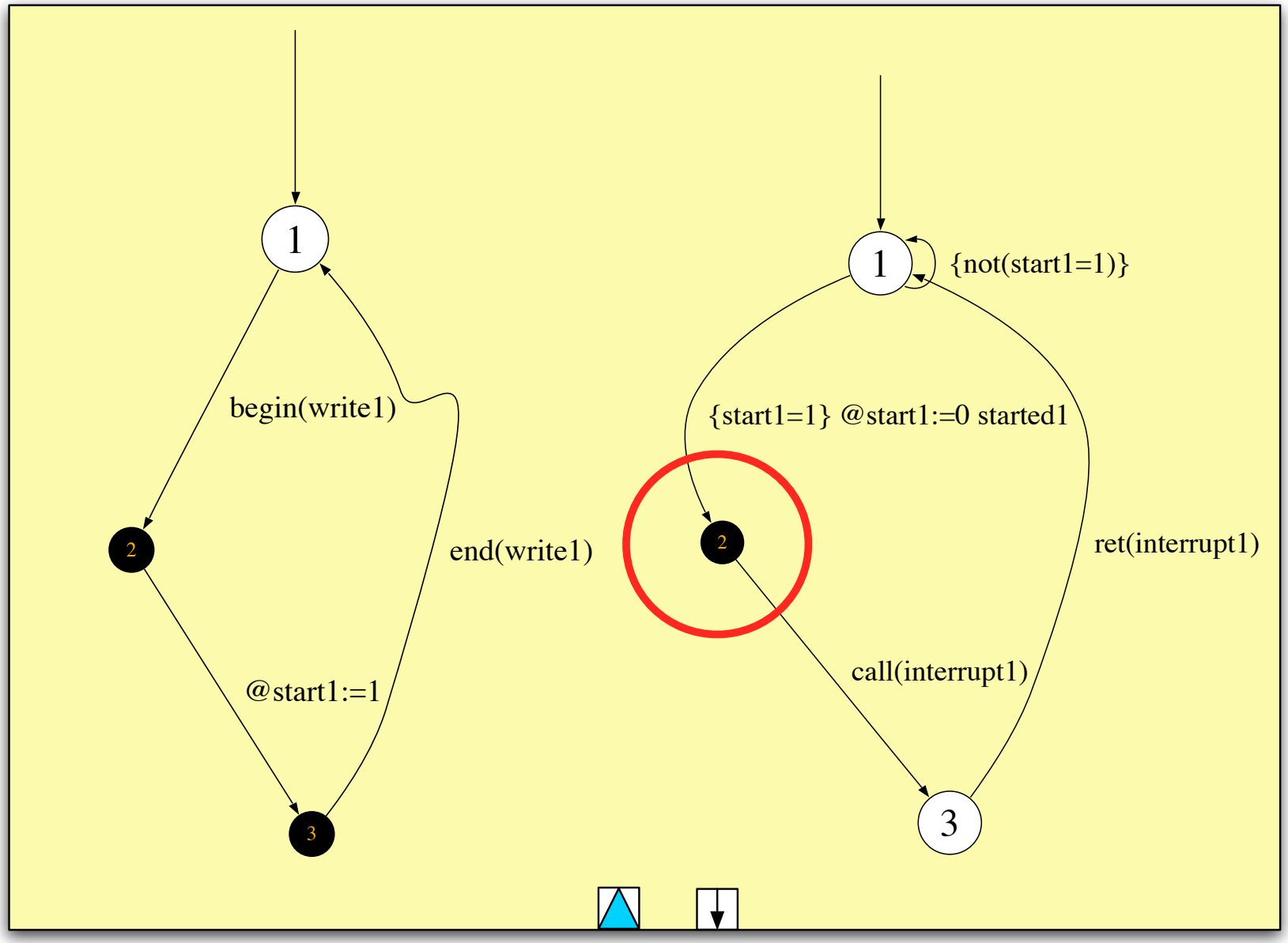
Generator PV bug...



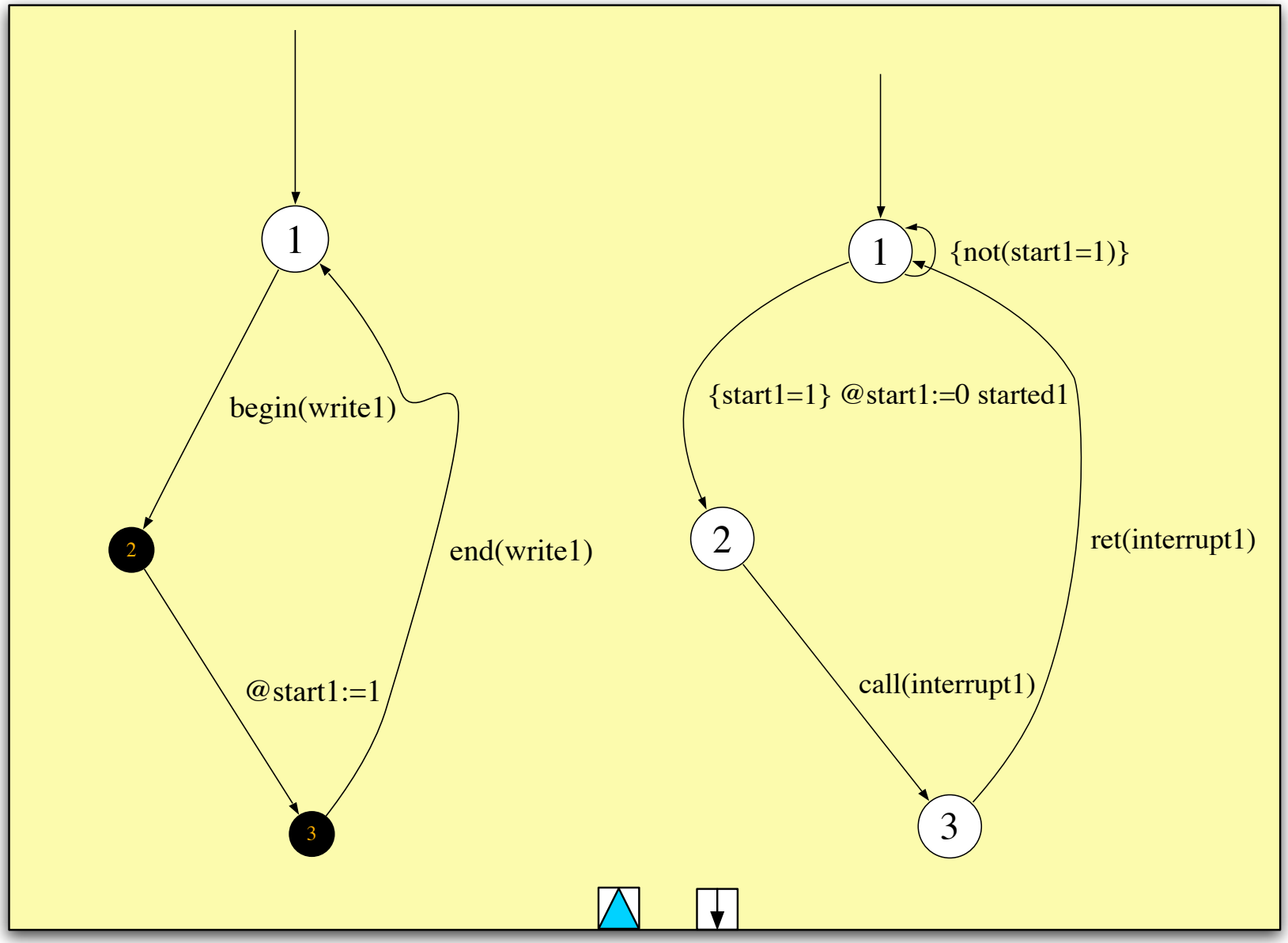
Generator PV fixed



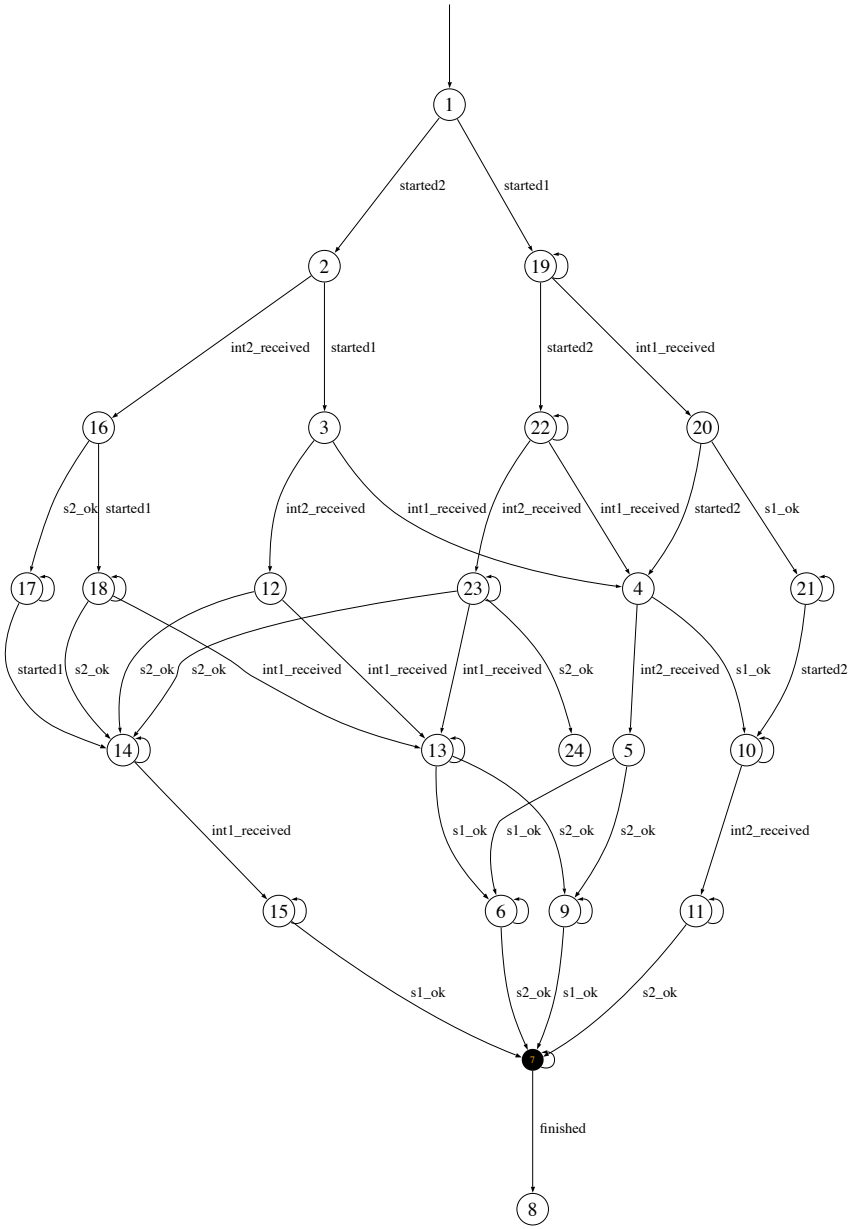
IP #1 PV bug...


1
begin(write1)
2
end(write1)
@start1:=1
3
1
{not(start1=1)}
{start1=1} @start1:=0 started1
2
ret(interrupt1)
call(interrupt1)
3

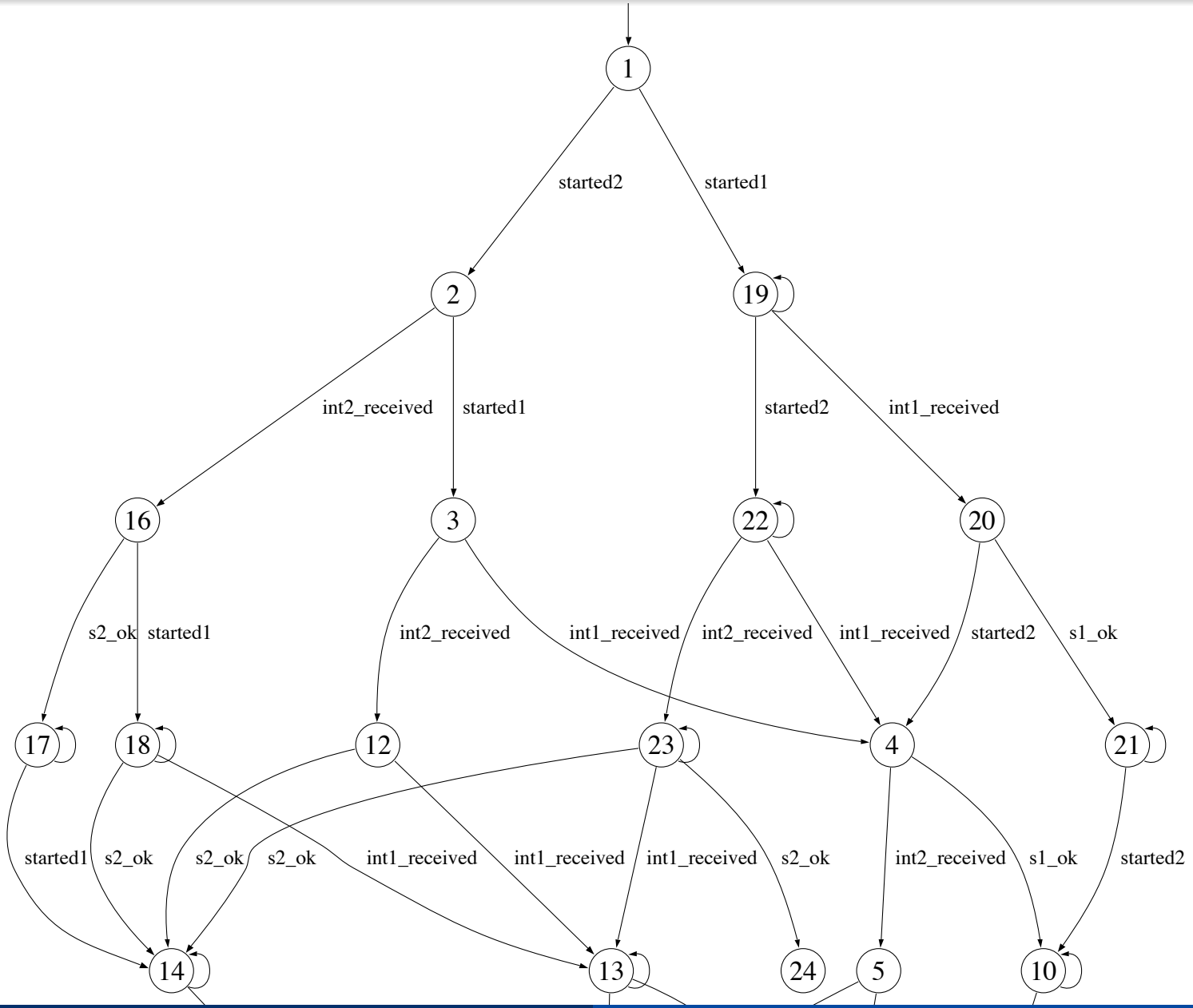
IP #1 PV fixed


1
begin(write1)
2
end(write1)
@start1:=1
3
1
{not(start1=1)}
{start1=1} @start1:=0 started1
2
ret(interrupt1)
call(interrupt1)
3

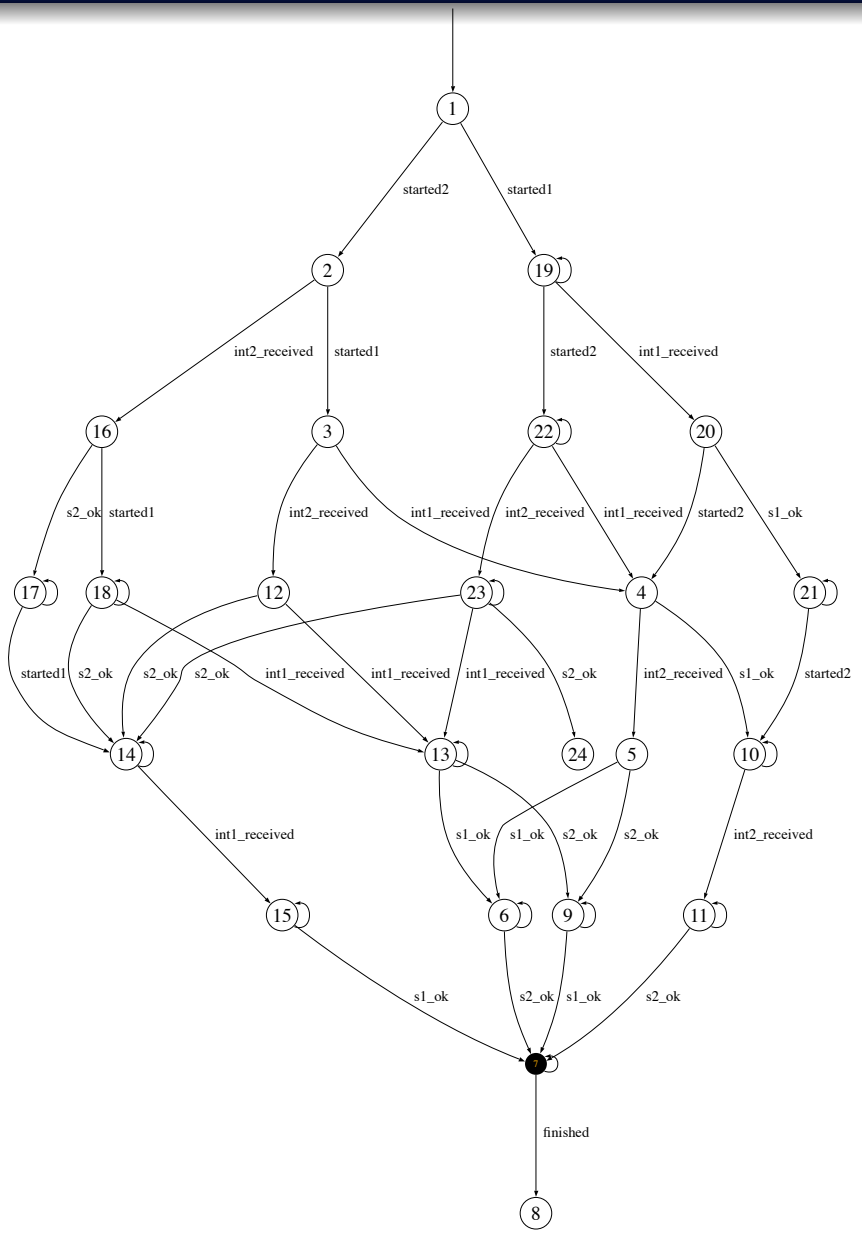
PV fixed Platform's behavior



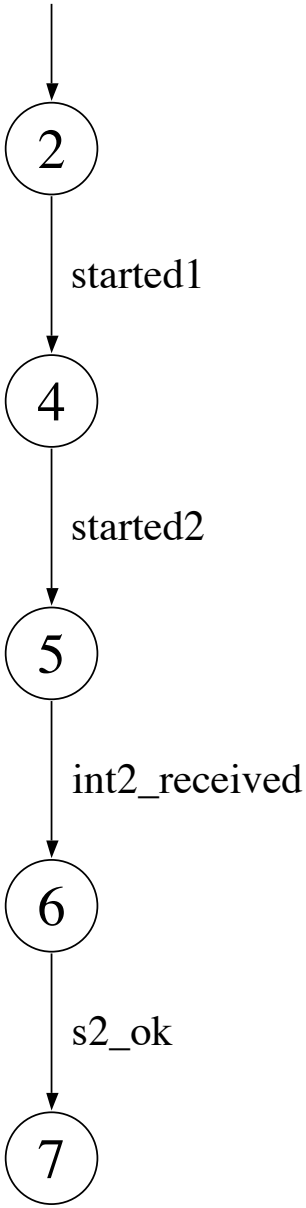
PV fixed Platform's behavior



Comparison (fixed platform)



≻



Compositional Comparison

- Rules about building PVT should be “per component”
- Need for comparison at component level
- Formally:

Local functional consistency

PV_a: micmac automaton of a PV Component a
PVT_a: micmac automaton of a PVT Component a
$\mathcal{C}_{\mathcal{PV}}$: micmac automaton of the rest of the PV platform
$\mathcal{C}_{\mathcal{PVT}}$: micmac automaton of the rest of the PVT platform

- We want to define $\prec^*$ such as

$$PVT_a|_{PV_a} \prec^* PV_a \implies \begin{cases} \forall\, \mathcal{C}_{\mathcal{PVT}}, \mathcal{C}_{\mathcal{PV}} \;/\; \mathcal{C}_{\mathcal{PVT}}|_{\mathcal{C}_{\mathcal{PV}}} \prec^* \mathcal{C}_{\mathcal{PV}}, \\ (PVT_a \times \mathcal{C}_{\mathcal{PVT}})|_{PV_a, \mathcal{C}_{\mathcal{PV}}} \prec^* PV_a \times \mathcal{C}_{\mathcal{PV}} \end{cases}$$

Compositional Comparison

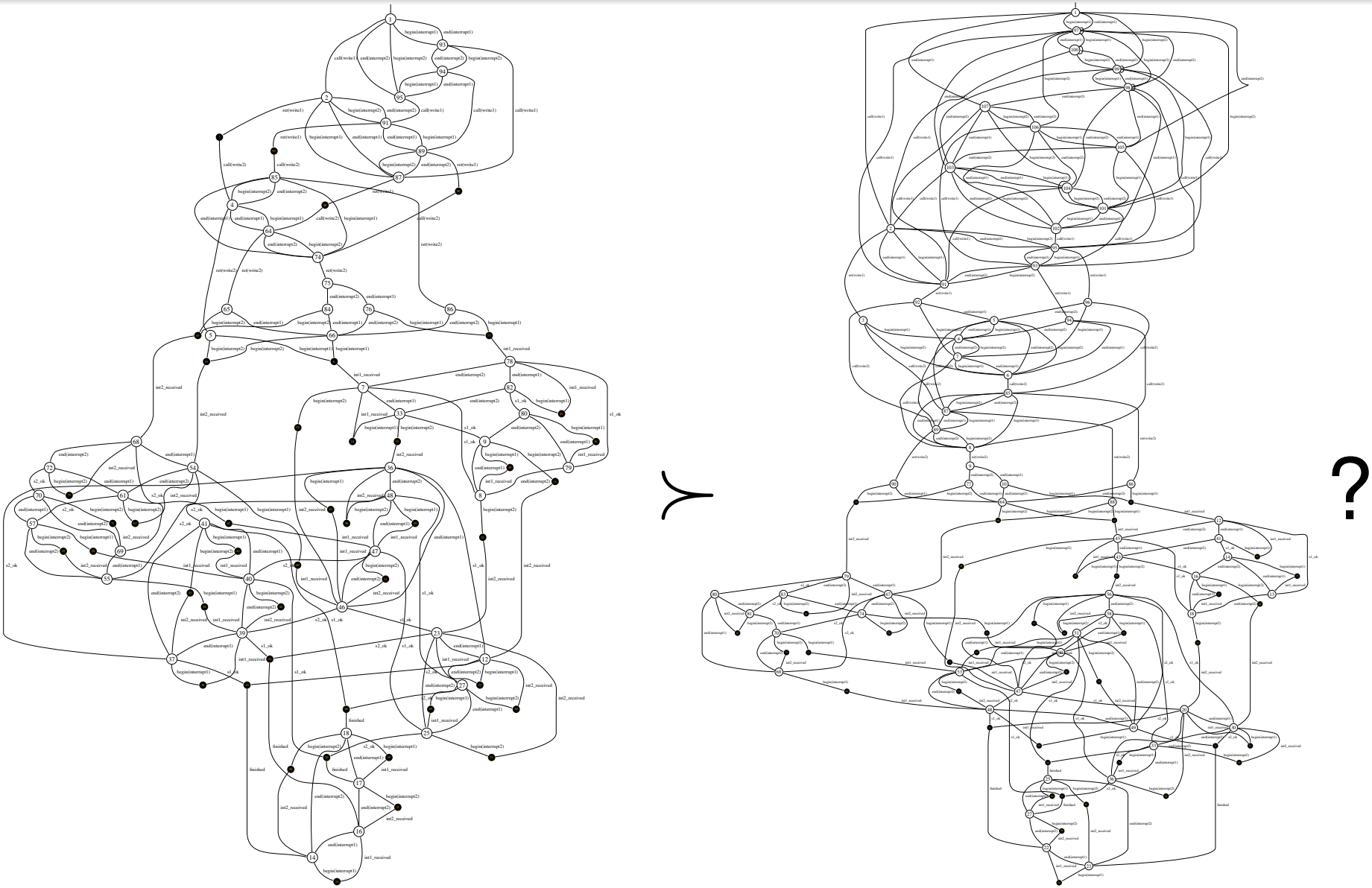
- Result: actually $\prec^* \equiv \prec$

Local functional consistency

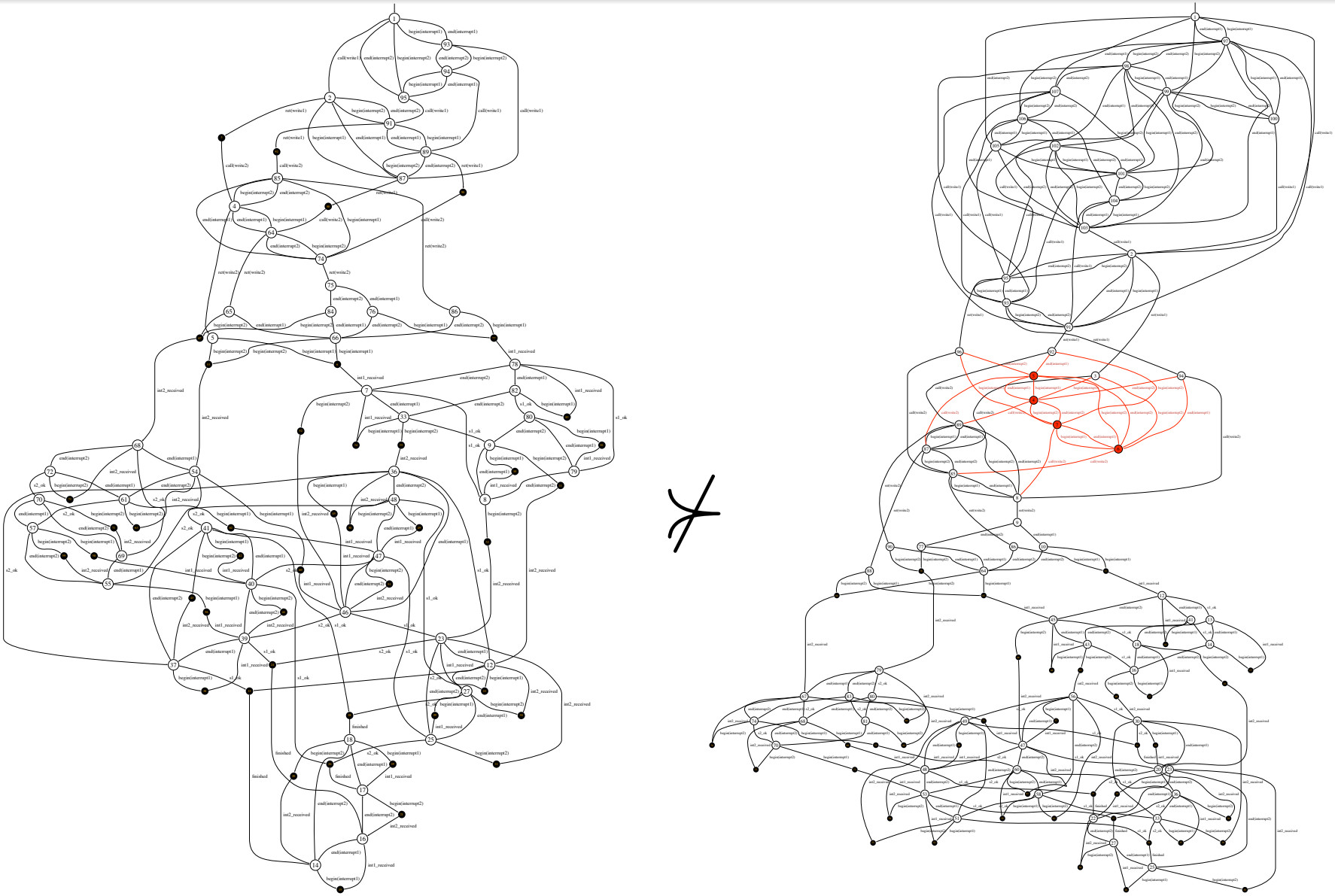
PV_a: micmac automaton of a PV Component a
PVT_a: micmac automaton of a PVT Component a
$\mathcal{C}_{\mathcal{PV}}$: micmac automaton of the rest of the PV platform
$\mathcal{C}_{\mathcal{PVT}}$: micmac automaton of the rest of the PVT platform

$$PVT_a|_{PV_a} \prec PV_a \implies$$
$$\begin{cases} \forall\, \mathcal{C}_{\mathcal{PVT}}, \mathcal{C}_{\mathcal{PV}} \;/\; \mathcal{C}_{\mathcal{PVT}}|_{\mathcal{C}_{\mathcal{PV}}} \prec \mathcal{C}_{\mathcal{PV}}, \\ (PVT_a \times \mathcal{C}_{\mathcal{PVT}})|_{PV_a, \mathcal{C}_{\mathcal{PV}}} \prec PV_a \times \mathcal{C}_{\mathcal{PV}} \end{cases}$$

Comparison for Generator (bugged version)



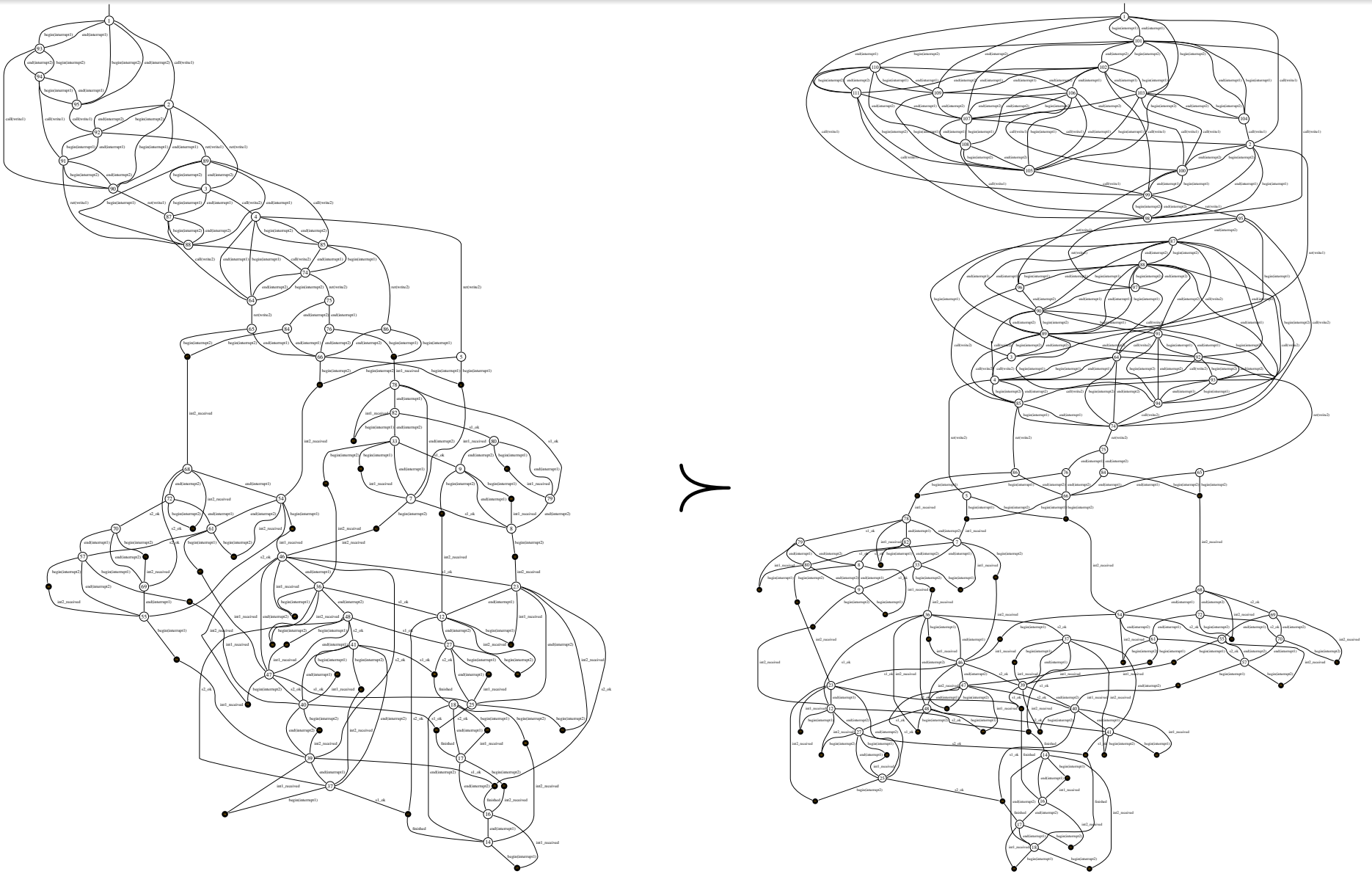
Comparison for Generator (bugged version)



Diagnostic analysis

- Diagnostic gives for instance:
  call(write1) ret(write1) begin(interrupt1) end(interrupt1) ...
- $\longrightarrow$ interrupt1 call not supposed to arrive before call to write2 in bugged PV version
- PVT transform works only on PVs following some rules...
- Bugged PVs wrongly model reality

Comparison for Generator (fixed version)



Conclusion

- Micmac automata
  - Reflect general semantics of SystemC
    - Non-preemption
    - Time
  - Ability to encode very quickly complex behavior of the SystemC specification
- PV/PVT transform
  - Ability to study functionality preservation on a specific example
  - Hopes for Proof Genericity:
    - Examples are sufficiently general
    - Limited set of synchronisation schemes

Future work

- Study remaining synchronisation schemes
- Extend Micmac Automata function calls to return a value
- Summarise/Formalise PV/PVT rules
- Other properties to guarantee about the process
- Transfer?
  - Rules for writing models
  - T submodels library
  - TLM synchronisations library?

Bibliography

Matthieu Moy, Florence Maraninchi, and Laurent Maillet-Contoz.

LusSy: an open Tool for the Analysis of Systems-on-a-Chip at the Transaction Level.

Design Automation for Embedded Systems, 10(2-3):73–104, September 2005.

Matthieu Moy.

Techniques and Tools for the Verification of Systems-on-a-Chip at the Transaction Level.

PhD thesis, Institut National Polytechnique de Grenoble, December 2005.