Automated Reasoning on Consistency Models for Replicated Data Systems with MONA

Isabelle Coget

12^{th} FRIDA Workshop – October 27, 2025

Context

Consistency models

Context

Consistency models

Linearizability: every operation appears to take place atomically, in some order, consistent with the real-time ordering of those operations

Eventual consistency: all replicas converge to the same value eventually

Read-your-writes: each process always read its latest write

Monotonic reads: once a process has read a value of a data item, its future reads will never return an older value

Problems

Problem 1: Given an implementation of a replicated data system, can we formally and fully automatically verify that it satisfies a specific consistency model?

Problem 2: Given a distributed application that uses a (black-box) replicated data system, and assuming this system conforms to a given consistency model, can we formally and fully automatically verify that the application behaves correctly (e.g., is functional or safe)?

More generally:

How can we formally and automatically reason about consistency models in replicated data systems?

Table of Contents

Histories

We define $\mathcal{H}(\mathbb{P}, \mathbb{T}, \mathbb{O}, \mathbb{V})$ as the set of all well-defined finite histories.

- $\mathbb{P}$ is the **finite** set of processes
- $\mathbb{T} = \{read, write\}$ is the set of operation types
- $\mathbb{O}$ is the **finite** set of objects
- $\mathbb{V}$ is the **finite** set of values

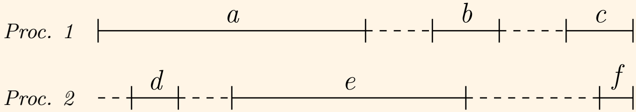
Histories

We define $\mathcal{H}(\mathbb{P}, \mathbb{T}, \mathbb{O}, \mathbb{V})$ as the set of all well-defined finite histories.

- $\mathbb{P}$ is the **finite** set of processes
- $\mathbb{T} = \{read, write\}$ is the set of operation types
- $\mathbb{O}$ is the **finite** set of objects
- $\mathbb{V}$ is the **finite** set of values

A history $H \in \mathcal{H}(\mathbb{P}, \mathbb{T}, \mathbb{O}, \mathbb{V})$ is a set of **operations**; each of its operations having attributes drawn from the sets above.

Histories



The MONA tool

* an automatic verification tool that analyzes logical formulas
* in particular, formulas of a fragment of **weak Monadic Second Order logic (MSO)**
* it translates MSO formulas into **finite**-state automata

The MONA tool

* an automatic verification tool that analyzes logical formulas
* in particular, formulas of a fragment of **weak Monadic Second Order logic (MSO)**
* it translates MSO formulas into **finite**-state automata

MSO logic allows $\forall x, \forall X, \exists x, \exists X, P(x), P(X), P(X, Y), ...$

The MONA tool

- an automatic verification tool that analyzes logical formulas
- in particular, formulas of a fragment of **weak Monadic Second Order logic (MSO)**
- it translates MSO formulas into **finite**-state automata

MSO logic allows $\forall x, \forall X, \exists x, \exists X, P(x), P(X), P(X,Y), ...$

Automata's inputs are **finite** words of bit vectors

The MONA tool

- an automatic verification tool that analyzes logical formulas
- in particular, formulas of a fragment of **weak Monadic Second Order logic (MSO)**
- it translates MSO formulas into **finite**-state automata

MSO logic allows $\forall x, \forall X, \exists x, \exists X, P(x), P(X), P(X,Y), ...$

Automata's inputs are **finite** words of bit vectors

$$\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\cdots\begin{bmatrix}0\\0\end{bmatrix}$$

MSO logic of words of bit vectors

- A vector is called a **position**
- A row is called a **set of positions**

$$
\begin{aligned}
\psi := \; & \forall x.\psi \\
\mid \; & \exists x.\psi \\
\mid \; & \forall X.\psi \\
\mid \; & \exists X.\psi \\
\mid \; & x = \mathrm{succ}(y) \quad \text{with } \mathrm{succ}(y) := y + 1 \\
\mid \; & x = y \quad \text{equality of positions} \\
\mid \; & x \in X \quad \text{position } x \text{ is in the set } X \\
\mid \; & \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid (\psi)
\end{aligned}
$$

Motivation

Theorem

The satisfiability of MSO formulas over finite histories is decidable

Motivation

Theorem
The satisfiability of MSO formulas over finite histories is decidable

Extending the result to infinite, non Zeno histories, seems easy

Monadic second-order logic of histories

Let a and b be some operations.

$$
\begin{aligned}
\phi := \; & a.proc = b.proc \quad \text{with } a.proc, b.proc \in \mathbb{P} \\
\mid \; & a.type = b.type \quad \text{with } a.proc, b.proc \in \mathbb{T} \\
\mid \; & a.obj = b.obj \quad \text{with } a.proc, b.proc \in \mathbb{P} \\
\mid \; & a.ival = b.ival \\
\mid \; & a.oval = b.oval \quad \text{with } a.ival, b.ival, a.ival, b.ival \in \mathbb{V} \\
\mid \; & t < t \\
\mid \; & \forall a.\phi \\
\mid \; & \exists a.\phi \\
\mid \; & \forall A.\phi \\
\mid \; & \exists A.\phi \\
\mid \; & \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid (\phi)
\end{aligned}
$$

$$
\begin{aligned}
t := \; & a.stime \\
\mid \; & a.rtime
\end{aligned}
$$

Monadic second-order logic of histories

Arbitration: $a \xrightarrow{ar} b$ denotes that operation a is considered to be done before operation b

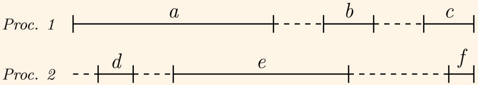
Visibility: $a \xrightarrow{vis} b$ denotes that the effects of operation a are visible to the client performing b

The MONA tool for histories

MONA handles **discrete time**, which can be seen as *snapshots*

$$\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix},\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}0\\0\end{bmatrix}\dots\begin{bmatrix}0\\0\end{bmatrix}$$

Histories represent **continuous time**, which can be seen as a *timeline*



Outline


Traces of executions called *Histories*
Input: words of bit vectors
Consistency verification *(with the MONA tool)*

Outline

Traces of executions called *Histories*

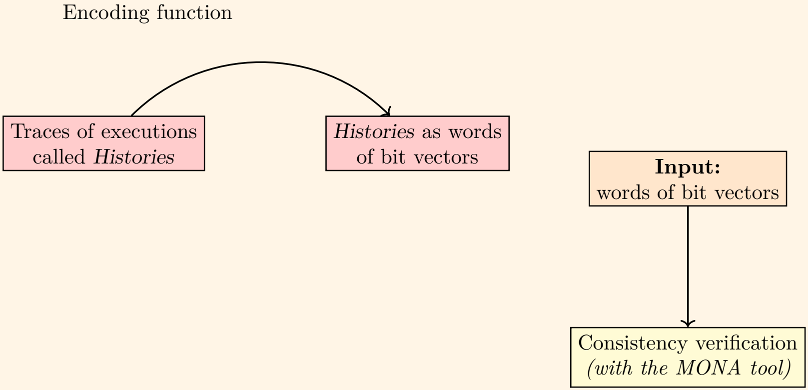
Histories as words of bit vectors

Input: words of bit vectors

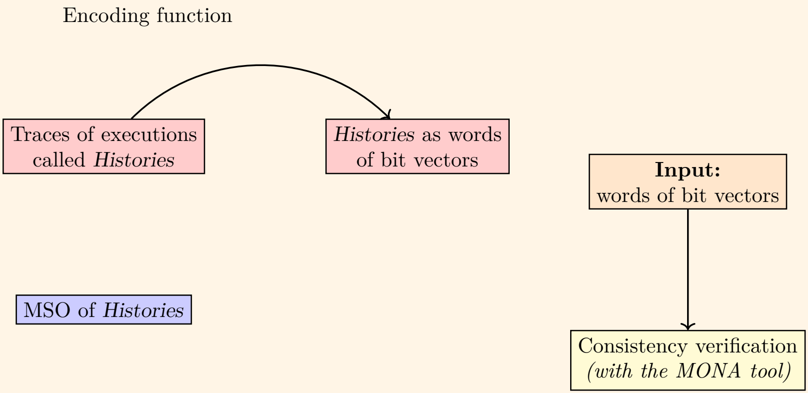
↓

Consistency verification *(with the MONA tool)*

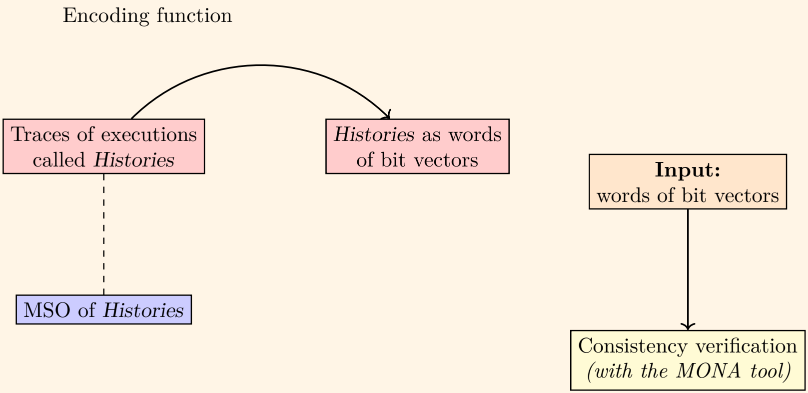
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
Consistency verification *(with the MONA tool)*

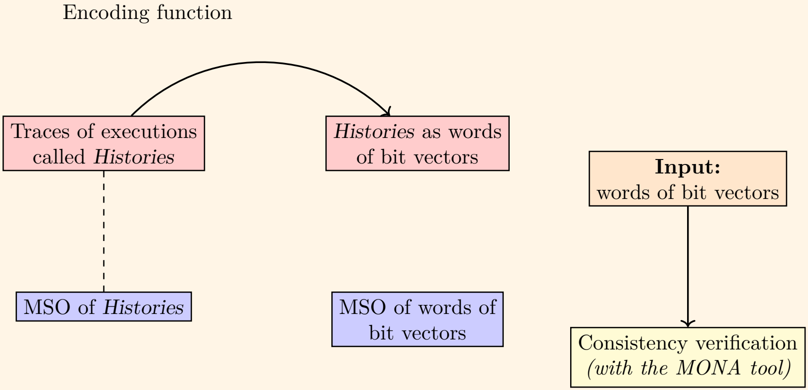
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
Consistency verification *(with the MONA tool)*

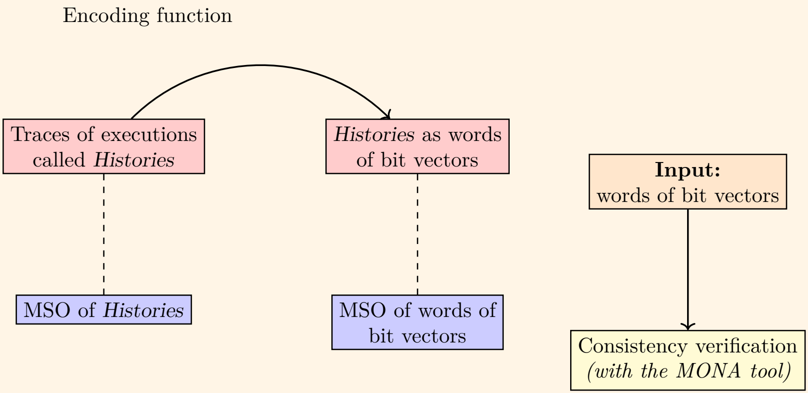
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
Consistency verification *(with the MONA tool)*

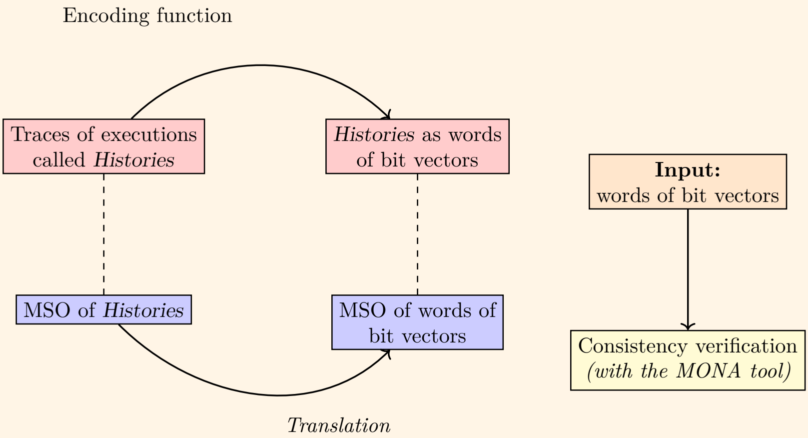
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
MSO of words of bit vectors
Consistency verification (with the MONA tool)

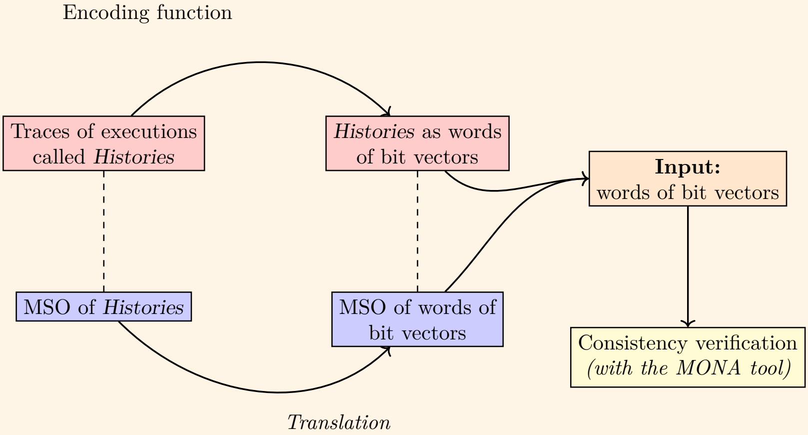
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
MSO of words of bit vectors
Consistency verification (with the MONA tool)

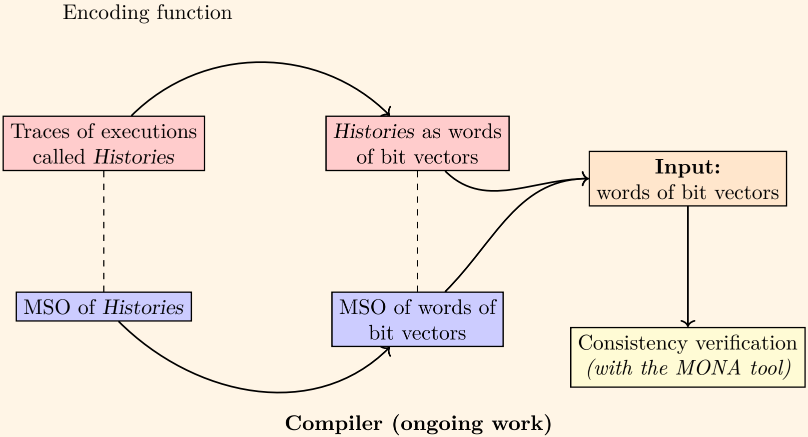
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
MSO of *Histories*
MSO of words of bit vectors
Translation
Input: words of bit vectors
Consistency verification *(with the MONA tool)*

Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
MSO of words of bit vectors
Consistency verification (with the MONA tool)
Translation

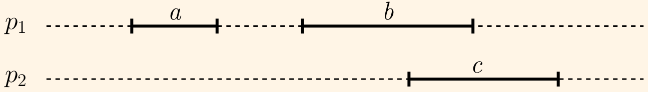
Outline


Encoding function
Traces of executions called *Histories*
Histories as words of bit vectors
Input: words of bit vectors
MSO of *Histories*
MSO of words of bit vectors
Consistency verification *(with the MONA tool)*
Compiler (ongoing work)

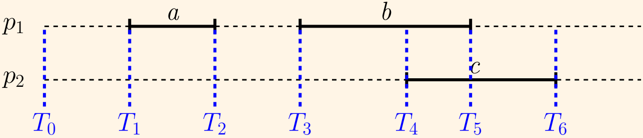
Vector structure

$$\begin{bmatrix} p_1 \\ \vdots \\ p_n \\ t \\ v_1 \\ \vdots \\ v_k \\ o_1 \\ \vdots \\ o_\ell \\ \alpha_1 \\ \vdots \\ \alpha_a \\ \nu_1 \\ \vdots \\ \nu_v \end{bmatrix} \begin{array}{l} \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{processes} \\ \left.\right\} \text{type (here } \textit{read} \text{ or } \textit{write}\text{)} \\ \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{input or output value} \\ \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{object} \\ \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{arbitration relation} \\ \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} \text{visibility} \end{array}$$

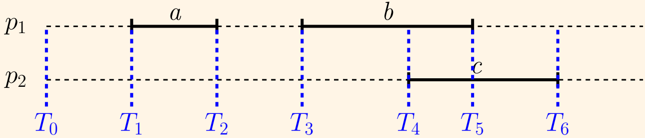
Encoding histories as words



Encoding histories as words

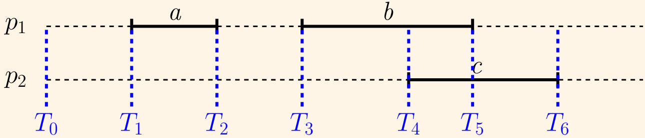


Encoding histories as words



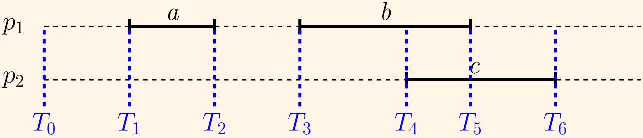
	a	b	c
$proc$	p_1	p_1	p_2
$stime$	T_1	T_3	T_4
$rtime$	T_2	T_5	T_6
$type$	$write$	$read$	$write$
$ival$	14	$\varnothing$	$"fly"$
$oval$	$\varnothing$	$"bee"$	$\varnothing$
obj	y	x	x

Encoding histories as words



	a	b	c
proc	p_1	p_1	p_2
stime	T_1	T_3	T_4
rtime	T_2	T_5	T_6
type	1	0	1
ival	00	$\varnothing$	10
oval	$\varnothing$	01	$\varnothing$
obj	1	0	0

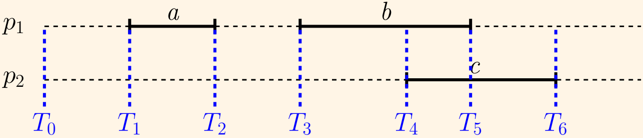
Encoding histories as words



	a	b	c
$proc$	p_1	p_1	p_2
$stime$	T_1	T_3	T_4
$rtime$	T_2	T_5	T_6
$type$	1	0	1
$ival$	00	$\varnothing$	10
$oval$	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{c} T_0 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

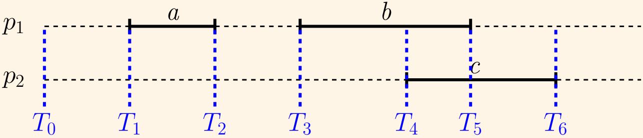
Encoding histories as words



	a	b	c
proc	p_1	p_1	p_2
stime	T_1	T_3	T_4
rtime	T_2	T_5	T_6
type	1	0	1
ival	00	$\varnothing$	10
oval	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{cc} T_0 & T_1 \\ \begin{bmatrix}0\\0\\0\\0\\0\\0\\\vdots\end{bmatrix} & \begin{bmatrix}1\\0\\1\\0\\0\\1\\\vdots\end{bmatrix}\end{array}$$

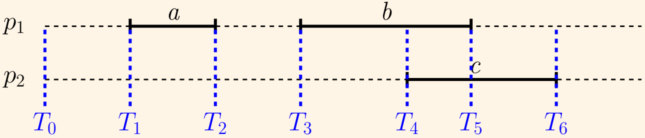
Encoding histories as words



	a	b	c
proc	p_1	p_1	p_2
stime	T_1	T_3	T_4
rtime	T_2	T_5	T_6
type	1	0	1
ival	00	$\varnothing$	10
oval	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{ccc} T_0 & T_1 & T_2 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

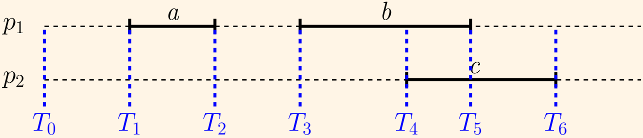
Encoding histories as words



	a	b	c
$proc$	p_1	p_1	p_2
$stime$	T_1	T_3	T_4
$rtime$	T_2	T_5	T_6
$type$	1	0	1
$ival$	00	$\varnothing$	10
$oval$	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{cccc} T_0 & T_1 & T_2 & T_3 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

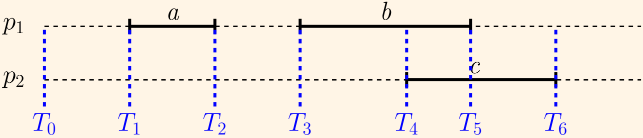
Encoding histories as words



	a	b	c
proc	p_1	p_1	p_2
stime	T_1	T_3	T_4
rtime	T_2	T_5	T_6
type	1	0	1
ival	00	$\varnothing$	10
oval	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{ccccc} T_0 & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \end{array}$$

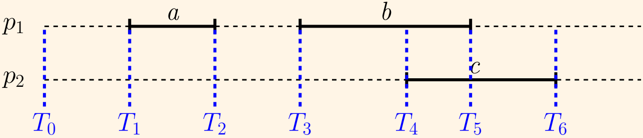
Encoding histories as words



	a	b	c
proc	p_1	p_1	p_2
stime	T_1	T_3	T_4
rtime	T_2	T_5	T_6
type	1	0	1
ival	00	$\varnothing$	10
oval	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$
\begin{array}{cccccc}
T_0 & T_1 & T_2 & T_3 & T_4 & T_5 \\
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} &
\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} &
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} &
\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} &
\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}
\end{array}
$$

Encoding histories as words



	a	b	c
$proc$	p_1	p_1	p_2
$stime$	T_1	T_3	T_4
$rtime$	T_2	T_5	T_6
$type$	1	0	1
$ival$	00	$\varnothing$	10
$oval$	$\varnothing$	01	$\varnothing$
obj	1	0	0

$$\begin{array}{ccccccc} T_0 & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\ \begin{bmatrix}0\\0\\0\\0\\0\\0\\\vdots\end{bmatrix} & \begin{bmatrix}1\\0\\1\\0\\0\\1\\\vdots\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\\vdots\end{bmatrix} & \begin{bmatrix}1\\0\\0\\0\\1\\0\\\vdots\end{bmatrix} & \begin{bmatrix}1\\1\\1\\1\\0\\0\\\vdots\end{bmatrix} & \begin{bmatrix}0\\1\\0\\0\\0\\0\\\vdots\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\\vdots\end{bmatrix} \end{array}$$

Encoding arbitration

$$
\begin{array}{cc}
 & \begin{array}{ccc} p_1 & \cdots & p_n \end{array} \\
\begin{array}{c} p_1 \\ \vdots \\ p_n \end{array} &
\begin{bmatrix}
0 & \alpha_{12} & \cdots & \alpha_{1n} \\
\neg\alpha_{12} & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \alpha_{(n-1)n} \\
\neg\alpha_{1n} & \cdots & \neg\alpha_{(n-1)n} & 0
\end{bmatrix}
\end{array}
$$

A variable α_{ij} is set to 1 if the operation on process i is arbitrated before the operation on process j, and is set to 0 if not

Encoding arbitration

$$
\begin{array}{cc}
 & \begin{array}{ccc} p_1 & \cdots & p_n \end{array} \\
\begin{array}{c} p_1 \\ \vdots \\ p_n \end{array} &
\begin{bmatrix}
0 & \alpha_{12} & \cdots & \alpha_{1n} \\
\neg\alpha_{12} & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \alpha_{(n-1)n} \\
\neg\alpha_{1n} & \cdots & \neg\alpha_{(n-1)n} & 0
\end{bmatrix}
\end{array}
\qquad
\begin{bmatrix}
\alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \\ \vdots \\ \alpha_{23} \\ \alpha_{24} \\ \vdots \\ \alpha_{(n-1)n}
\end{bmatrix}
$$

A variable α_{ij} is set to 1 if the operation on process i is arbitrated before the operation on process j, and is set to 0 if not

Encoding visibility

A way to represent visibility with respect to an operation a, is to encode, for each process p, the most recent operation initiated on p that is visible to a.

$$
\begin{bmatrix} \# \\ \vdots \\ \# \\ p_{1,1} \\ \vdots \\ p_{1,S} \\ \vdots \\ p_{n,1} \\ \vdots \\ p_{n,S} \end{bmatrix}
\begin{array}{l}
\left.\begin{array}{c} \\ \\ \\ \end{array}\right\}\text{previous encoding} \\
\left.\begin{array}{c} \\ \\ \\ \end{array}\right\}\text{most recent operation initiated on } p_1 \text{ that is visible to } a \\
\\
\left.\begin{array}{c} \\ \\ \\ \end{array}\right\}\text{most recent operation initiated on } p_n \text{ that is visible to } a
\end{array}
$$

Compiler

MSO of histories $\rightarrow$ MSO of words of bit vectors

Compiler

MSO of histories → MSO of words of bit vectors

Input:

```
forall X. exists x.  exists y.  x in X and x.type=y.type
```

Output:

```
var2 Process2;
var2 Process1;
var2 TypeRow;
var2 ObjectRow1;
var2 ValueRow1;
pred op(var1 x)= x in Process2 & ex1 y:  y + 1 = x & ~(y in
Process2) | x in Process1 & ex1 y:  y + 1 = x & ~(y in
Process1);
all2 X: ((all1 a:  a in X => op(a)) => ex1 x:  op(x) & ex1
y:  op(y) & x in X & (x in TypeRow <=> y in TypeRow));
```

Conclusion

- MSO satisfiability over finite histories is decidable
- formal and automatic reasoning about consistency models in replicated data systems

Conclusion

- MSO satisfiability over finite histories is decidable
- formal and automatic reasoning about consistency models in replicated data systems

Ongoing work: compiler for translation (in OCaml)

Conclusion

- ▷ MSO satisfiability over finite histories is decidable
- ▷ formal and automatic reasoning about consistency models in replicated data systems

Ongoing work: compiler for translation (in OCaml)

Future work:

First part: extend to infinite histories (should be easy)

Conclusion

- ▷ MSO satisfiability over finite histories is decidable
- ▷ formal and automatic reasoning about consistency models in replicated data systems

Ongoing work: compiler for translation (in OCaml)

Future work:

First part: extend to infinite histories (should be easy)

Second part: weaker assumptions (visibility and arbitration)

Thank you