On Verifying Causal Consistency

Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, Jad Hamza

IRIF, Université Paris Diderot

May 2017
Geo-Replicated Data Structures

- Strong (sequential) consistency

---

1S. Gilbert and N. A. Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services.
Geo-Replicated Data Structures

- Strong (sequential) consistency

---

1S. Gilbert and N. A. Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services.
Geo-Replicated Data Structures

- Strong (sequential) consistency

S. Gilbert and N. A. Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services.
Geo-Replicated Data Structures

- Strong (sequential) consistency is **impossible** while being **available** and tolerating **network partitions**: the CAP theorem

1S. Gilbert and N. A. Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services.
Geo-Replicated Data Structures

- Tolerating faults while preserving availability leads to anomalies w.r.t. strong (sequential) consistency
Geo-Replicated Data Structures

- Tolerating **faults** while preserving **availability** leads to **anomalies** w.r.t. strong (sequential) consistency

```
write(x, 2)  write(x, 1)
```

read(x) ▶ 1
read(x) ▶ 2
read(x) ▶ 1
read(x) ▶ 2
Geo-Replicated Data Structures

- Tolerating faults while preserving availability leads to anomalies w.r.t. strong (sequential) consistency

Updates are seen in different orders
Goal: Verifying Causal Consistency

The set of allowed anomalies are defined by \textit{weak consistency} criteria, e.g., eventual consistency, causal consistency.

Algorithmic methods for checking \textit{causal consistency}.

\textbf{Single-Trace Verification}: Check if one trace is causally consistent
- Application to testing, monitoring (by enumerating traces)

\textbf{All-Traces Verification}: Check if all traces are causally consistent
- Static verification
Comparison with other Consistency Criteria

Single-Trace Verification:

- **NP-complete** for most consistency criteria\(^2\)
Comparison with other Consistency Criteria

Single-Trace Verification:

- **NP-complete** for most consistency criteria
- **NP-complete** for causal consistency as well

---

6 Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.
Comparison with other Consistency Criteria

Single-Trace Verification:
- **NP-complete** for most consistency criteria\(^2\)
- **NP-complete** for causal consistency as well

All-Traces Verification:
- **EXPSPACE-complete** for linearizability\(^3,4\)

---

\(^3\) Memory Model-aware Testing. Furbach et al. 2014.
\(^4\) Model-Checking of Correctness Conditions. Alur et al. 1996.
\(^6\) Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.
Comparison with other Consistency Criteria

Single-Trace Verification:
- **NP-complete** for most consistency criteria\(^2\)
- **NP-complete** for causal consistency as well

All-Traces Verification:
- **EXPSPACE-complete** for linearizability\(^3,4\)
- **Undecidable** for sequential consistency\(^5,6\)

---

\(^3\)Memory Model-aware Testing. Furbach et al. 2014.
\(^4\)Model-Checking of Correctness Conditions. Alur et al. 1996.
\(^6\)Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.
Comparison with other Consistency Criteria

Single-Trace Verification:
- **NP-complete** for most consistency criteria\(^2\)
- **NP-complete** for causal consistency as well

All-Traces Verification:
- **EXPSPACE-complete** for linearizability\(^3,4\)
- **Undecidable** for sequential consistency\(^5,6\)
- **Decidable** for eventual consistency\(^7\)

---

\(^3\) Memory Model-aware Testing. Furbach et al. 2014.
\(^4\) Model-Checking of Correctness Conditions. Alur et al. 1996.
\(^6\) Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.
Comparison with other Consistency Criteria

Single-Trace Verification:
- **NP-complete** for most consistency criteria\(^2\)
- **NP-complete** for causal consistency as well

All-Traces Verification:
- **EXPSPACE-complete** for linearizability\(^3,4\)
- **Undecidable** for sequential consistency\(^5,6\)
- **Decidable** for eventual consistency\(^7\)
- **Undecidable** for causal consistency

---

\(^3\)Memory Model-aware Testing. Furbach et al. 2014.
\(^4\)Model-Checking of Correctness Conditions. Alur et al. 1996.
\(^6\)Verifying Eventual Consistency of ORS. Bouajjani et al. 2014.
What About Usual Data Structures?

**Key-value store** (read/write operations):
one of the **simplest** and most **widely used** data structures.
What About Usual Data Structures?

**Key-value store** (read/write operations):
one of the **simplest** and most **widely used** data structures.

---

**Theorem (All-Traces Verification)**

*Checking if all traces of an implementation are causally consistent is undecidable.*
What About Usual Data Structures?

**Key-value store** (read/write operations):
one of the **simplest** and most **widely used** data structures.

---

**Theorem (All-Traces Verification)**

Checking if all traces of an implementation are causally consistent is **undecidable**.

---

Even with the following restrictions:

- For **key-value stores**
- For a bounded number of **sites**
- For **finite-state** implementations
- For a bounded number of **variables**
- For a bounded variables’ **domain**
What About Usual Data Structures?

**Key-value store** (read/write operations): one of the **simplest** and most **widely used** data structures.

**Theorem (All-Traces Verification)**

Checking if all traces of an implementation are causally consistent is **undecidable**.

Even with the following restrictions:

- For **key-value stores**
- For a bounded number of **sites**
- For **finite-state** implementations
- For a bounded number of **variables**
- For a bounded variables’ **domain**

(Input: **finite-state automaton** representing all traces)
Key Observation: Implementations Are Data Independent

Key-value store implementations are data independent

The behaviors do not depend on the particular values stored in the KVS.
Key Observation: Implementations Are Data Independent

Key-value store implementations are **data independent**

The **behaviors** do not depend on the particular values stored in the KVS.

⇒ **Writes can be assumed to be unique**
Results: Causal Consistency Violations Using Bad Patterns

**Bad Pattern**: A set of operations *related* in a particular way.
Results: Causal Consistency Violations Using Bad Patterns

**Bad Pattern:** A set of operations *related* in a particular way

Identify a set of bad patterns $X$ such that:

**Theorem (Bad Patterns)**

*A trace is not causally consistent iff it contains some bad pattern from $X*
Results: Causal Consistency Violations Using Bad Patterns

**Bad Pattern**: A set of operations related in a particular way

Identify a set of bad patterns $X$ such that:

**Theorem (Bad Patterns)**

*A trace is not causally consistent iff it contains some bad pattern from $X*

$X$ contains 4-6 bad patterns
Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for data-independent implementations:

**Theorem (Single-Trace Verification)**

*Singe-Trace Verification of causal consistency is polynomial when writes are unique.*
Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for data-independent implementations:

Theorem (Single-Trace Verification)

Single-Trace Verification of causal consistency is polynomial when writes are unique.

Theorem (Reduction to Reachability)

All-Traces Verification can be reduced to reachability or invariant checking. (by building a monitor (state machine) $M$ that tracks bad patterns)
Results: Complexity/Decidability and Reduction to Reachability

Bad patterns implications for data-independent implementations:

Theorem (Single-Trace Verification)

Single-Trace Verification of causal consistency is polynomial when writes are unique.

Theorem (Reduction to Reachability)

All-Traces Verification can be reduced to reachability or invariant checking. (by building a monitor (state machine) $M$ that tracks bad patterns)

Theorem (All-Traces Verification)

Checking whether all traces of a data-independent finite-state implementation are causally consistent is decidable.
Definition(s) of *causal consistency*
Outline

- Definition(s) of **causal consistency**
- Characterize **all causal consistency violations** using bad patterns
Outline

- Definition(s) of causal consistency
- Characterize all causal consistency violations using bad patterns
- Using bad patterns for verifying data-independent implementations
  - Single-Trace Verification: polynomial time
  - Bad patterns can be recognized with state machines
  - Generic reduction from causal consistency to reachability
  - All-Traces Verification: decidable
Outline

- Definition(s) of **causal consistency**
- Characterize all **causal consistency violations** using bad patterns
- Using bad patterns for verifying data-independent implementations
  - Single-Trace Verification: **polynomial time**
  - **Bad patterns** can be recognized with **state machines**
  - **Generic reduction** from causal consistency to **reachability**
  - All-Traces Verification: **decidable**
Definition of Causal Consistency

There exists a causality order $CO$ such that the causal past of every read can explain its value $CO$ includes the program (site) order.
Definition of Causal Consistency

There exists a *causality order* \( CO \) such that

- the causal past of every *read* can explain its value
- \( CO \) includes the program (site) order
Definition of Causal Consistency

There exists a causality order $CO$ such that
the causal past of every read can explain its value

$CO$ includes the program (site) order
Definition of Causal Consistency

There exists a causality order $CO$ such that

the causal past of every read can explain its value

$CO$ includes the program (site) order
Definition of Causal Consistency

There exists a causality order \( CO \) such that

the causal past of every read can explain its value

\( CO \) includes the program (site) order
Definition of Causal Consistency

There exists a **causality order** $CO$ such that

the **causal past of every read** can explain its value

$CO$ includes the program (site) order
There exists a **causality order** $CO$ such that

the **causal past of every read** can explain its value

$CO$ includes the program (site) order
There exists a **causality order** \( CO \) such that

the causal past of every read can explain its value

\( CO \) includes the program (site) order
There exists a *causality order* $CO$ such that

the causal past of every read can explain its value

$CO$ includes the program (site) order
There exists a **causality order** $CO$ such that

the **causal past of every read** can explain its value

$CO$ includes the program (site) order
There exists a **causality order** $CO$ such that

the causal past of every read can explain its value

$CO$ includes the program (site) order
Causally related writes must be seen by all sites in the same order.
Causally related writes must be seen by all sites in the same order.
Causally related \textit{writes} must be seen by all sites in the same order.

\begin{itemize}
\item \texttt{write}(x, 1)
\item \texttt{write}(x, 2) \hspace{1cm} \texttt{write}(y, 3)
\item \texttt{read}(x) \triangleright 1
\item \texttt{read}(y) \triangleright 3 \hspace{1cm} \texttt{read}(x) \triangleright 1
\end{itemize}
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

- \(\text{write}(x, 1) \cdot \text{write}(y, 2) \cdot \text{read}(x) \triangleright 1 \cdot \text{read}(y) \triangleright 2\)
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

- \(\text{write}(x, 1) \cdot \text{write}(y, 2) \cdot \text{read}(x) \triangleright 1 \cdot \text{read}(y) \triangleright 2\)

A history \(h = (O, PO)\) is **causally consistent** w.r.t. a **specification** \(S\) iff there exists a strict **partial order** \(CO\) s.t.
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

- \texttt{write}(x, 1) \cdot \texttt{write}(y, 2) \cdot \texttt{read}(x) \triangleright 1 \cdot \texttt{read}(y) \triangleright 2

A history \( h = (O, PO) \) is **causally consistent** w.r.t. a specification \( S \) iff there exists a strict partial order \( CO \) s.t.

\[
\text{AxCausal} : \ PO \subseteq CO
\]
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

\[ \text{write}(x, 1) \cdot \text{write}(y, 2) \cdot \text{read}(x) \triangleright 1 \cdot \text{read}(y) \triangleright 2 \]

A history \( h = (O, PO) \) is **causally consistent** w.r.t. a specification \( S \) iff there exists a strict partial order \( CO \) s.t.

\[
\begin{align*}
\text{AxCausal}: & \quad PO \subseteq CO \\
\text{AxCausalValue}: & \quad \forall o \in O. \text{CausalPast}(CO, o) \subseteq S
\end{align*}
\]
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

- `write(x, 1) \cdot write(y, 2) \cdot read(x) \blacktriangleright 1 \cdot read(y) \blacktriangleright 2`

A history \( h = (O, PO) \) is **causally consistent** w.r.t. a specification \( S \) iff there exists a strict partial order \( CO \) s.t.

**AxCausal** : \( PO \subseteq CO \)

**AxCausalValue** : \( \forall o \in O. \) CausalPast\((CO, o) \subseteq S \)

\((\text{CausalPast}(CO, o) = \text{the restriction of } CO \text{ to } CO^{-1}(o) \cup \{o\})\)
Formalizing Causal Consistency

**Specification** = a set of sequences of operations

\[ \text{write}(x, 1) \cdot \text{write}(y, 2) \cdot \text{read}(x) \triangleright 1 \cdot \text{read}(y) \triangleright 2 \]

A history \( h = (O, PO) \) is **causally consistent** w.r.t. a specification \( S \) iff there exists a strict partial order \( CO \) s.t.

\[
\text{AxCausal} : \quad PO \subseteq CO \\
\text{AxCausalValue} : \quad \forall o \in O. \ CausalPast(CO, o) \subseteq S
\]

\( (\text{CausalPast}(CO, o) = \text{the restriction of } CO \text{ to } CO^{-1}(o) \cup \{o\}) \)

\( \sqsubseteq \text{ means “can be linearized to”} \)
Causal Convergence\textsuperscript{8}

- Conflicts are resolved using a global \textit{arbitration order}
- \textbf{Strong eventual consistency:}  
  If two sites see the same writes, they are in the same state\textsuperscript{7}

\textsuperscript{7}A comprehensive study of CRDTs. 2011. Shapiro et al.
\textsuperscript{8}Understanding Eventual Consistency. Burckhardt et al. 2013.
Causal Convergence

- Conflicts are resolved using a global arbitration order
- **Strong eventual consistency**: If two sites see the same writes, they are in the same state

**Not allowed** by causal convergence:

7 A comprehensive study of CRDTs. 2011. Shapiro et al.
A history $h = (O, PO)$ is **causally convergent** w.r.t. a specification $S$ iff there exists a strict partial order $CO$ and a strict total order $ARB$ (arbitration) s.t.
Causal Convergence

A history \( h = (O, PO) \) is **causally convergent** w.r.t. a specification \( S \) iff there exists a strict partial order \( CO \) and a strict total order \( ARB \) (arbitration) s.t.

\[
\begin{align*}
\text{AxCausal} : & \quad PO \subseteq CO \\
\text{AxArb} : & \quad CO \subseteq ARB \\
\text{AxCausalArb} : & \quad \forall o \in O. \ CausalPast(CO, o) \oplus ARB \in S
\end{align*}
\]

\( (\text{CausalPast}(CO, o) = \text{the restriction of } CO \text{ to } CO^{-1}(o) \cup \{o\}) \)

\( \text{“} \oplus \text{ ARB” means adding the constraints in ARB) \)
Satisfying Causal Convergence
Satisfying Causal Convergence but not Sequential Consistency

\[ \text{write}(x, 1) \quad \text{read}(y) \triangleright 0 \]

\[ \text{write}(y, 1) \quad \text{read}(x) \triangleright 0 \]
Different Notions of Causal Consistency

Causal memory = Causal consistency + local arbitration
Outline

- Definition(s) of causal consistency
- Characterize all causal consistency violations using bad patterns
- Using bad patterns for verifying data-independent implementations
  - Single-Trace Verification: polynomial time
  - Bad patterns can be recognized with state machines
  - Generic reduction from causal consistency to reachability
  - All-Traces Verification: decidable
Data Independent Implementations

**Observation**: Written values do not influence behaviors.

$\Rightarrow$ We can assume written values are **unique**.
Data Independent Implementations

**Observation**: Written values do not influence behaviors.
⇒ We can assume written values are unique.

\[
\begin{align*}
\text{write}(x, 1) & \quad \text{write}(x, 3) \\
\text{write}(x, 2) & \quad \text{write}(x, 4) \\
\text{read}(x) \uparrow 3
\end{align*}
\]
Data Independent Implementations

**Observation**: Written values do not influence behaviors.  
⇒ We can assume written values are unique.

\[
\begin{align*}
\text{write}(x,1) & \quad \text{write}(x,3) \\
\text{write}(x,2) & \quad \text{write}(x,4) \\
\text{read}(x) & \quad \triangleright 3
\end{align*}
\]
Observation: Written values do not influence behaviors.
⇒ We can assume written values are unique.

Unicity of writes implies a canonical causality relation (included in every other causality relation).
Bad Patterns to Characterize Violations

**Bad pattern**: set of operations related is a particular way
Bad Patterns to Characterize Violations

**Bad pattern**: set of operations related is a particular way

Defined using the following orders:

- **PO** (program order): connects operations from the same site
Bad Patterns to Characterize Violations

**Bad pattern**: set of operations related is a particular way

Defined using the following orders:

- **PO** (program order): connects operations from the same site
- **RF** (reads-from relation): connects write to read
Bad Patterns to Characterize Violations

**Bad pattern**: set of operations related in a particular way

Defined using the following orders:

- **PO** (program order): connects operations from the same site
- **RF** (reads-from relation): connects write to read
- **CO** (causal order): defined as \((PO \cup RF)^+\)
Bad Pattern for Causal Consistency: WriteCORead

- Two writes $w_1$ and $w_2$, and one read $r_1$ on the same variable:
  - $r_1$ reads-from $w_1$
  - $w_1 <_{CO} w_2 <_{CO} r_1$

Example:
write($x, 1$)
write($y, 2$)
read($y$) → 1
write($x, 2$)
read($x$) → 1
read($x$) → 1
Bad Pattern for Causal Consistency: WriteCORead

- Two writes \( w_1 \) and \( w_2 \), and one read \( r_1 \) on the same variable:
  - \( r_1 \) reads-from \( w_1 \)
  - \( w_1 <_{co} w_2 <_{co} r_1 \)

Example:

\[
\begin{align*}
\text{write}(x, 1) & \quad \text{write}(y, 2) \\
\text{write}(x, 2) & \\
\text{read}(y) & \quad \triangleright 2 \\
\text{read}(x) & \quad \triangleright 2 \\
\text{read}(x) & \quad \triangleright 1
\end{align*}
\]
WriteCORead: Litmus tests

\[ w_1 <_{PO} w_2 <_{PO} r_1: \]

write\((x, 1)\)
write\((x, 2)\)
read\((x) \bigtriangledown 1\)
WriteCORead: Litmus tests

\[ w_1 <_PO w_2 <_PO r_1 : \]
write(x,1)
write(x,2)
read(x) ▶ 1

\[ w_1 <_PO w_2 <_CO r_1 : \]
write(x,1)
write(x,2) || read(x) ▶ 1
write(y,3) read(y) ▶ 3
WriteCOR: Litmus tests

\[ w_1 <_{PO} w_2 <_{PO} r_1 : \]
write(x, 1)
write(x, 2)
read(x) ▶ 1

\[ w_1 <_{PO} w_2 <_{CO} r_1 : \]
write(x, 1)
write(x, 2) || read(x) ▶ 1
write(y, 3)

\[ w_1 <_{CO} w_2 <_{PO} r_1 : \]
write(x, 1)
write(y, 3) || write(x, 2)
read(x) ▶ 1

\[ w_1 <_{CO} w_2 <_{CO} r_1 : \]
write(x, 1)
write(x, 2) || write(x, 2) || read(x) ▶ 1
write(y, 3)
write(z, 4)
Bad Patterns for Causal Consistency

- **WriteCORead**: two writes $w_1$ and $w_2$, and one read $r_1$ on some $x$ s.t.
  - $r_1$ reads-from $w_1$
  - $w_1 <_{co} w_2 <_{co} r_1$

- **CyclicCO**: $CO = (PO \cup RF)^+$ is cyclic

- **ThinAir**: a read operation $r = \text{read}(x) \triangleright v$ with $v \neq 0$ s.t.
  - $w \not<_{RF} r$ for every write $w$

- **WriteCOInit**: a read operation $r = \text{read}(x) \triangleright 0$ s.t.
  - $w <_{CO} r$ for some write $w$ on $x$
## Bad Patterns for Causal Consistency Variants

<table>
<thead>
<tr>
<th>Causal Consistency</th>
<th>Causal Memory</th>
<th>Causal Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CyclicCO</td>
<td>CyclicCO</td>
<td>CyclicCO</td>
</tr>
<tr>
<td>WriteCOInitRead</td>
<td>WriteCOInitRead</td>
<td>WriteCOInitRead</td>
</tr>
<tr>
<td>ThinAirRead</td>
<td>ThinAirRead</td>
<td>ThinAirRead</td>
</tr>
<tr>
<td>WriteCORead</td>
<td>WriteCORead</td>
<td>WriteCORead</td>
</tr>
<tr>
<td></td>
<td>WriteHBInitRead</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CyclicHB</td>
<td>CyclicCF</td>
</tr>
</tbody>
</table>
## Bad Patterns for Causal Consistency Variants

<table>
<thead>
<tr>
<th>Causal Consistency</th>
<th>Causal Memory</th>
<th>Causal Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CyclicCO</td>
<td>CyclicCO</td>
<td>CyclicCO</td>
</tr>
<tr>
<td>WriteCOInitRead</td>
<td>WriteCOInitRead</td>
<td>WriteCOInitRead</td>
</tr>
<tr>
<td>ThinAirRead</td>
<td>ThinAirRead</td>
<td>ThinAirRead</td>
</tr>
<tr>
<td>WriteCORead</td>
<td>WriteCORead</td>
<td>WriteCORead</td>
</tr>
<tr>
<td></td>
<td>WriteHBIInitRead</td>
<td>WriteCORead</td>
</tr>
<tr>
<td></td>
<td>CyclicHB</td>
<td>CyclicCF</td>
</tr>
</tbody>
</table>

**Theorem (Bad Patterns)**

A trace doesn’t satisfy the criterion X iff it contains a bad pattern for X.
Definition(s) of causal consistency

Characterize all causal consistency violations using bad patterns

Using bad patterns for verifying data-independent implementations
  - Single-Trace Verification: polynomial time
  - Bad patterns can be recognized with state machines
  - Generic reduction from causal consistency to reachability
  - All-Traces Verification: decidable
Polynomial-Time Single-Trace Verification

Theorem (Single-Trace Verification)

Singe-Trace Verification of causal consistency is \textbf{NP}-complete.
Polynomial-Time Single-Trace Verification

Theorem (Single-Trace Verification)

Single-Trace Verification of causal consistency is **NP-complete**.

Theorem (Single-Trace Verification)

Single-Trace Verification of causal consistency is **polynomial** when writes are **unique**.

(By checking the absence of bad patterns.)
Recognizing Bad Patterns with Register Automata

- By data independence, we can use a bounded number of values.
- Registers are needed to store variable names while tracking causality paths.
- **WriteCORead:**

\[
\begin{align*}
\text{CausalPath} & \quad [d \mapsto 3] \\
\text{p, wr(x, 1)} & \quad \text{wrt} := x \\
\text{var} := x & \\
\text{site} := p \\
\end{align*}
\]

\[
\begin{align*}
\text{CausalPath} & \quad [d \mapsto 4] \\
p, \text{wr(x, 2)} & \quad \text{wrt} := x \\
\text{var} := x & \\
\text{site} := p \\
\end{align*}
\]

\[
\begin{align*}
p, \text{rd(x)} \rightarrow 1 & \quad \text{q}_{\text{err}} \\
\text{wrt} := x & \\
\text{site} := p \\
\end{align*}
\]
Recognizing Bad Patterns with Register Automata

- By data independence, we can use a bounded number of values
- Registers are needed to store variable names while tracking causality paths

WriteCOPRead:

\[ q_1 \xrightarrow{p, wr(x, 1)} \text{CausalPath} [d \mapsto 3] \xrightarrow{p, wr(x, 2)} \text{CausalPath} [d \mapsto 4] \xrightarrow{p, rd(x) \rhd 1} q_{err} \]

CausalPath tracks alternations of \( PO \) and \( RF \)
Recognizing Bad Patterns with Register Automata

- By data independence, we can use a bounded number of values
- Registers are needed to store variable names while tracking causality paths
- \textbf{WriteCORead}:

\[
\begin{align*}
\text{wit} & : = x \\
\text{var} & : = x \\
\text{site} & : = p \\
\text{wit} & : = x \\
\text{var} & : = x \\
\text{site} & : = p \\
p, \text{rd}(x) \xrightarrow{1} q_{\text{err}}
\end{align*}
\]

CausalPath tracks alternations of PO and RF
(PTime) Reduction to Reachability/Invariant Checking

Machine $M$ tracking all bad patterns.

**Theorem (Reduction to Reachability)**

An implementation $I$ is **causally consistent** iff $I \times M$ cannot reach an error state.
(PTime) Reduction to Reachability/Invariant Checking

Machine $M$ tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation $I$ is **causally consistent** iff $I \times M$ cannot reach an error state.

- Holds for any data-independent implementation
(PTime) Reduction to Reachability/Invariant Checking

Machine $M$ tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation $I$ is causally consistent iff $I \times M$ cannot reach an error state.

- Holds for any data-independent implementation
- Reuse of existing tools that solve reachability
(PTime) Reduction to Reachability/Invariant Checking

Machine $M$ tracking all bad patterns.

Theorem (Reduction to Reachability)

An implementation $I$ is causally consistent iff $I \times M$ cannot reach an error state.

- Holds for any data-independent implementation
- Reuse of existing tools that solve reachability
- Manual or semi-automated proofs
All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)
All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)

**Theorem (All-Traces Verification)**

*Checking whether all traces of a finite-state implementation are causally consistent is **undecidable**.*
All-Traces Verification

Setting: **Finite** number of **finite-state sites**.
(All traces are modelled by a finite-state automaton)

**Theorem (All-Traces Verification)**

*Checking whether all traces of a finite-state implementation are causally consistent is **undecidable**.*

**Theorem (All-Traces Verification)**

*Checking whether all traces of a data-independent finite-state implementation are causally consistent is **decidable**.*
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)

Future work:

- Bad patterns for other criteria
  (FIFO consistency, . . . )

- For other specifications
  (Multi-Value Register, CRDTs, . . . )

- Application to existing causally consistent systems
to prove their correctness (or find bugs)

Thank you
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations

Future work:

- Bad patterns for other criteria (FIFO consistency, ...)
- For other specifications (Multi-Value Register, CRDTs, ...)
- Application to existing causally consistent systems to prove their correctness (or find bugs)

Thank you
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable

Future work:

- Bad patterns for other criteria (FIFO consistency, . . .)
- For other specifications (Multi-Value Register, CRDTs, . . .)
- Application to existing causally consistent systems to prove their correctness (or find bugs)

Thank you
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to **reachability**: approach for verifying causal consistency

Future work:

- Bad patterns for other criteria (FIFO consistency, . . .)
- For other specifications (Multi-Value Register, CRDTs, . . .)
- Application to existing causally consistent systems to prove their correctness (or find bugs)

Thank you
Summary and Future Work

Summary:
- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to reachability: approach for verifying causal consistency

Future work:
- Bad patterns for **other criteria** (FIFO consistency, . . .)
- for **other specifications** (Multi-Value Register, CRDTs, . . .)
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to reachability: approach for verifying causal consistency

Future work:

- Bad patterns for **other criteria** (FIFO consistency, . . . )
- for **other specifications** (Multi-Value Register, CRDTs, . . . )
- Application to existing **causally consistent systems** to prove their correctness
  (or find bugs)
Summary and Future Work

Summary:

- Difficult to verify causal consistency in general
  (Single-Trace: NP-complete, All-Traces: Undecidable)
- **Bad patterns** for data-independent implementations
  - Single-Trace: PTime, All-Traces: Decidable
  - Polynomial-time reduction to reachability: approach for verifying causal consistency

Future work:

- Bad patterns for **other criteria** (FIFO consistency, . . .)
- for **other specifications** (Multi-Value Register, CRDTs, . . .)
- Application to existing **causally consistent systems** to prove their correctness
  (or find bugs)

Thank you