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
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, ...)
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
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
- separate 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]
Global predicate detection

Several classes of predicate have been studied:

- **Stable** predicates [CL85] (once true, remains true)
- **Disjunctive** predicates (of the form \( lp_1 \lor lp_2 \lor \cdots \lor lp_n \))
- **Conjunctive** predicates [GW94, GW96] (of the form \( lp_1 \land lp_2 \land \cdots \land lp_n \))
- **Observer independent** predicates [CDF95] (such that \( \forall \diamond \phi \iff \exists \diamond \phi \))
- **Linear, semi-linear** [CG98] or **regular** predicates (RCTL logic) [GM01, SG03]
Global predicate detection

Several classes of predicate have been studied:

- **Stable** predicates [CL85] (once true, remains true)
- **Disjunctive** predicates (of the form \( l_p_1 \lor l_p_2 \lor \cdots \lor l_p_n \))
- **Conjunctive** predicates [GW94, GW96] (of the form \( l_p_1 \land l_p_2 \land \cdots \land l_p_n \))
- **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]

\( \implies \neq \) our approach
Traces

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

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th></th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x:=0</td>
<td></td>
<td>w:=4</td>
</tr>
<tr>
<td></td>
<td>y:=3</td>
<td></td>
<td>w:=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x:=5</td>
<td></td>
</tr>
</tbody>
</table>

Monitor

non-deterministic finite automaton with guards on transitions built from an LTL formula using tableau construction. Loop transitions are omitted, and the monitor never blocks. Each guard is a boolean formula on one variable. Some states are marked as bad.
**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 omitted 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?
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

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x:=0</td>
<td>w:=4</td>
</tr>
<tr>
<td></td>
<td>y:=3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x:=5</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{init} \xrightarrow{x > 0} \text{tmp} \xrightarrow{w = 0} \text{bad} \]
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

<table>
<thead>
<tr>
<th></th>
<th>x:=0</th>
<th>y:=3</th>
<th>x:=5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P2</strong></td>
<td>w:=4</td>
<td></td>
<td>w:=0</td>
</tr>
</tbody>
</table>

Diagram:

1. **init**
2. **tmp**
3. **w = 0**
4. **bad**

- $x > 0$
- $w = 0$
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

P1
\[
\begin{array}{c}
\text{x:=0} \\
\text{y:=3} \\
\text{x:=5}
\end{array}
\]

P2
\[
\begin{array}{c}
\text{w:=4} \\
\text{w:=0}
\end{array}
\]

x:=0

init \xrightarrow{x>0} \text{tmp} \xrightarrow{w=0} \text{bad}
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

<table>
<thead>
<tr>
<th>P1</th>
<th>x:=0</th>
<th>y:=3</th>
<th>x:=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>w:=4</td>
<td></td>
<td>w:=0</td>
</tr>
</tbody>
</table>

\[ x:=0; w:=4 \]

\[ \text{init} \xrightarrow{x > 0} \text{tmp} \xrightarrow{w = 0} \text{bad} \]
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?

Example

```
\begin{align*}
\text{P1} & \quad x:=0 \quad y:=3 \quad x:=5 \\
\text{P2} & \quad w:=4 \quad w:=0
\end{align*}
```

\[
x:=0; \ w:=4; \ y:=3
\]
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

P1: x:=0 → y:=3 → x:=5
P2: w:=4 → w:=0

init → x>0 → tmp → w=0 → bad

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

\[
\begin{array}{c|c}
\text{P1} & x := 0 \rightarrow y := 3 \rightarrow x := 5 \\
\hline
\text{P2} & w := 4 \rightarrow w := 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{init} \xrightarrow{x > 0} \text{tmp} \xrightarrow{w = 0} \text{bad}
\end{array}
\]

x := 0; w := 4; y := 3; x := 5; w := 0
A simple solution

Compose the trace with the monitor to explore all possibilities:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>x:=0</td>
<td>y:=3</td>
</tr>
<tr>
<td>P2</td>
<td>w:=4</td>
<td>w:=0</td>
</tr>
</tbody>
</table>

Configuration = Cut + Monitor State

Problem:
exponential number of interleavings/configurations!

⇒ Can we do better?

A. Genon, T. Massart, C. Meuter
Monitoring Distributed Controllers
A simple solution

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

<table>
<thead>
<tr>
<th>P1</th>
<th>x:=0</th>
<th>y:=3</th>
<th>x:=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>w:=4</td>
<td>w:=0</td>
<td></td>
</tr>
</tbody>
</table>

**Configuration** = Cut + Monitor State

**Problem:** exponential number of interleavings/configurations!

⇒ Can we do better?

A. Genon, T. Massart, C. Meuter

Monitoring Distributed Controllers
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:

  ![Diagram](image)

<table>
<thead>
<tr>
<th>P1</th>
<th>x:=0 → y:=3 → x:=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>w:=4 → w:=0 → Cut</td>
</tr>
</tbody>
</table>

- **Configuration** = Cut + Monitor State
- **Problem**: exponential number of interleavings/configurations!
  \[\Rightarrow\] Can we do better?
The trace monitoring problem is **NP-Complete**.
The trace monitoring problem is **NP-Complete**

**NP-easiness**

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
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 $\phi$
- use the trace to encode all possible valuations
- $\exists$ a valuation satisfying $\phi$ iff the bad state is reachable
The monitor is not always **sensitive** to all events:

- **P1**
  - ... → $x := 3$ → ...

- **P2**
  - ... → $z := 5$ → ...

- **m**
  - $x < 6$
  - $w < 5$
Observation

- The monitor is not always sensitive to all events:

\[\begin{align*}
\text{P1} & \quad \ldots \quad x:=3 \quad \ldots \\
\text{P2} & \quad \ldots \quad z:=5 \quad \ldots \\
\text{m} & \quad x < 6 \\
& \quad w < 5
\end{align*}\]
Sensitive events

Observation

- The monitor is not always **sensitive** to all events:

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

  ![Diagram]

  - Node `m`:
    - `x < 6`
    - `w < 5`

  - Edges:
    - `P1`: `... x:=3 ...`
    - `P2`: `... z:=5 ...`
Observation

- The monitor is not always **sensitive** to all events:

  \[\begin{align*}
  \text{P1} & \quad \ldots \quad \xcol{3}{\text{x:=3}} \quad \ldots \\
  \text{P2} & \quad \ldots \quad \xcol{5}{\text{z:=5}} \quad \ldots
  \end{align*}\]

- 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**
  \[\Rightarrow \text{explore only one arbitrary interleaving}\]

- **But**: those events might be **interesting** in another state of the monitor
  \[\Rightarrow \text{keep them separate during exploration}\]
First all **non-sensitive** events are fired in an arbitrary order.

- **P1**: $x:=0 \rightarrow y:=3 \rightarrow x:=5$
- **P2**: $w:=4 \rightarrow w:=0$

The diagram shows:
- **Init** state: $x > 0$
- **Tmp** state: $w = 0$
- **Bad** state

Sensitive events:
- **Init**
- **Tmp**

Experimental result

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

- First all non-sensitive events are fired in an arbitrary order.
- Both the origin cut and the final cut are kept.
- Then, sensitive events are fired on all the cuts in between, in one step.
- Some previously non-sensitive events become sensitive $\Rightarrow$ they must be explored!

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

\[ P_1 \]

\[ P_2 \]

- \[ init \] $x > 0$
- \[ tmp \] $w = 0$
- \[ bad \]
Symbolic Exploration

Idea

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

- First all non-sensitive events are fired in an arbitrary order
- Both the origin cut and the final cut are kept
- Then, sensitive events are fired on all the cuts in between, in one step
- Some previously non-sensitive events become sensitive =⇒ they must be explored!
Idea

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

\[
\begin{align*}
\text{P1} & \quad x:=0 \quad y:=3 \quad x:=5 \\
\text{P2} & \quad w:=4 \quad w:=0 \\
& \quad [1,0] \rightarrow [2,2]
\end{align*}
\]
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**

\[
\begin{align*}
P1 & : x:=0 \rightarrow y:=3 \rightarrow x:=5 \\
P2 & : w:=4 \rightarrow w:=0 \\
\end{align*}
\]

\[
\begin{align*}
[1,0] & \rightarrow [2,2] \\
\end{align*}
\]
**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**.

![Diagram showing the symbolic exploration process](image)
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
   \[\Rightarrow\] they must be explored!

\[x := 0\]
\[y := 3\]
\[x := 5\]
\[w := 0\]
\[w := 4\]

\[\begin{align*}
P1 &\quad x := 0 &\quad y := 3 &\quad x := 5 \\
P2 &\quad w := 4 &\quad w := 0
\end{align*}\]

\[\begin{align*}
[3,0] &\quad \rightarrow &\quad [3,2]
\end{align*}\]
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**

⇒ they **must** be explored!

P1

\[ x := 0 \quad y := 3 \quad x := 5 \]

\[ \text{Bad} \quad \text{tmp} \quad w = 0 \quad \text{bad} \]

\[ [3, 0] \rightarrow [3, 2] \]
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

⇒ they must be explored!
Symbolic Composition

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

- **P1**
  - x:=0 → y:=3 → x:=5

- **P2**
  - w:=4 → w:=0

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!
Symbolic Composition

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

```
P1  x:=0  y:=3  x:=5
P2  w:=4  w:=0
```

Symbolic Configuration = Origin Cut + Final Cut + Monitor State

One symbolic configuration represents a set of configurations:

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

Symbolic composition is both sound and complete!
Symbolic Composition

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

\[
\begin{align*}
\text{P1} & : x := 0 \rightarrow y := 3 \rightarrow x := 5 \\
\text{P2} & : w := 4 \rightarrow w := 0
\end{align*}
\]

\[
\begin{align*}
\text{init} & \quad x > 0 \\
\text{tmp} & \quad w = 0 \rightarrow \text{bad}
\end{align*}
\]

\[
\begin{align*}
((0,0),[0,1],\text{init}) & \rightarrow ((1,0),[2,2],\text{init}) \quad x := 5 \\
((0,0),[0,1],\text{init}) & \rightarrow ((1,0),[2,2],\text{init}) \quad w := 4 \quad \Rightarrow ((3,0),[3,2],\text{init}) \quad w := 0
\end{align*}
\]

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

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

  ![Symbolic Composition Diagram](image)

- 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 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!
```
**Algorithm 1: Symbolic exploration**

input : $T = (E, \lambda, \preceq), M = (M, m^0, B, \rightarrow m)$

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

$x \leftarrow w$

$w \leftarrow w \cup \{ e \in \text{enabled}(w) \mid e \not\in \text{sensitive}(m) \}$

until $(w = x)$

if $(w = E) \land (m \in B)$ then

return false

forall $(t', w', m') : (t, w, m) \xrightarrow{e} (t', w', m') \land (t', w', m') \not\in T$ do

$W \leftarrow W \cup \{ (t', w', m') \}$

return true

end

In practice
Symbolic Exploration Algorithm

input : $T = (E, \lambda, \preceq), M = (M, m_0, B, \rightarrow m)$
output: $\text{reachable}_s(\emptyset, \emptyset, m_0) \cap B \neq \emptyset$
begin
    $T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m_0)\}$
    while ($W \neq \emptyset$)
        let $(t, w, m) \in W$
        $W \leftarrow W \setminus \{(t, w, m)\}$
        $T \leftarrow T \cup (t, w, m)$
        $x \leftarrow w$
        $w \leftarrow w \cup \{e \in \text{enabled}(w) \mid e \not\in \text{sensitive}(m)\}$
        until ($w = x$)
        if $(w = E) \land (m \in B)$ then
            return false
        for all $(t', w', m') : (t, w, m) \xrightarrow{e} (t', w', m') \land (t', w', m') \not\in T$
            $W \leftarrow W \cup \{(t', w', m')\}$
        end
return true
end

pick a symbolic configuration
Algorithm 1: Symbolic exploration

input: $T = (E, \lambda, \preceq), M = (M, m^0, B, \rightarrow m)$
output: $\text{reachable}_s(\emptyset, \emptyset, m^0) \cap B \neq \emptyset$

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

begin
$T \leftarrow \emptyset$, $W \leftarrow \{(\emptyset, \emptyset, m^0)\}$
while $W \neq \emptyset$ do
    let $(t, w, m) \in W$
    $W \leftarrow W \setminus \{(t, w, m)\}$
    $W \leftarrow \text{reachable}_s(\emptyset, \emptyset, m^0) \cap B$
    repeat
        $x \leftarrow w$
        $w \leftarrow w \cup \{e \in \text{enabled}(w) \mid e \notin \text{sensitive}(m)\}$
    until ($w = x$)
end
return false

forall $(t', w', m') : (t, w, m) \xrightarrow{s} (t', w', m') \land (t', w', m') \notin T$ do
    $W \leftarrow W \cup \{(t', w', m')\}$
end
return true
Symbolic Exploration Algorithm

**Algorithm 1: Symbolic exploration**

**Input:** \( T = (E, \lambda, \preceq) \), \( M = (M, m_0, B, \rightarrow_m) \)

**Output:** \( \text{reachable}_s(\emptyset, \emptyset, m_0) \cap B \neq \emptyset \)

1. **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** \( (w = E) \)
   - **if** \( (w = E) \land (m \in B) \) **then**
     - **return** false
   - **for all** \( (t', w', m') : (t, w, m) \rightarrow_s (t', w', m') \land (t', w', m') \notin T \) **do**
     - \( W \leftarrow W \cup \{(t', w', m')\} \)
   - **return** true

2. **pick a symbolic configuration**
3. **extend it with non-sensitive events**
4. **check if a bad state is reached**
Symbolic Exploration Algorithm

**input**: T = (E, λ, ⪯), M = (M, m₀, B, → m)
**output**: reachable_s(∅, ∅, m₀) ∩ B ̸= ∅

```
begin
    T ← ∅, W ← {(∅, ∅, m₀)}
while W ≠ ∅ do
    let (t, w, m) ∈ W
    W ← W \ {(t, w, m)}
    T ← T ∪ (t, w, m)
    repeat
        x ← w
        w ← w ∪ {e ∈ enabled(w) | e ∉ sensitive(m)}
    until (w = x)
    if (w = E) ∧ (m ∈ B) then
        return false
    forall (t', w', m') : (t, w, m) → s (t', w', m') ∧ (t', w', m') ∉ T do
        W ← W ∪ {(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
### Early results

<table>
<thead>
<tr>
<th>Model</th>
<th>Processes</th>
<th>Events</th>
<th>Property</th>
<th>Explicit</th>
<th>Symbolic</th>
<th>Spin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Error</td>
<td>Time</td>
<td>Conf.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
<td>Conf.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>10000</td>
<td>Mutex</td>
<td>NO</td>
<td>1.39s</td>
<td>21551</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100000</td>
<td>Mutex</td>
<td>NO</td>
<td>16.88s</td>
<td>215544</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1000000</td>
<td>Mutex</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>10000</td>
<td>Mutex</td>
<td>YES</td>
<td>1.11s</td>
<td>21384</td>
</tr>
<tr>
<td>Faulty</td>
<td>2</td>
<td>10000</td>
<td>Mutex</td>
<td>YES</td>
<td>15.95s</td>
<td>214727</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100000</td>
<td>Mutex</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ABProtocol</td>
<td>2</td>
<td>10000</td>
<td>Received</td>
<td>NO</td>
<td>2.17s</td>
<td>31185</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100000</td>
<td>Received</td>
<td>NO</td>
<td>31.08s</td>
<td>316414</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1000000</td>
<td>Received</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ABProtocol</td>
<td>2</td>
<td>10000</td>
<td>Received</td>
<td>YES</td>
<td>2.06s</td>
<td>31495</td>
</tr>
<tr>
<td>Faulty</td>
<td>2</td>
<td>10000</td>
<td>Received</td>
<td>YES</td>
<td>29.70s</td>
<td>315808</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100000</td>
<td>Received</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Philosopher</td>
<td>3</td>
<td>100</td>
<td>Fork</td>
<td>NO</td>
<td>1.03s</td>
<td>6190</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>Fork</td>
<td>NO</td>
<td>87.02s</td>
<td>60727</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100</td>
<td>Fork</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Philosopher</td>
<td>3</td>
<td>100</td>
<td>Fork</td>
<td>YES</td>
<td>0.09s</td>
<td>1187</td>
</tr>
<tr>
<td>Faulty</td>
<td>5</td>
<td>100</td>
<td>Fork</td>
<td>YES</td>
<td>78.72s</td>
<td>55982</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100</td>
<td>Fork</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Does it work?
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/ssf/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 controller development environment **dSL** [DGMM05]
- apply the method to similar models, such as **MSC**
- extend the method to do **model-checking**, using McMillan’s unfoldings