A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

Gilles Geeraerts

gigeerae@ulb.ac.be.

Université Libre de Bruxelles – Département d’informatique
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

What is a parametrized concurrent system?
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

What is a parametrized concurrent system?
- Need for verification, how to formalize...
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a parametrized concurrent system?
  - Need for verification, how to formalize...

- How do we verify?
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

What is a parametrized concurrent system?
- Need for verification, how to formalize...

How do we verify?
- Forward and backward approach, decidability results...
Plan of the talk

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a parametrized concurrent system?
  - Need for verification, how to formalize...

- How do we verify?
  - Forward and backward approach, decidability results...

- What can we compare?
Plan of the talk

**A Comparison** of Various Backward Analyzers for Parametrized Concurrent Systems

- What is a *parametrized concurrent system*?
  - Need for verification, how to formalize.

- How do we **verify**?
  - Forward and backward approach, decidability results.

- What can we **compare**?
  - Performances with different datastructures.
Motivation – Concurrent systems

Concurrent system = system with many processes interacting and communicating...
Motivation – Concurrent systems

Concurrent system = system with many processes interacting and communicating...

... they can be found everywhere!
Motivation – Concurrent systems

Concurrent system = system with many processes interacting and communicating...

... they can be found everywhere!

e.g.: Multi-threaded Java programs for web-based applications.
Motivation – Concurrent systems

Concurrent system = system with many processes interacting and communicating...

...they can be found everywhere!

e.g.: Multi-threaded Java programs for web-based applications.

They often get involved in safety-critical environments.
Motivation – Concurrent systems

Concurrent system = system with many processes interacting and communicating...

...they can be found everywhere!
  e.g.: Multi-threaded Java programs for web-based applications.

They often get involved in safety-critical environments.
  e.g.: Online secured billing.
Motivation – Concurrent systems

- Concurrent system = system with many processes interacting and communicating...

- ...they can be found everywhere!
  - e.g.: Multi-threaded Java programs for web-based applications.

- They often get involved in safety-critical environments.
  - e.g.: Online secured billing.

We need well-suited verification procedures!
Motivation – Parametrized verification

Most of the time, the number of process is not fixed (it is a parameter).
Motivation – Parametrized verification

Most of the time, the number of process is not fixed (it is a parameter).

- e.g.: How many clients are going to connect to a given web-server?
Motivation – Parametrized verification

Most of the time, the number of process is not fixed (it is a parameter).

  e.g.: How many clients are going to connect to a given web-server?

Classical approach: try for one process, two processes, three processes...
Motivation – Parametrized verification

Most of the time, the number of process is not fixed (it is a parameter).

  e.g.: How many clients are going to connect to a given web-server?

**Classical approach:** try for one process, two processes, three processes... 

...and hope the property holds for other values of the parameter!
Most of the time, the number of process is not fixed (it is a parameter).

- e.g.: How many clients are going to connect to a given web-server?

**Classical approach:** try for one process, two processes, three processes...

...and hope the property holds for other values of the parameter!

**Parametrized approach:** Verify the property for any value of the parameter.
The verification process

Concurrent system

[Diagram showing the verification process with three main steps]
The verification process

Concurrent system

Global/Local Machine

Formalization
The verification process

Concurrent system

Global/Local Machine

Multi-Transfer Net

Formalization

Translation
Global/Local machines

One global machine = collection of several local machines + global boolean variables.
Global/Local machines

- One global machine = collection of several local machines + global boolean variables.
- The local machines synchronize through rendez-vous, broadcasts and asynchronous rendez-vous.
Global/Local machines

- One **global machine** = collection of several **local machines** + global **boolean variables**.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.

```
(a) b ← true      (b) l!! : b = true
    \ arrow \       \ arrow \\
    \   \        \   \    \\
  (d) l??       (e) \\
    \   \        \   \    \\
  (f) l??       (g)
```
Global/Local machines

- One global machine = collection of several local machines + global boolean variables.
- The local machines synchronize through rendez-vous, broadcasts and asynchronous rendez-vous.

\[
\begin{align*}
  & a \quad b \leftarrow \text{true} \quad b \text{!!} : b = \text{true} \\
  & d \quad l?? \\
  & f \quad l??
\end{align*}
\]
Global/Local machines

- One **global machine** = collection of several **local machines** + global boolean variables.
- The local machines **synchronize** through *rendez-vous*, broadcasts and asynchronous *rendez-vous*.

```
\begin{tikzpicture}
  \node (a) at (0,0) {$a \quad b \leftarrow \text{true}$};
  \node (b) at (2,0) {$b \quad l!! : b = \text{true}$};
  \node (c) at (4,0) {$c$};
  \node (d) at (0,-2) {$d \quad l??$};
  \node (e) at (2,-2) {$e$};
  \node (f) at (0,-4) {$f \quad l??$};
  \node (g) at (2,-4) {$g$};
  \draw[->] (a) -- (b);
  \draw[->] (b) -- (c);
  \draw[->] (d) -- (e);
  \draw[->] (f) -- (g);
\end{tikzpicture}
```
Multi-transfer nets
Multi-transfer nets

\[ T_1 \]

1

2
Multi-transfer nets
Forward v.s. Backward

Verification of a safety property → is a MTN marking reachable?
Forward v.s. Backward

- Verification of a safety property $\rightarrow$ is a MTN marking reachable?
- Two approaches:
Forward v.s. Backward

Verification of a safety property $\rightarrow$ is a MTN marking reachable?

Two approaches:

Forward
Forward v.s. Backward

Verification of a safety property $\rightarrow$ is a MTN marking reachable?

Two approaches:

Forward
Forward v.s. Backward

- Verification of a safety property is a MTN marking reachable?
- Two approaches:

Forward
Forward v.s. Backward

Verification of a safety property $\rightarrow$ is a MTN marking reachable?

Two approaches:

- Forward
Forward v.s. Backward

- Verification of a safety property $\rightarrow$ is a MTN marking reachable?
- Two approaches:

Forward
Forward v.s. Backward

Verification of a safety property $\rightarrow$ is a MTN marking reachable?

Two approaches:

Forward
Forward v.s. Backward

- Verification of a safety property \( \rightarrow \) is a MTN marking reachable?
- Two approaches:
Forward v.s. Backward

- Verification of a safety property → is a MTN marking reachable?
- Two approaches:

Forward

Backward
Forward v.s. Backward

Verification of a safety property $\rightarrow$ is a MTN marking reachable?

Two approaches:

Forward

Backward
Forward v.s. Backward

Verification of a safety property → is a MTN marking reachable?

Two approaches:

Forward

Backward
Forward v.s. Backward

Verification of a safety property → is a MTN marking reachable?

Two approaches:

Forward

Backward
Forward v.s. Backward

Verification of a safety property → is a MTN marking reachable?

Two approaches:
Decidability

- The fixed-point algorithm working backwards will finish if the set of unsafe points is upward-closed [Abdulla, Cerans, ...].

- An upward-closed set of points (markings) is characterized by its generator.
To store the set of reachable markings, we need efficient datastructures.
To store the set of reachable markings, we need efficient datastructures.

Which one is best-suited?
Datastructures

To store the set of reachable markings, we need efficient datastructures.

Which one is best-suited?

Let’s compare the practical performances of four of them: CST, IST, DDD, NDD!
Covering Sharing Trees

\[(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 0 \land p_5 \geq 2)\]
\[\lor\]
\[(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 1 \land p_5 \geq 1)\]
\[\lor\]
\[(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 2 \land p_5 \geq 0)\]
Covering Sharing Trees

\[(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 0 \land p_5 \geq 2) \lor (p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 1 \land p_5 \geq 1) \lor (p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 2 \land p_5 \geq 0)\]
Covering Sharing Trees

\[
\begin{align*}
(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 0 \land p_5 \geq 2) \\
\lor
\begin{align*}
(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 1 \land p_5 \geq 1) \\
\lor
\begin{align*}
(p_1 \geq 0 \land p_2 \geq 0 \land p_3 \geq 0 \land p_4 \geq 2 \land p_5 \geq 0)
\end{align*}
\end{align*}
\]
Interval Sharing Trees

\[ \frac{1 \leq m_1 \leq 2}{\frac{5 \leq m_2 \leq 7}{\frac{4 \leq m_3 \leq 5}{\frac{2 \leq m_4 \leq 4}{\frac{1 \leq m_1 \leq 2}{\frac{3 \leq m_2 \leq 4}{\frac{7 \leq m_3 \leq 9}{\frac{2 \leq m_4 \leq 4}{\frac{1 \leq m_1 \leq 2}{\frac{3 \leq m_2 \leq 4}{\frac{1 \leq m_3 \leq 2}{\frac{2 \leq m_4 \leq 4}}}}}}}}}}}} \]
Interval Sharing Trees

\[(1 \leq m_1 \leq 2) \land (5 \leq m_2 \leq 7) \land (4 \leq m_3 \leq 5) \land (2 \leq m_4 \leq 4)\]

\[\lor\]

\[(1 \leq m_1 \leq 2) \land (3 \leq m_2 \leq 4) \land (7 \leq m_3 \leq 9) \land (2 \leq m_4 \leq 4)\]

\[\lor\]

\[(1 \leq m_1 \leq 2) \land (3 \leq m_2 \leq 4) \land (1 \leq m_3 \leq 2) \land (2 \leq m_4 \leq 4)\]
Interval Sharing Trees

(1 \leq m_1 \leq 2) \land (5 \leq m_2 \leq 7) \land (4 \leq m_3 \leq 5) \land (2 \leq m_4 \leq 4)

\lor

(1 \leq m_1 \leq 2) \land (3 \leq m_2 \leq 4) \land (7 \leq m_3 \leq 9) \land (2 \leq m_4 \leq 4)

\lor

(1 \leq m_1 \leq 2) \land (3 \leq m_2 \leq 4) \land (1 \leq m_3 \leq 2) \land (2 \leq m_4 \leq 4)
\neg (m_1 < 1) \land (m_1 \leq 3) \land \neg (m_2 < 2) \\
\lor \\
\neg (m_1 < 1) \land (m_1 \leq 3) \land (m_2 < 2) \land \neg (m_2 - m_1 < 0)
Difference Decision Diagrams

\[ m_1 - Z < 1 \]
\[ m_1 - Z \leq 3 \]
\[ m_2 - Z < 2 \]
\[ m_2 - m_1 < 0 \]

\[ \neg (m_1 < 1) \land (m_1 \leq 3) \land \neg (m_2 < 2) \]
\[ \lor \]
\[ \neg (m_1 < 1) \land (m_1 \leq 3) \land (m_2 < 2) \land \neg (m_2 - m_1 < 0) \]
Number Decision Diagrams

\[ m_3 = m_1 + m_2 \]

1+1=2: 001+001=010

(simplified version)
Number Decision Diagrams

\[ m_3 = m_1 + m_2 \]

1+1=2: \( 001 + 001 = 010 \)

(simplified version)
Number Decision Diagrams

$m_3 = m_1 + m_2$

1+1=2: 001+001=010

(simplified version)
\[ m_3 = m_1 + m_2 \]

\[
\begin{align*}
1+1=2: & \quad 001+001=010 \\
000 & 011 101 \\
010 & 100 111 \\
i, j, i+j & \\
i, j, i+j-2 & \\
i, j, i+j+1 & \\
i, j, i+j-1 &
\end{align*}
\]
The comparison

First, we need a good set of examples:

- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:

- Execution time (User/System);
- Memory consumption (Resident/Data/Total/...);
- Bottleneck operations;
...
The comparison

First, we need a good set of examples:

- Bounded or unbounded Petri nets;
First, we need a good set of examples:

- Bounded or unbounded Petri nets;
- Cache coherency protocol;
The comparison

First, we need a good set of examples:

- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.
The comparison

First, we need a good set of examples:
- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:
First, we need a good set of examples:
- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:
- Execution time (User/System);
First, we need a good set of examples:
- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:
- Execution time (User/System);
- Memory consumption (Resident/Data/Total/...);
The comparison

First, we need a good set of examples:
- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:
- Execution time (User/System);
- Memory consumption (Resident/Data/Total/...);
- Bottleneck operations;
The comparison

First, we need a good set of examples:
- Bounded or unbounded Petri nets;
- Cache coherency protocol;
- Multi-threaded Java programs.

Then, select the set of parameters:
- Execution time (User/System);
- Memory consumption (Resident/Data/Total/…);
- Bottleneck operations;
  …
A twofold comparison – First phase

BABYLON: an unified model-checker.
A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as **constraint solvers**;
A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as *constraint solvers*;
- The three model-checking algorithms are *shared* among the datastructures;
A twofold comparison – First phase

**BABYLON**: an unified model-checker.

The datastructures are seen as **constraint solvers**;

The three model-checking algorithms are **shared** among the datastructures;

Only Petri nets.
A twofold comparison – First phase

- **BABYLON**: an unified model-checker.
- The datastructures are seen as constraint solvers;
- The three model-checking algorithms are shared among the datastructures;
- Only Petri nets.

```cpp
class Set {
    virtual Set * Union (const Set * S) = 0;
    virtual Set * Intersection (const Set * S) = 0 ;
    virtual Set * Difference (const Set * S) = 0 ;
    virtual bool IsEmpty() = 0 ;
    virtual Set * Pre(void) = 0 ; virtual Set * Pre(int i) = 0 ;
    virtual void EmptySet() = 0 ; […] }
```
## Results – Second Phase

### Execution times (sec.) – Algorithm 3

<table>
<thead>
<tr>
<th>Example</th>
<th>CST</th>
<th>IST</th>
<th>DDD</th>
<th>NDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peterson</td>
<td>0.54</td>
<td>0.34</td>
<td>0.33</td>
<td>2'172.19</td>
</tr>
<tr>
<td>Lamport</td>
<td>0.14</td>
<td>0.1</td>
<td>0.13</td>
<td>139.19</td>
</tr>
<tr>
<td>Multipool</td>
<td>14.19</td>
<td>9.36</td>
<td>3.04</td>
<td>&gt;3 hours</td>
</tr>
<tr>
<td>Mesh3x2</td>
<td>466.31</td>
<td>513.62</td>
<td>195.99</td>
<td>&gt;3 hours</td>
</tr>
</tbody>
</table>

(...)

A Comparison of Various Backward Analyzers for Parametrized Concurrent Systems – p.17/21
A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
- NDD part → LASH library (ULg);
A twofold comparison – Second phase

**YABA**: a model-checker based on DDD and NDD.

- NDD part → LASH library (ULg);
- DDD part → DDD library (Møller) + new extensions.
A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part $\rightarrow$ LASH library (ULg);
  - DDD part $\rightarrow$ DDD library (Møller) + new extensions.
- Other tools already exist for CST and IST.
A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.

- Other tools already exist for CST and IST.

- The algorithms are peculiar to the datastructures.
A twofold comparison – Second phase

**YABA**: a model-checker based on DDD and NDD.
- NDD part → LASH library (ULg);
- DDD part → DDD library (Møller) + new extensions.

Other tools already exist for CST and IST.

The algorithms are **peculiar** to the datastructures.

Many **optimizations (invariants)** have been used.
A twofold comparison – Second phase

**YABA**: a model-checker based on DDD and NDD.
- NDD part → LASH library (ULg);
- DDD part → DDD library (Møller) + new extensions.

Other tools already exist for CST and IST.

The algorithms are **peculiar** to the datastructures.

Many **optimizations (invariants)** have been used.

**New optimizations** techniques have been developed for DDD and NDD.
A twofold comparison – Second phase

- **YABA**: a model-checker based on DDD and NDD.
  - NDD part → LASH library (ULg);
  - DDD part → DDD library (Møller) + new extensions.

Other tools already exist for CST and IST.

The algorithms are **peculiar** to the datastructures.

Many **optimizations (invariants)** have been used.

**New optimizations** techniques have been developed for DDD and NDD.

Large set of examples.
# Results – Second Phase

## Execution times (sec.)

<table>
<thead>
<tr>
<th>Example</th>
<th>CST</th>
<th>IST</th>
<th>DDD</th>
<th>NDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peterson</td>
<td>0.88</td>
<td>0.2</td>
<td>0.31</td>
<td>691.12</td>
</tr>
<tr>
<td>Multipool</td>
<td>3.39</td>
<td>5.44</td>
<td>0.49</td>
<td>1'309.12</td>
</tr>
<tr>
<td>Client/Server</td>
<td>0.27</td>
<td>0.09</td>
<td>0.44</td>
<td>3.34</td>
</tr>
<tr>
<td>Client/Server (Ex I)</td>
<td>–</td>
<td>–</td>
<td>0.04</td>
<td>0.9</td>
</tr>
<tr>
<td>Client/Server (He I)</td>
<td>0.04</td>
<td>0</td>
<td>6.28</td>
<td>–</td>
</tr>
<tr>
<td>Illinois</td>
<td>–</td>
<td>0</td>
<td>0.04</td>
<td>0.66</td>
</tr>
</tbody>
</table>

(...
Conclusion

NDD are definitely too slow. (Next version ?)
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
  - DDD are quicker (more powerful implementation)…
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
  - DDD are quicker (more powerful implementation) . . .
  - . . . but have poor memory consumption.
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
  - DDD are quicker (more powerful implementation)...
  - ...but have poor memory consumption.
  - Their increased expressivity is not interesting here.
- IST and CST thus seem best-suited.
Conclusion

- NDD are definitely too slow. (Next version ?)
- CST, IST and DDD have more or less the same performances:
  - DDD are quicker (more powerful implementation) . . .
  - . . .but have poor memory consumption.
  - Their increased expressivity is not interesting here.
- IST and CST thus seem best-suited.
- These experiments could be largely refined.
Conclusion

NDD are definitely too slow. (Next version ?)

CST, IST and DDD have more or less the same performances:
- DDD are quicker (more powerful implementation) . . .
- . . . but have poor memory consumption.
- Their increased expressivity is not interesting here.

IST and CST thus seem best-suited.

These experiments could be largely refined.

Other datastructures ?
Personal contributions

- Development (with Giorgio) of the methodology.

- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library: for the invariant-based optimization; to let it handle transfers (MTN).
- Implementation of ABYLON (with Pierre and Laurent)
- Benchmarks and collection of the results.
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of $Y_{ABA}$.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of $Y_{ABA}$.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of $Y_{ABA}$.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).
- Implementation of BABYLON (with Pierre and Laurent)
Personal contributions

- Development (with Giorgio) of the methodology.
- Conception and implementation of YABA.
- Adaptation of the invariant-based optimization to NDD and DDD.
- Extension of the DDD library:
  - for the invariant-based optimization;
  - to let it handle transfers (MTN).
- Implementation of BABYLON (with Pierre and Laurent)
- Benchmarks and collection of the results.