Monitoring Distributed Controllers

When an Efficient LTL Algorithm on Sequences is Needed to Model-Check Traces

A. Genon T. Massart C. Meuter

Université Libre de Bruxelles
Département d'Informatique

August 25, 2006

ULB

Need for validation

Distributed control systems

- concurrent processes
- running on physically distributed hardware
- hard to design in nature
- critical systems (e.g. plant control system, ...)

Need for validation

Distributed control systems

- concurrent processes
- running on physically distributed hardware
- hard to design in nature
- critical systems (e.g. plant control system, ...)

Validation techniques

Model-Checking:

- exhaustive verification
- but: not (yet) scalable to real-sized systems

Testing and Monitoring:

- non-exhaustive verification
- scalable for real-sized systems
- widely used in industry

Monitoring

The system

- distributed and asynchronous
- instrumented to emit relevant events
  (e.g. variable assignments, message transfer)
- an execution trace is collected

Monitoring

The system

- distributed and asynchronous
- instrumented to emit relevant events
  (e.g. variable assignments, message transfer)
- an execution trace is collected

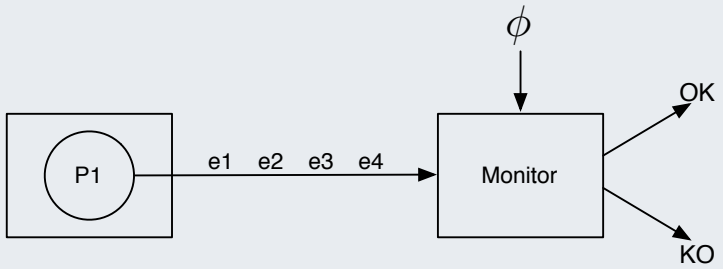
The monitor

- seperate process which collects those events
- checks whether a certain property holds
- can be done online or offline

Centralized v.s. Distributed Systems

Centralized System

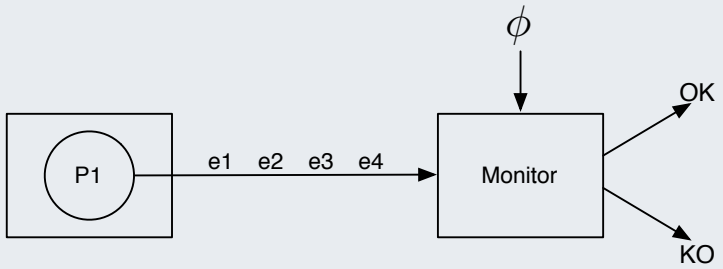
- only one process
- events are totally ordered



Centralized v.s. Distributed Systems

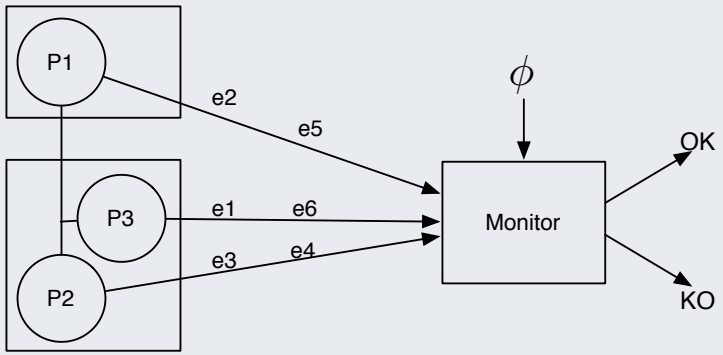
Centralized System

- only one process
- events are totally ordered



Distributed System

- multiple processes
- events are not totally ordered
- but a partial order can be obtained using vector clocks [Lam78, Mat89]



Related Works

Global predicate detection

Several classes of predicate have been studied:

- Stable predicates [CL85] (once true, remains true)
- Disjunctive predicates (of the form $lp_1 \vee lp_2 \vee \cdots \vee lp_n$)
- Conjunctive predicates [GW94, GW96] (of the form $lp_1 \wedge lp_2 \wedge \cdots \wedge lp_n$)
- Observer independent predicates [CDF95] (such that $\forall\Diamond\phi \iff \exists\Diamond\phi$)
- Linear, semi-linear [CG98] or regular predicates (RCTL logic) [GM01, SG03]

Related Works

Global predicate detection

Several classes of predicate have been studied:

- Stable predicates [CL85] (once true, remains true)
- Disjunctive predicates (of the form $lp_1 \vee lp_2 \vee \cdots \vee lp_n$)
- Conjunctive predicates [GW94, GW96] (of the form $lp_1 \wedge lp_2 \wedge \cdots \wedge lp_n$)
- Observer independent predicates [CDF95] (such that $\forall\Diamond\phi \iff \exists\Diamond\phi$)
- Linear, semi-linear [CG98] or regular predicates (RCTL logic) [GM01, SG03]

Trace model checking

Temporal logics for Mazurkiewicz traces:

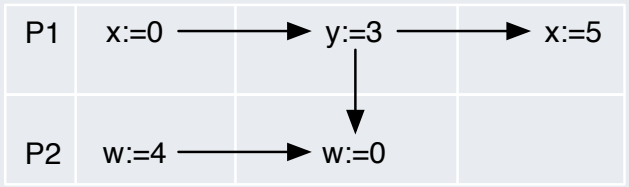
- TrPTL [Thi94]
- TCL [APP95]
- LTrL [TW02]
- LTL over traces [DG02]

$\Longrightarrow \neq$ our approach

Traces and Monitors

Traces

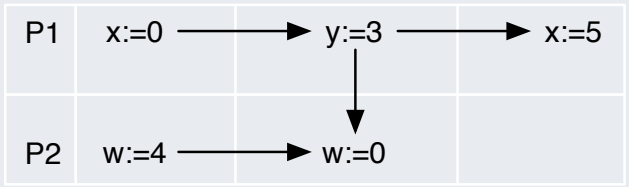
- partially ordered set of events
- events are labelled with assignments
- finite number of events



Traces and Monitors

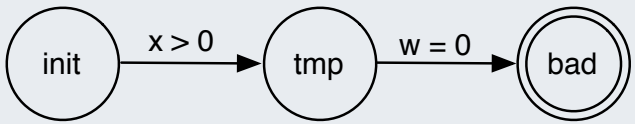
Traces

- partially ordered set of events
- events are labelled with assignments
- finite number of events



Monitor

- non-deterministic finite automaton with guards on transitions
- built from an LTL formula using tableau construction [VW86]
- loop transitions are ommitted and the monitor never blocks
- each guard is a boolean formula on one variable
- some states are marked as bad



The problem

Question

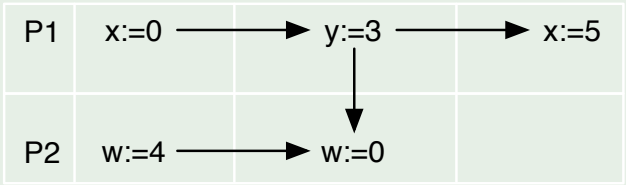
Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

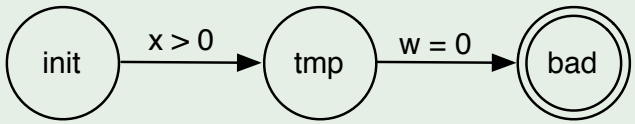
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example



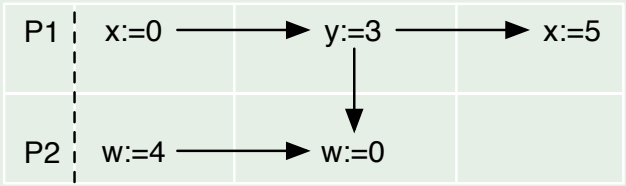


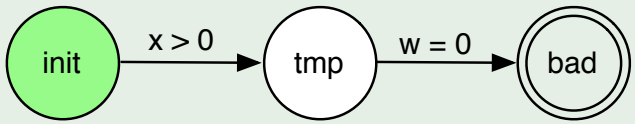
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example



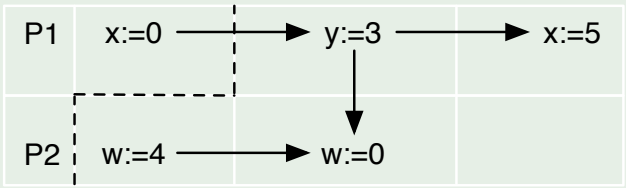


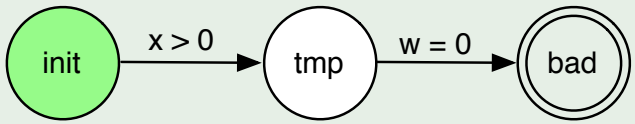
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example





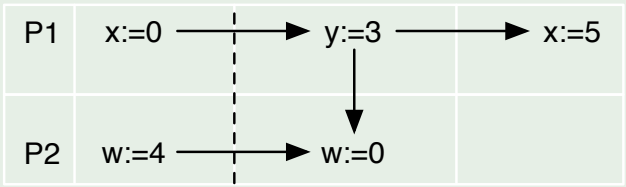
x:=0

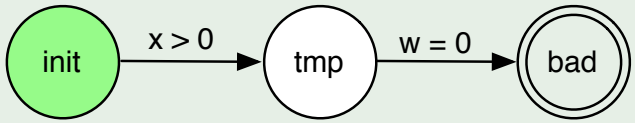
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example





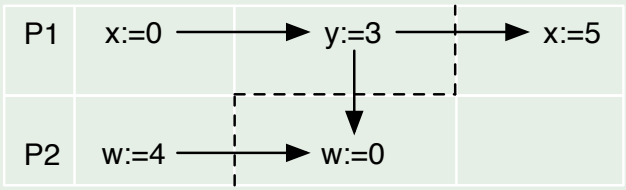
x:=0; w:=4

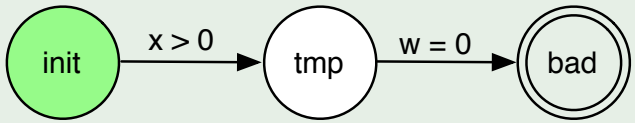
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example





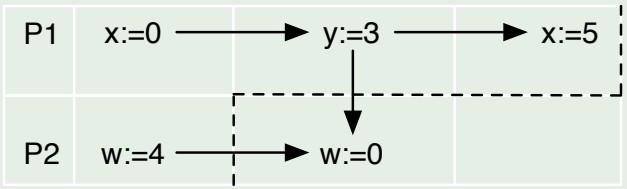
x:=0; w:=4; y:=3

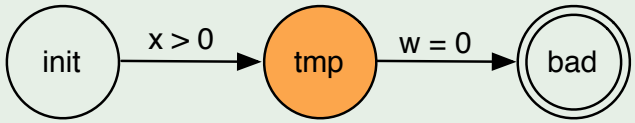
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example





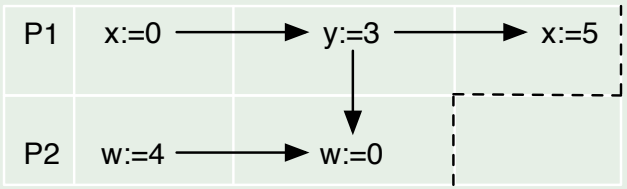
x:=0; w:=4; y:=3; x:=5

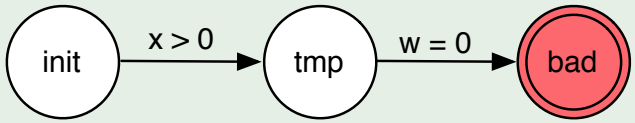
The problem

Question

Does there exist an interleaving (i.e. total order) of events compatible with the partial order, leading the monitor to a bad state?

Example



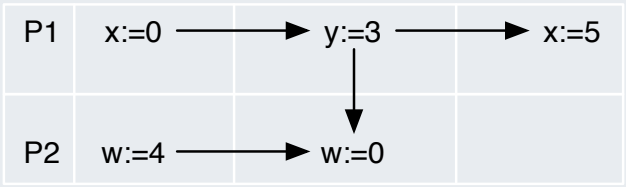


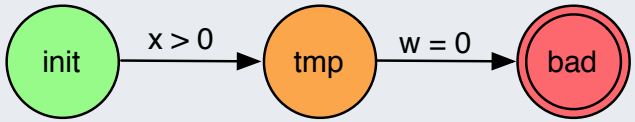
x:=0; w:=4; y:=3; x:=5; w:=0

The problem

A simple solution

- Compose the trace with the monitor to explore all possibilities:

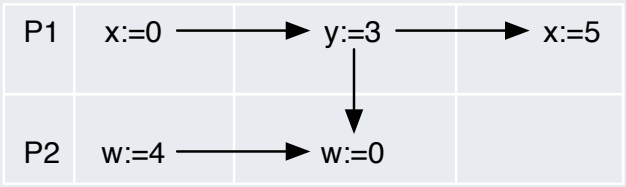


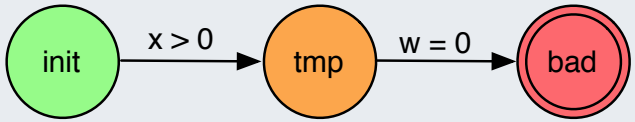


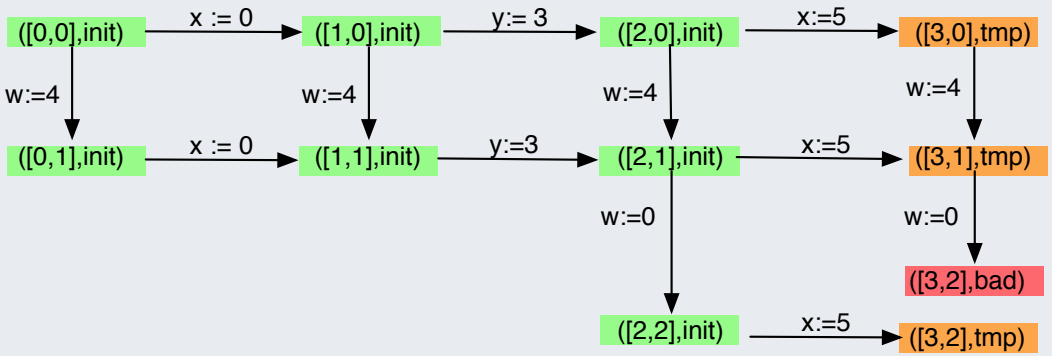
The problem

A simple solution

- Compose the trace with the monitor to explore all possibilities:



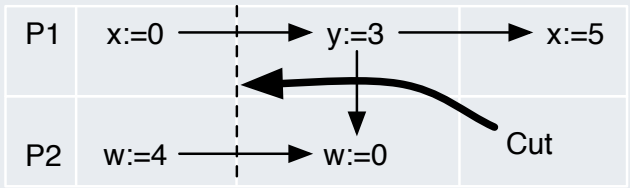


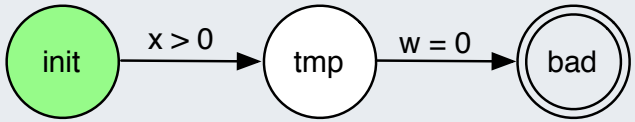


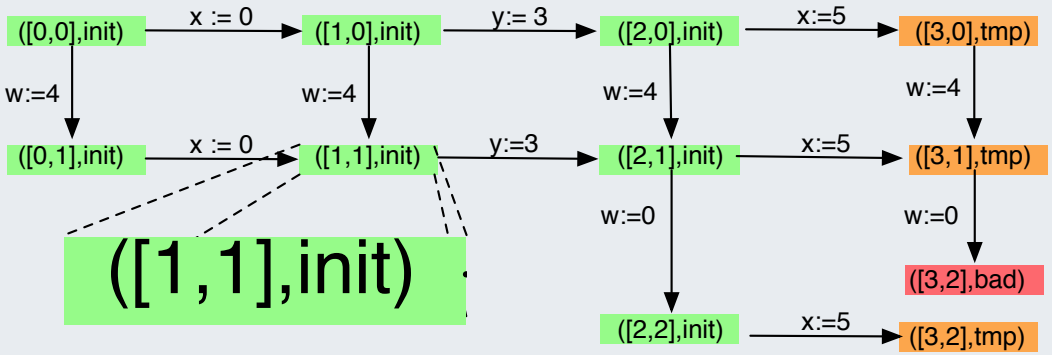
The problem

A simple solution

- Compose the trace with the monitor to explore all possibilities:





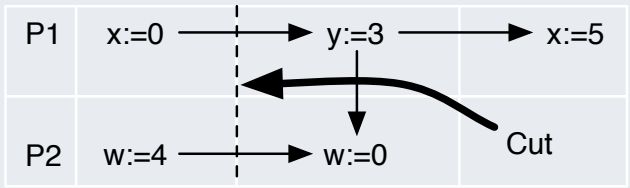


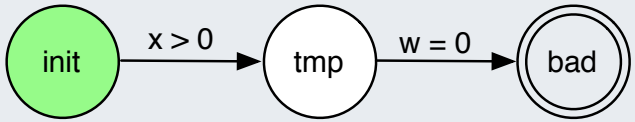
- Configuration = Cut + Monitor State

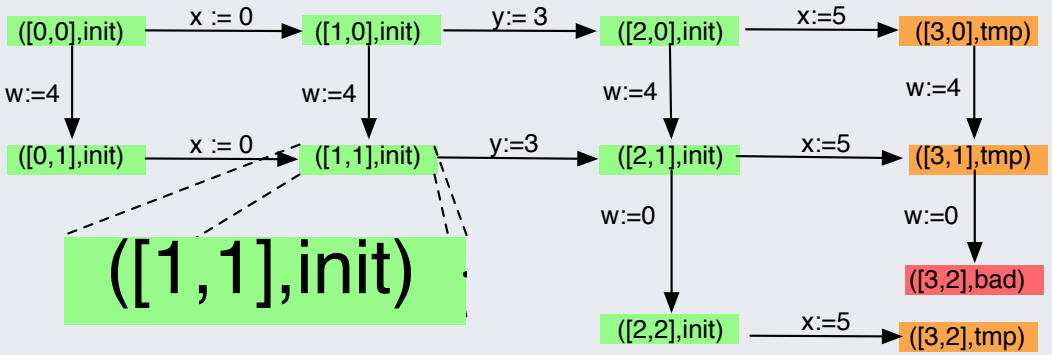
The problem

A simple solution

- Compose the trace with the monitor to explore all possibilities:







- Configuration = Cut + Monitor State
- Problem: exponential number of interleavings/configurations!

$\Longrightarrow$ Can we do better?

Complexity result

Theorem

The trace monitoring problem is NP-Complete

Complexity result

Theorem

The trace monitoring problem is NP-Complete

NP-easyness

Simple non-deterministic algorithm:

- guess an interleaving of the events in the trace
- check in polynomial time if it leads the monitor to a bad state

Complexity result

Theorem

The trace monitoring problem is NP-Complete

NP-easyness

Simple non-deterministic algorithm:

- guess an interleaving of the events in the trace
- check in polynomial time if it leads the monitor to a bad state

NP-hardness

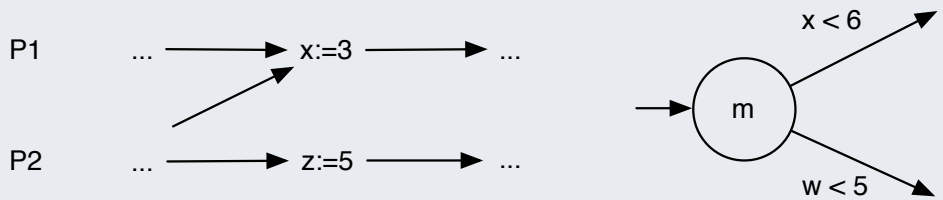
Reduction from 3-SAT problem:

- use the monitor to encode the formula ϕ
- use the trace to encode all possible valuations
- $\exists$ a valuation satisfying ϕ iff the bad state is reachable

Sensitive events

Observation

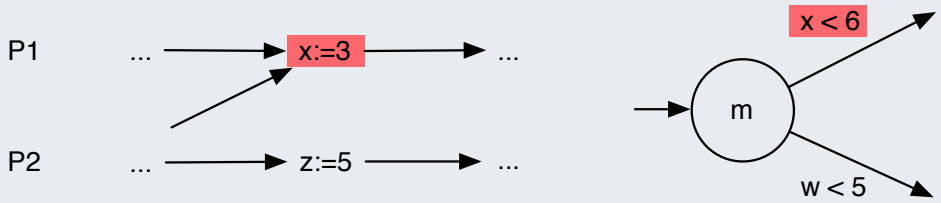
- The monitor is not always sensitive to all events:



Sensitive events

Observation

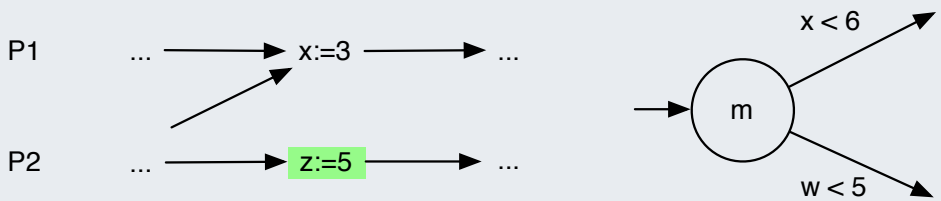
- The monitor is not always sensitive to all events:



Sensitive events

Observation

- The monitor is not always sensitive to all events:

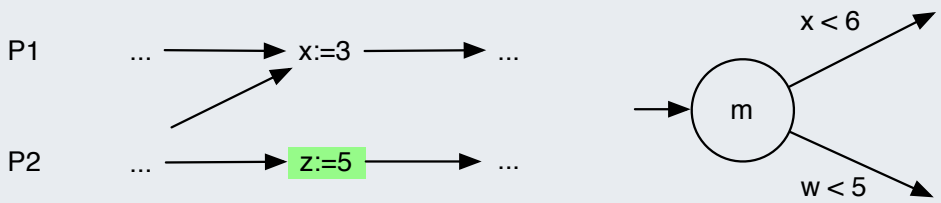


- Firing non-sensitive events has no effect on the monitor state

Sensitive events

Observation

- The monitor is not always sensitive to all events:



- Firing non-sensitive events has no effect on the monitor state

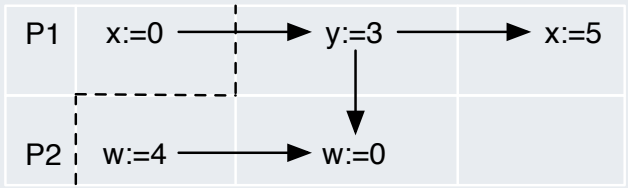
Can we do something about it?

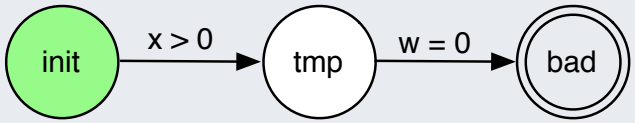
- Exploring all interleavings of non-sensitive events is not necessary
  $\Longrightarrow$ explore only one arbitrary interleaving
- But : those events might be interesting in another state of the monitor
  $\Longrightarrow$ keep them seperate during exploration

Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order

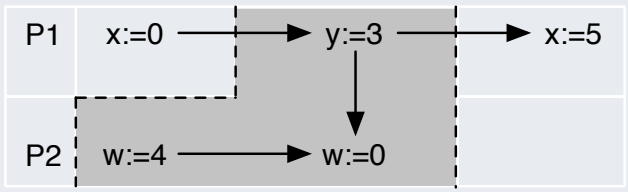


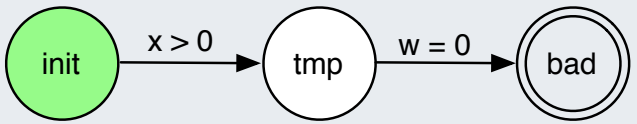


Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order

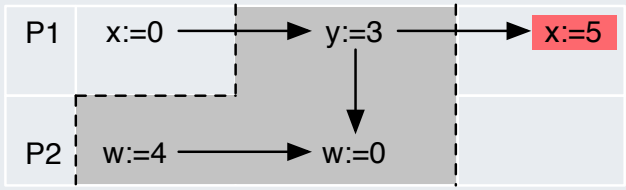


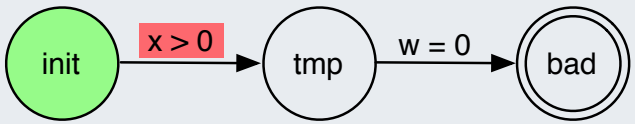


Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order

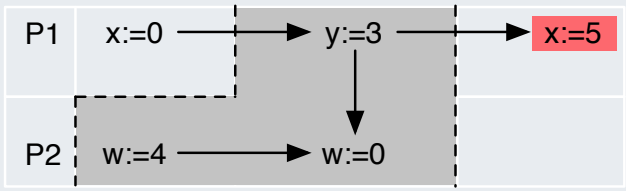




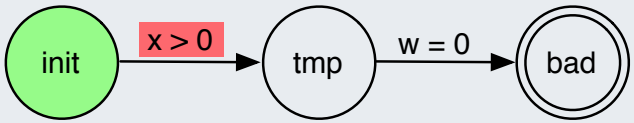
Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept



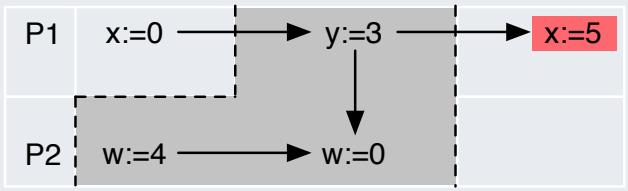
$[1,0] \longrightarrow [2,2]$



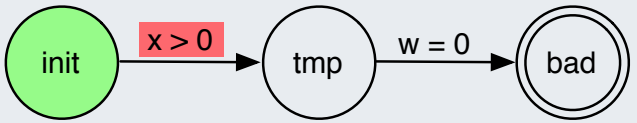
Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept
3. Then, sensitive events are fired on all the cuts in between, in one step



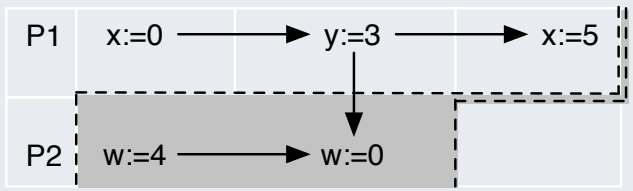
$[1,0] \longrightarrow [2,2]$



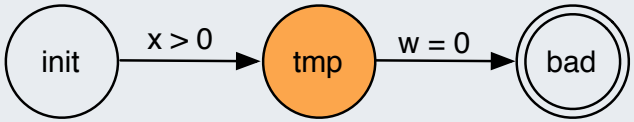
Symbolic Exploration

Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept
3. Then, sensitive events are fired on all the cuts in between, in one step



$[3,0] \longrightarrow [3,2]$

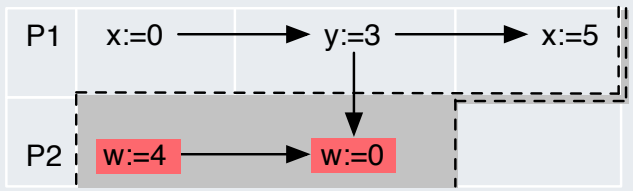


Symbolic Exploration

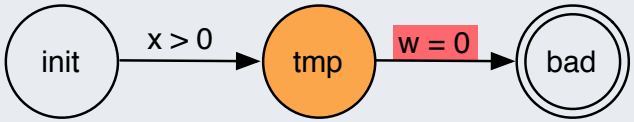
Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept
3. Then, sensitive events are fired on all the cuts in between, in one step
4. Some previously non-sensitive events become sensitive

$\Longrightarrow$ they must be explored!



$[3,0] \longrightarrow [3,2]$

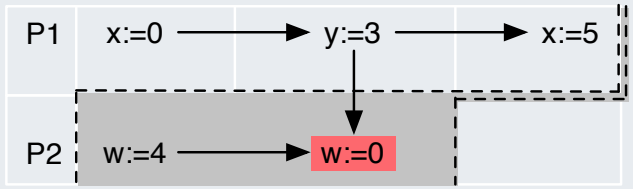


Symbolic Exploration

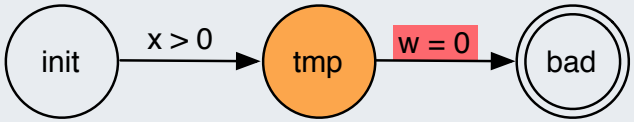
Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept
3. Then, sensitive events are fired on all the cuts in between, in one step
4. Some previously non-sensitive events become sensitive

$\Longrightarrow$ they must be explored!



$[3,0] \longrightarrow [3,2]$

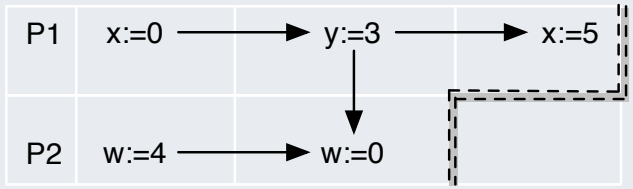


Symbolic Exploration

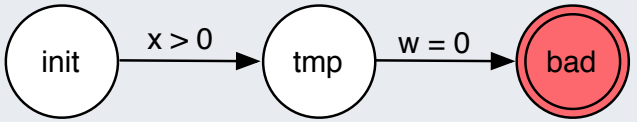
Idea

1. First all non-sensitive events are fired in an arbitrary order
2. Both the origin cut and the final cut are kept
3. Then, sensitive events are fired on all the cuts in between, in one step
4. Some previously non-sensitive events become sensitive

$\Longrightarrow$ they must be explored!



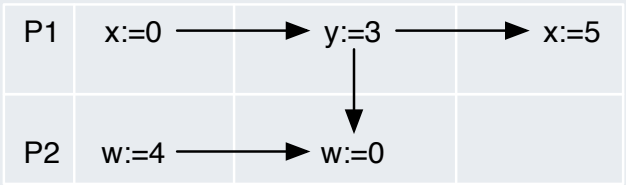
$[3,2] \longrightarrow [3,2]$

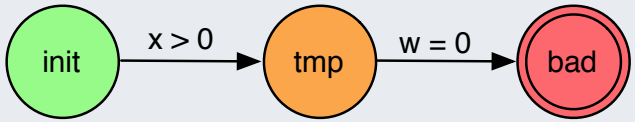


Symbolic Exploration (cont'd)

Symbolic Composition

- Symbolically compose the trace with the monitor to explore all possibilities:

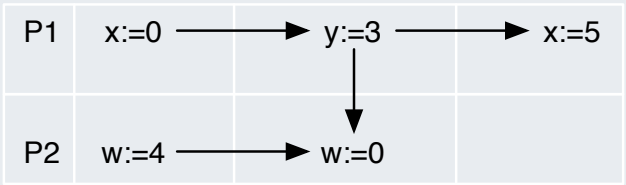


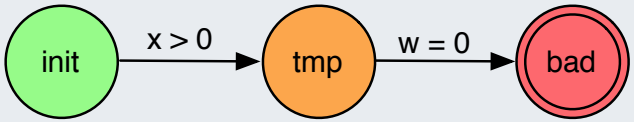


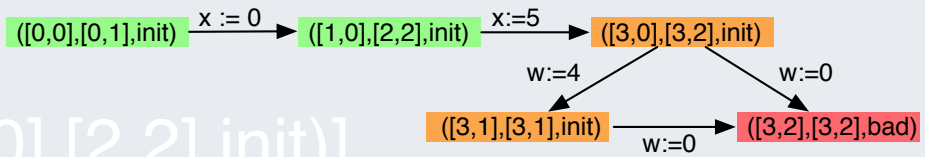
Symbolic Exploration (cont'd)

Symbolic Composition

- Symbolically compose the trace with the monitor to explore all possibilities:





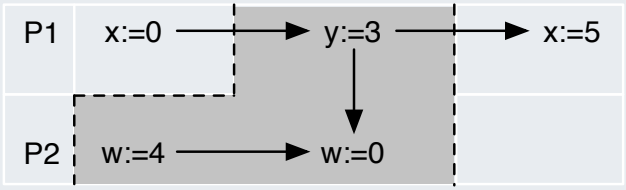


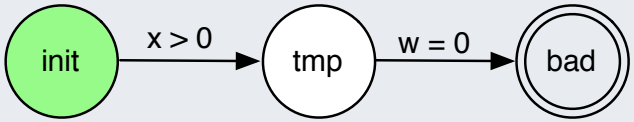
([1,0],[2,2],init)]

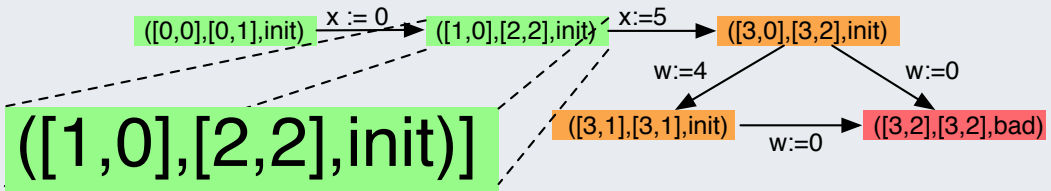
Symbolic Exploration (cont'd)

Symbolic Composition

- Symbolically compose the trace with the monitor to explore all possibilities:





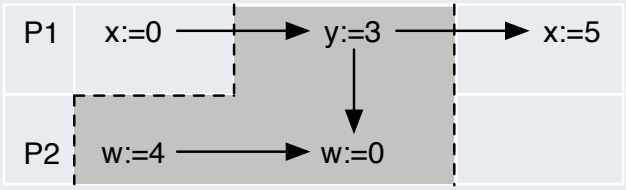


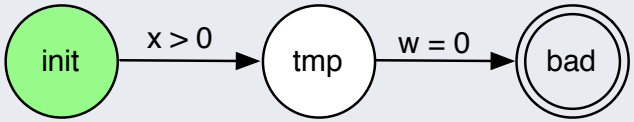
- Symbolic Configuration = Origin Cut + Final Cut + Monitor State

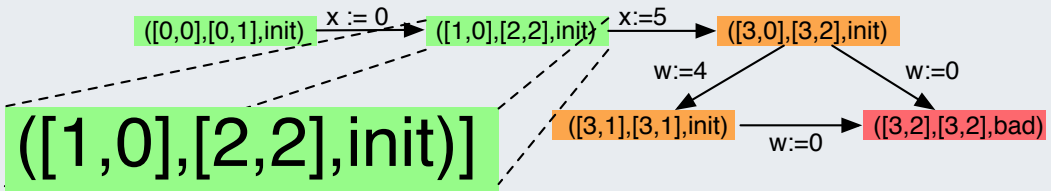
Symbolic Exploration (cont'd)

Symbolic Composition

- Symbolically compose the trace with the monitor to explore all possibilities:







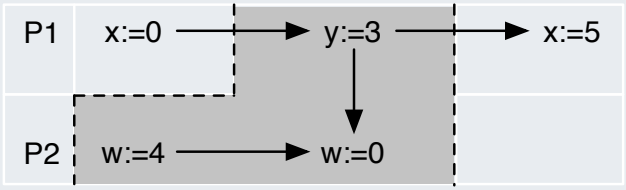
- Symbolic Configuration = Origin Cut + Final Cut + Monitor State
- One symbolic configuration represents a set of configurations:

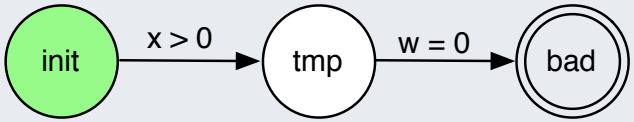
$$([1,0],[2,2],\text{init}) \equiv \{ ([1,0],\text{init}), ([1,1],\text{init}), ([1,2],\text{init}), ([2,2],\text{init}) \}$$

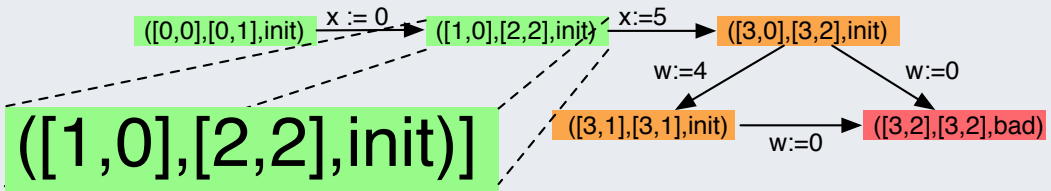
Symbolic Exploration (cont'd)

Symbolic Composition

- Symbolically compose the trace with the monitor to explore all possibilities:







- Symbolic Configuration = Origin Cut + Final Cut + Monitor State
- One symbolic configuration represents a set of configurations:

  ([1,0],[2,2],init) ≡ { ([1,0],init), ([1,1],init), ([1,2],init), ([2,2],init) }
- Symbolic composition is both sound and complete!

In practice

Symbolic Exploration Algorithm

input : $\mathcal{T} = (E, \lambda, \preceq), \mathcal{M} = (M, m^0, B, \rightarrow_m)$
output: $\mathsf{reachable}_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$
begin
  $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
  while $W \neq \emptyset$ **do**
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$
    $T \leftarrow T \cup (t, w, m)$
    repeat
      $x \leftarrow w$
      $w \leftarrow w \cup \{e \in \mathsf{enabled}(w) \mid e \notin \mathsf{sensitive}(m)\}$
    until $(w = x)$
    if $(w = E) \wedge (m \in B)$ **then**
      return false
    forall $(t', w', m') : (t, w, m) \xrightarrow{e}_s (t', w', m') \wedge (t', w', m') \notin T$ **do**
      $W \leftarrow W \cup \{(t', w', m')\}$
  return true
end

In practice

Symbolic Exploration Algorithm

input : $\mathcal{T} = (E, \lambda, \preceq), \mathcal{M} = (M, m^0, B, \rightarrow_m)$
output: $\text{reachable}_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$
begin
  $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
  while $W \neq \emptyset$ **do**
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$
    $T \leftarrow T \cup (t, w, m)$
    repeat
      $x \leftarrow w$
      $w \leftarrow w \cup \{e \in \text{enabled}(w) \mid e \notin \text{sensitive}(m)\}$
    until $(w = x)$
    if $(w = E) \wedge (m \in B)$ **then**
      return false
    forall $(t', w', m') : (t, w, m) \xrightarrow{e}_s (t', w', m') \wedge (t', w', m') \notin T$ **do**
      $W \leftarrow W \cup \{(t', w', m')\}$
  return true
end

1. pick a symbolic configuration

In practice

Symbolic Exploration Algorithm

input : $\mathcal{T} = (E, \lambda, \preceq), \mathcal{M} = (M, m^0, B, \rightarrow_m)$
output: reachable$_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$
begin
  $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
  while $W \neq \emptyset$ **do**
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$

    repeat
      $x \leftarrow w$
      $w \leftarrow w \cup \{e \in \mathsf{enabled}(w) \mid e \notin \mathsf{sensitive}(m)\}$
    until $(w = x)$

    return false

    forall $(t', w', m') : (t, w, m) \xrightarrow{e}_s (t', w', m') \wedge (t', w', m') \notin T$ **do**
      $W \leftarrow W \cup \{(t', w', m')\}$

  return true
end

1. pick a symbolic configuration
2. extend it with non-sensitive events

In practice

Symbolic Exploration Algorithm

input : $\mathcal{T} = (E, \lambda, \preceq), \mathcal{M} = (M, m^0, B, \rightarrow_m)$
output: reachable$_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$
begin
  $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
  while $W \neq \emptyset$ **do**
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$
    $T \leftarrow T \cup (t, w, m)$
    repeat
      $x \leftarrow w$
      $w \leftarrow w \cup \{e \in \text{enabled}(w) \mid e \notin \text{sensitive}(m)\}$
    until $(w = x)$
    if $(w = E) \wedge (m \in B)$ **then**
      return false
    forall $(t', w', m') : (t, w, m) \rightarrow_s (t', w', m') \wedge (t', w', m') \notin T$ **do**
      $W \leftarrow W \cup \{(t', w', m')\}$
  return true
end

1. pick a symbolic configuration
2. extend it with non-sensitive events
3. check if a bad state is reached

In practice

Symbolic Exploration Algorithm

input : $\mathcal{T} = (E, \lambda, \preceq), \mathcal{M} = (M, m^0, B, \rightarrow_m)$
output: $\text{reachable}_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$
begin
  $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
  while $W \neq \emptyset$ **do**
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$
    $T \leftarrow T \cup (t, w, m)$
    repeat
      $x \leftarrow w$
      $w \leftarrow w \cup \{e \in \text{enabled}(w) \mid e \notin \text{sensitive}(m)\}$
    until $(w = x)$
    if $(w = E) \wedge (m \in B)$ **then**
      return false
    forall $(t', w', m') : (t, w, m) \xrightarrow{e}_s (t', w', m') \wedge (t', w', m') \notin T$ **do**
      $W \leftarrow W \cup \{(t', w', m')\}$
  return true
end

1. pick a symbolic configuration
2. extend it with non-sensitive events
3. check if a bad state is reached
4. fire symbolically sensitive events

Does it work?

Early results

Experiment				Explicit			Symbolic			*Spin*
Model	Processes	Events	Property	Error	Time	Conf.	Error	Time	Conf.	Time
Peterson	2	10000	Mutex	NO	1.39s	21551	NO	0.35s	4001	0.06s
	2	100000	Mutex	NO	16.88s	215544	NO	3.45s	40001	0.06s
	2	1000000	Mutex	-	-	-	NO	34.75s	400002	0.06s
Peterson	2	10000	Mutex	YES	1.11s	21384	YES	0.01s	4	0.05s
Faulty	2	100000	Mutex	YES	15.95s	214727	YES	0.05s	4	0.05s
	2	1000000	Mutex	-	-	-	YES	0.53s	4	0.05s
ABProtocol	2	10000	Received	NO	2.17s	31185	NO	0.42s	4654	0.15s
	2	100000	Received	NO	31.08s	316414	NO	4.25s	46684	0.15s
	2	1000000	Received	-	-	-	NO	43.09s	466887	0.15s
ABProtocol	2	10000	Received	YES	2.06s	31495	YES	0.01s	5	0.13s
Faulty	2	100000	Received	YES	29.70s	315808	YES	0.06s	5	0.13s
	2	1000000	Received	-	-	-	YES	0.53s	5	0.13s
Philosopher	3	100	Fork	NO	1.03s	6190	NO	0.05s	299	0.40s
	5	100	Fork	NO	87.02s	60727	NO	0.21s	2875	12.01s
	10	100	Fork	-	-	-	NO	1.52s	26791	-
Philosopher	3	100	Fork	YES	0.09s	1187	YES	0.01s	63	0.38s
Faulty	5	100	Fork	YES	78.72s	55982	YES	0.01s	78	11.01s
	10	100	Fork	-	-	-	YES	0.01s	55	-

Conclusion

What has been achieved?

- we proposed a symbolic method to monitor distributed systems
- has been implemented and works well in practice
- a tool, TraX (Trace eXplorer) is available:

`http://www.ulb.ac.be/di/ssd/cmeuter/trax`

Conclusion

What has been achieved?

- we proposed a symbolic method to monitor distributed systems
- has been implemented and works well in practice
- a tool, TraX (Trace eXplorer) is available:

  `http://www.ulb.ac.be/di/ssd/cmeuter/trax`

Future Work

- integrate the method into our distributed controler development environment dSL [DGMM05]
- apply the method to similar models, such as MSC
- extend the method to do model-checking, using McMillan's unfoldings