Fault-tolerant Distributed Runtime Monitoring

Borzoo Bonakdarpour

MICHIGAN STATE UNIVERSITY

Outline of talk

Go Forward

Outline of talk

Runtime Verification (RV)

- A lightweight technique where a *monitor* continually inspects the health of a system under inspection at run time with respect to a *formal specification*.

Runtime Verification (RV)

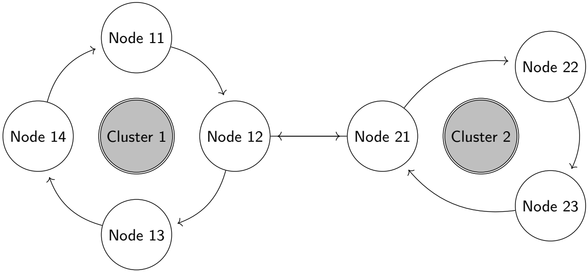
- A lightweight technique where a *monitor* continually inspects the health of a system under inspection at run time with respect to a *formal specification*.
- In *distributed RV*, one or more monitors observe the behavior of a distributed system at run time and collectively verify its correctness with respect to its specification.

Runtime Verification (RV)

- A lightweight technique where a *monitor* continually inspects the health of a system under inspection at run time with respect to a *formal specification*.
- In *distributed RV*, one or more monitors observe the behavior of a distributed system at run time and collectively verify its correctness with respect to its specification.
  - The monitor can be centralized or decentralized.

Applications

- Facebook developed *Cassandra* as an open-source, distributed, No-SQL database management system (no normalization).

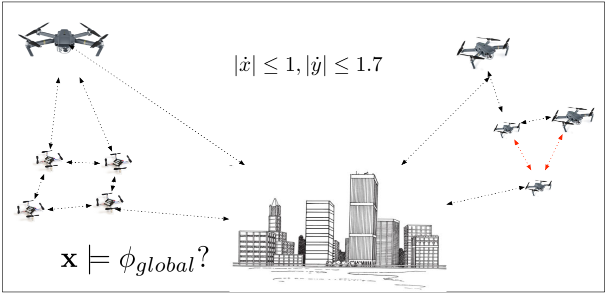


$$\varphi_{\text{rw}} = \bigwedge_{i=0}^{n} \Box\Big(\mathit{write}(i) \rightarrow \Diamond\, \mathit{read}(i)\Big)$$

Motivating Applications

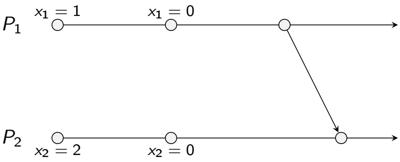
- *Global predicates* on analog signals like UAV position and velocity must be monitored by the ATC, e.g., *mutual separation*:

$$\bigwedge_{i \neq j} \square\, d(x_i, x_j) \geq \delta,$$



Technical Challenge 1: Combinatorial Explosion

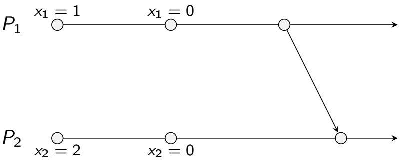
- Although distributed RV deals with *finite executions*, due to lack of a *global clock*, the order of occurrence of events cannot be determined by a runtime monitor.



$$\varphi = \bigcirc(x_1 + x_2 > 1)$$

Technical Challenge 1: Combinatorial Explosion

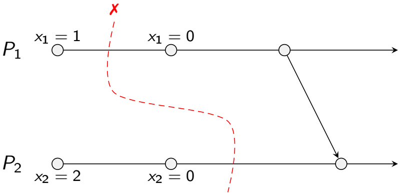
- Although distributed RV deals with *finite executions*, due to lack of a *global clock*, the order of occurrence of events cannot be determined by a runtime monitor.
- Different orders of events may result in *different verification verdicts*.



$$\varphi = \bigcirc(x_1 + x_2 > 1)$$

Technical Challenge 1: Combinatorial Explosion

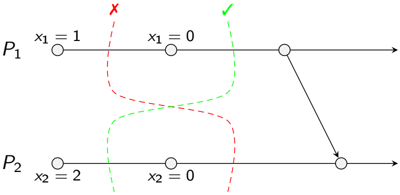
- Although distributed RV deals with *finite executions*, due to lack of a *global clock*, the order of occurrence of events cannot be determined by a runtime monitor.
- Different orders of events may result in *different verification verdicts*.



$$\varphi = \bigcirc(x_1 + x_2 > 1)$$

Technical Challenge 1: Combinatorial Explosion

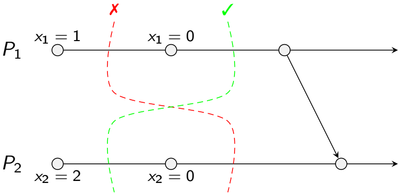
- Although distributed RV deals with *finite executions*, due to lack of a *global clock*, the order of occurrence of events cannot be determined by a runtime monitor.
- Different orders of events may result in *different verification verdicts*.



$$\varphi = \bigcirc(x_1 + x_2 > 1)$$

Technical Challenge 1: Combinatorial Explosion

- Although distributed RV deals with *finite executions*, due to lack of a *global clock*, the order of occurrence of events cannot be determined by a runtime monitor.
- Different orders of events may result in *different verification verdicts*.
- *Enumerating* all possible orders at run time is not practical.



$$\varphi = \bigcirc(x_1 + x_2 > 1)$$

Technical Challenge 1: Combinatorial Explosion

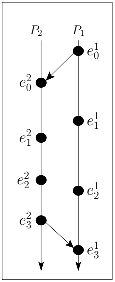
```
{x1=0}
Process P1()
{
   send(P2,m1);
   x1=5;
   x1=10;
   recv(m2);
}
```

```
{x2=0}
Process P2()
{
   recv(m1);
   x2=15;
   x2=20;
   send(P1,m2);
}
```

Technical Challenge 1: Combinatorial Explosion

```
{x1=0}
Process P1()
{
   send(P2,m1);
   x1=5;
   x1=10;
   recv(m2);
}
```

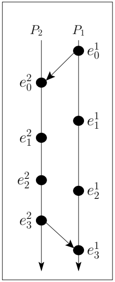
```
{x2=0}
Process P2()
{
   recv(m1);
   x2=15;
   x2=20;
   send(P1,m2);
}
```

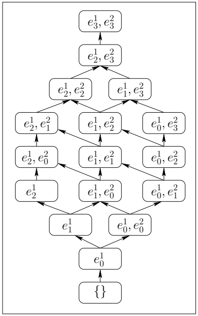


Technical Challenge 1: Combinatorial Explosion

```

    {x1=0}
    Process P1()
    {
       send(P2,m1);
       x1=5;
       x1=10;
       recv(m2);
    }

```

```

    {x2=0}
    Process P2()
    {
       recv(m1);
       x2=15;
       x2=20;
       send(P1,m2);
    }

```

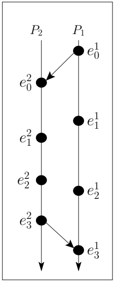


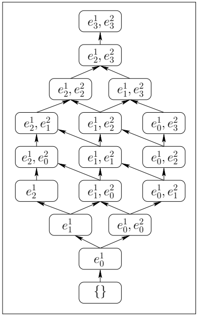


Technical Challenge 1: Combinatorial Explosion

```
{x1=0}
Process P1()
{
   send(P2,m1);
   x1=5;
   x1=10;
   recv(m2);
}
```

```
{x2=0}
Process P2()
{
   recv(m1);
   x2=15;
   x2=20;
   send(P1,m2);
}
```





We need to deal with a *combinatorial* blowup at *run time!*

Technical Challenge 2: Occurrence of Faults

Technical Challenge 2: Occurrence of Faults

Technical Challenge 2: Occurrence of Faults

Example

$$\varphi_{ra_2} = \{\Box(\neg a_1 \neg r_1) \vee [(\neg a_1\ \mathcal{U}\ r_1) \wedge \Diamond a_1]\} \wedge \{\Box(\neg a_2 \neg r_2) \vee [(\neg a_2\ \mathcal{U}\ r_2) \wedge \Diamond a_2]\}$$

Global state: $r_1, a_1 = T$ and $r_2, a_2 = F$

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

Technical Challenge 2: Occurrence of Faults

Example

$$\varphi_{ra_2} = \{\Box(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathcal{U}\, r_1) \wedge \Diamond a_1]\} \wedge \{\Box(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathcal{U}\, r_2) \wedge \Diamond a_2]\}$$

Global state: $r_1, a_1 = T$ and $r_2, a_2 = F$

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

M_0, M_1 →

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

Technical Challenge 2: Occurrence of Faults

Example

$$\varphi_{ra_2} = \{\Box(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathcal{U}\, r_1) \wedge \Diamond a_1]\} \wedge \{\Box(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathcal{U}\, r_2) \wedge \Diamond a_2]\}$$

Global state: $r_1, a_1 = T$ and $r_2, a_2 = F$

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

M_0, M_1 →

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

$M_0 || M_1$ →

M_0

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

Technical Challenge 2: Occurrence of Faults

Example

$$\varphi_{ra_2} = \{\Box(\neg a_1 \neg r_1) \vee [(\neg a_1 \,\mathcal{U}\, r_1) \wedge \Diamond a_1]\} \wedge \{\Box(\neg a_2 \neg r_2) \vee [(\neg a_2 \,\mathcal{U}\, r_2) \wedge \Diamond a_2]\}$$

Global state: $r_1, a_1 = T$ and $r_2, a_2 = F$

M_0

	M_0	M_1
r_1	T	♮
a_1	♮	♮
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	♮	T
a_1	♮	T
r_2	♮	♮
a_2	♮	♮

M_0, M_1

M_0

	M_0	M_1
r_1	T	♮
a_1	♮	♮
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

$M_0 || M_1$

M_0

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_1, M_0

M_0

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	♮	T
a_1	♮	T
r_2	♮	♮
a_2	♮	♮

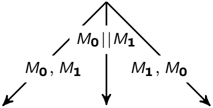
M_0

	M_0	M_1
r_1	T	♮
a_1	♮	♮
r_2	F	♮
a_2	F	♮

$\top$

M_1

	M_0	M_1
r_1	♮	T
a_1	♮	T
r_2	♮	♮
a_2	♮	♮



M_0

	M_0	M_1
r_1	T	♮
a_1	♮	♮
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_0

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_1

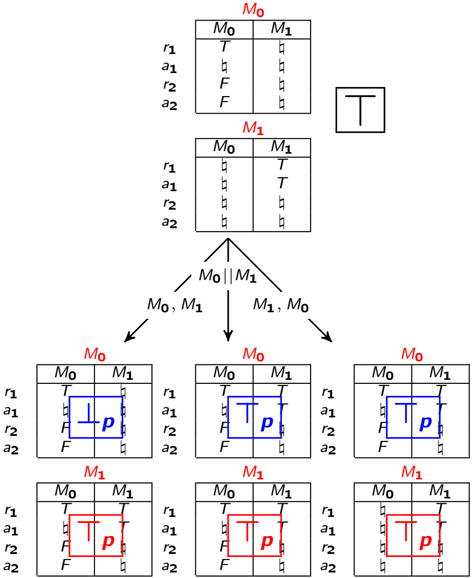
	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_0

	M_0	M_1
r_1	T	T
a_1	♮	T
r_2	F	♮
a_2	F	♮

M_1

	M_0	M_1
r_1	♮	T
a_1	♮	T
r_2	♮	♮
a_2	♮	♮



M0
M0 M1
r1 a1 r2 a2
T ♮ F F
♮ ♮ ♮ ♮
⊤
M1
M0 M1
r1 a1 r2 a2
♮ ♮ ♮ ♮
T T ♮ ♮
M0||M1
M0, M1
M1, M0
⊥ p
⊤ p
⊤ p

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

$\top$

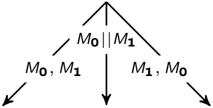
M_0

	M_0	M_1
r_1	$\natural$	$\natural$
a_1	$\natural$	$\natural$
r_2	T	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	F

$\bot$



M_0 ($\bot$ p)

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top$ p)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0 ($\top$ p)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top$ p)

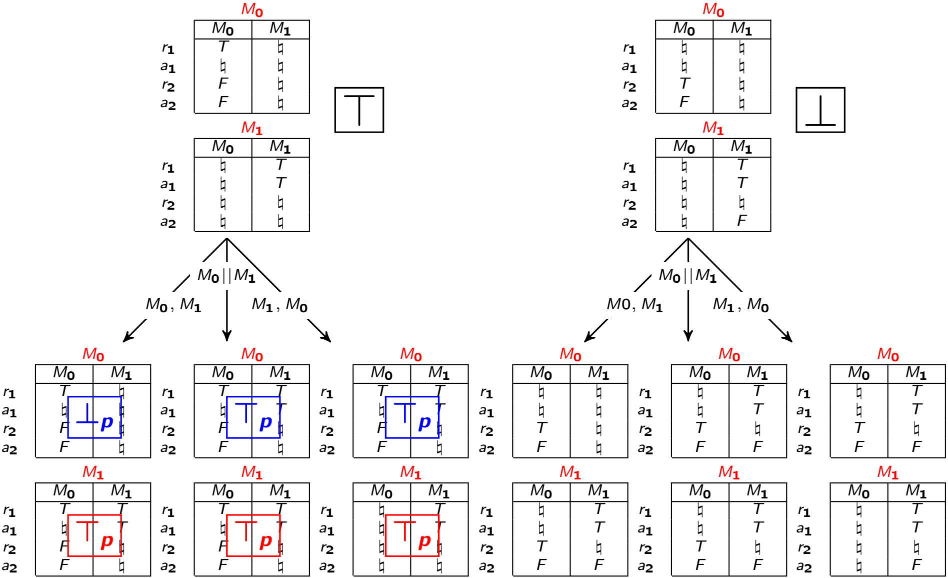
	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0 ($\top$ p)

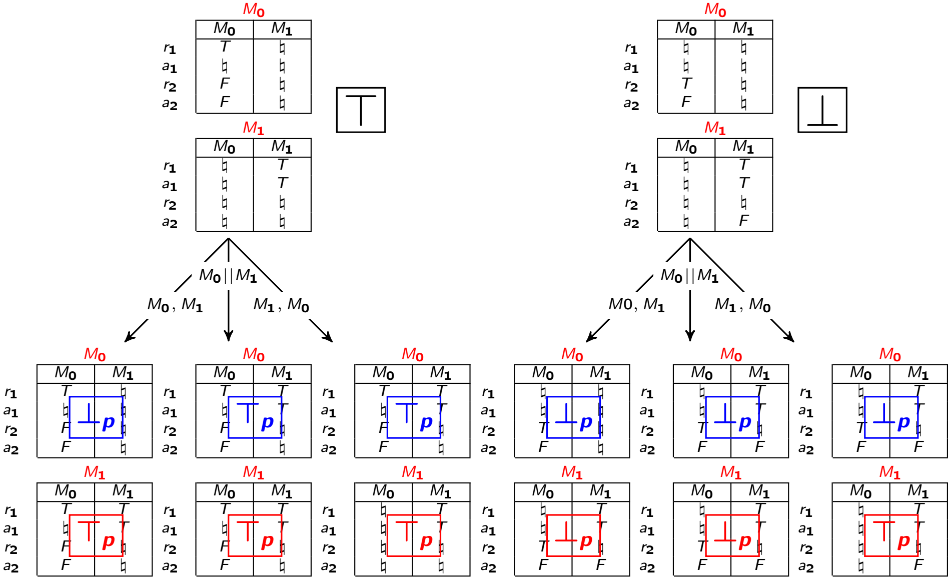
	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top$ p)

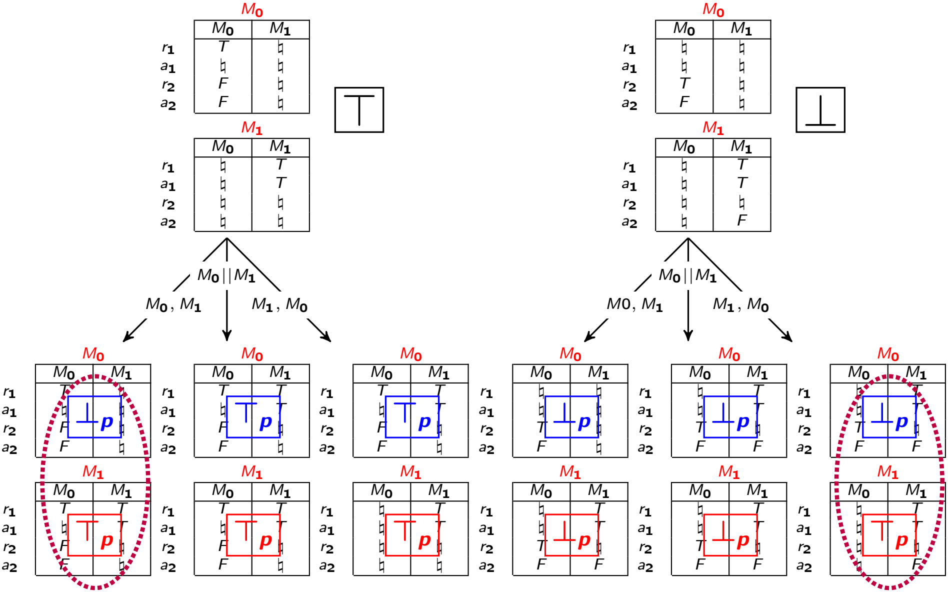
	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$



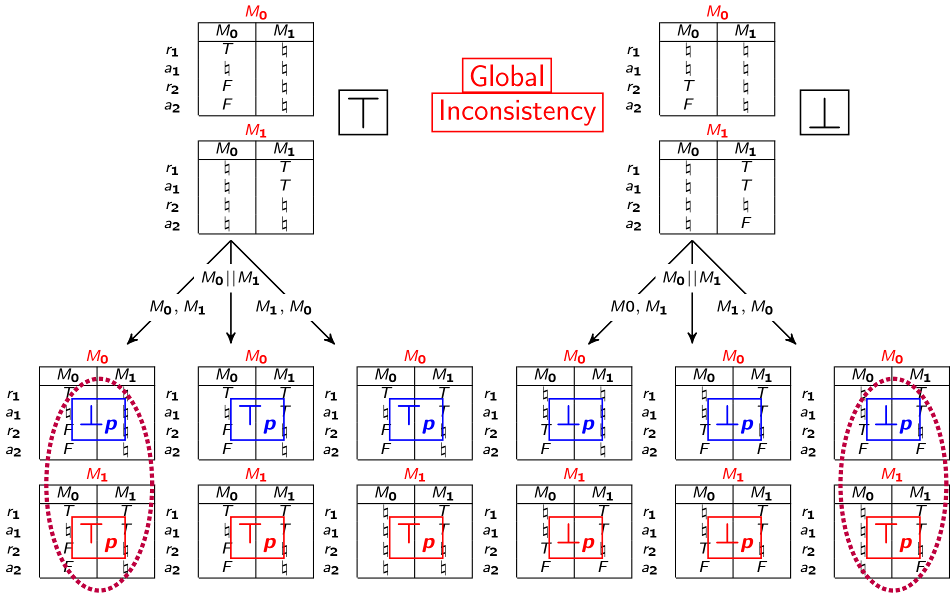
M_0
M_1
r_1 a_1 r_2 a_2
T F ♮
$\top$
$\bot$
$M_0 || M_1$
M_0, M_1
M_1, M_0
M0, M_1
$\bot$ p
$\top$ p



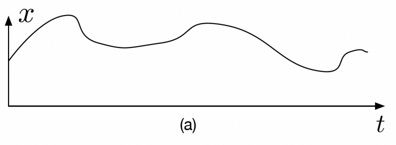
M_0
M_1
r_1
a_1
r_2
a_2
T
F
♮
⊤
⊥
$M_0||M_1$
M_0, M_1
M_1, M_0
$M0, M_1$
p



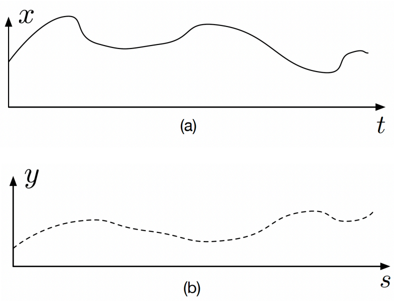
M0
M1
r1
a1
r2
a2
⊤
⊥
M0 || M1
M0 , M1
M1 , M0
M0, M1
p


Global
Inconsistency
⊤
⊥
M0
M1
M0||M1
M0, M1
M1, M0
M0, M1
M1, M0
r1
a1
r2
a2
T
F
p

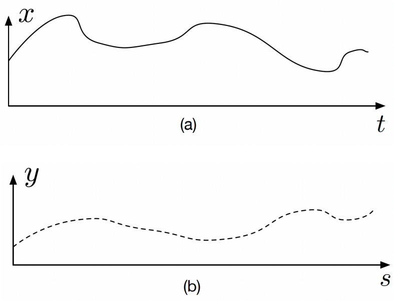
Technical Challenge 3: Continuous Signals



Technical Challenge 3: Continuous Signals

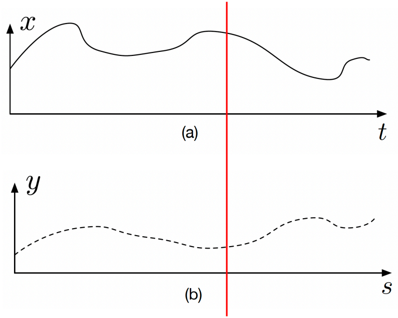


Technical Challenge 3: Continuous Signals



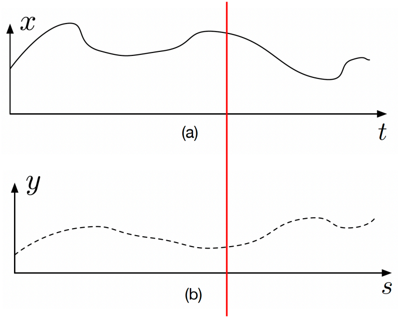
$$\square(x + y \geq 10)$$

Technical Challenge 3: Continuous Signals



$$\Box(x + y \geq 10)$$

Technical Challenge 3: Continuous Signals



$$\square(x + y \geq 10)$$

Even combinatorial enumeration doesn't work!

Related work

- **Asynchronous**
  - H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. A distributed abstraction algorithm for online predicate detection (SRDS 2013).
  - S. D. Stoller. Detecting global predicates in distributed systems with clocks (WDAG 1997).
  - B. Bonakdarpour and M. Mostafa. Decentralized Runtime Verification of LTL Specifications in Distributed Systems (IPDPS 2015).

Related work

- **Asynchronous**
  - H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. A distributed abstraction algorithm for online predicate detection (SRDS 2013).
  - S. D. Stoller. Detecting global predicates in distributed systems with clocks (WDAG 1997).
  - B. Bonakdarpour and M. Mostafa. Decentralized Runtime Verification of LTL Specifications in Distributed Systems (IPDPS 2015).
- **Synchronous**
  - A. Bauer and Y. Falcone. Decentralised LTL monitoring. FMSD 48(1-2), 2016.
  - L. M. Danielsson and C. Sánchez. Decentralized stream runtime verification (RV 2019).

Related work

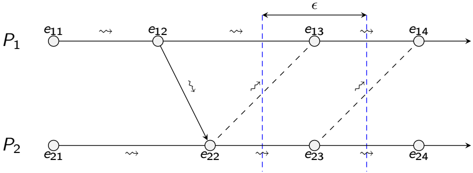
- **Asynchronous**
  - H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. A distributed abstraction algorithm for online predicate detection (SRDS 2013).
  - S. D. Stoller. Detecting global predicates in distributed systems with clocks (WDAG 1997).
  - B. Bonakdarpour and M. Mostafa. Decentralized Runtime Verification of LTL Specifications in Distributed Systems (IPDPS 2015).
- **Synchronous**
  - A. Bauer and Y. Falcone. Decentralised LTL monitoring. FMSD 48(1-2), 2016.
  - L. M. Danielsson and C. Sánchez. Decentralized stream runtime verification (RV 2019).
- V. T. Valapil, S. Yingchareonthawornchai, S. S. Kulkarni, E. Torng, and M. Demirbas. Monitoring partially synchronous distributed systems using SMT solvers (RV 2017).

Our Approach: Partial Synchrony

- We assume a *clock synchronization* algorithm, that ensures *bounded skew* ϵ between all local clocks.

Our Approach: Partial Synchrony

- We assume a *clock synchronization* algorithm, that ensures *bounded skew* ϵ between all local clocks.
- This limits the impact of asynchrony within ϵ.



Outline of talk

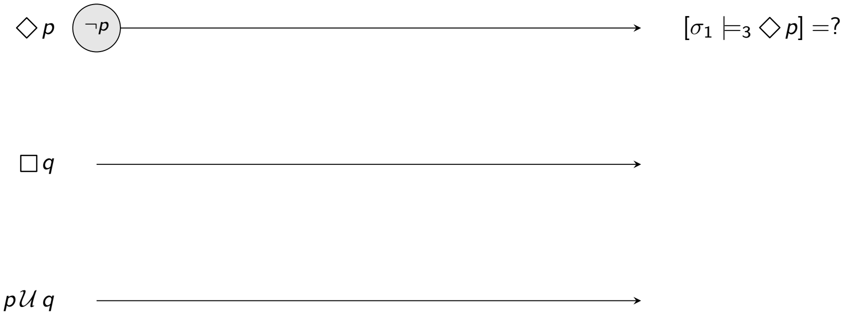
- **Monitoring Distributed Systems under Partial Synchrony** (OPODIS'20)

- **Runtime Verification of Partially-Synchronous Distributed System** (FMSD'23)

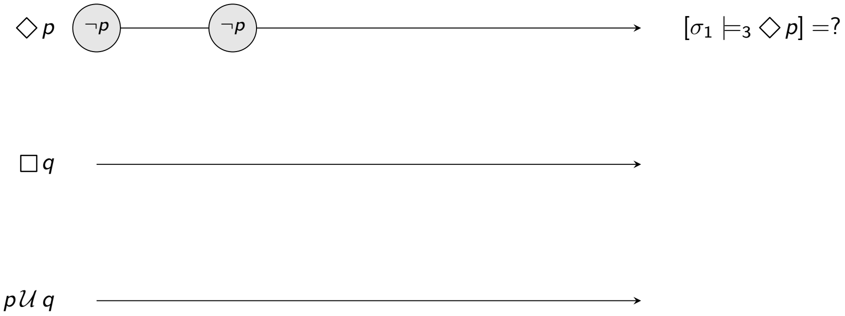
Ritam Ganguly

Anik Momtaz

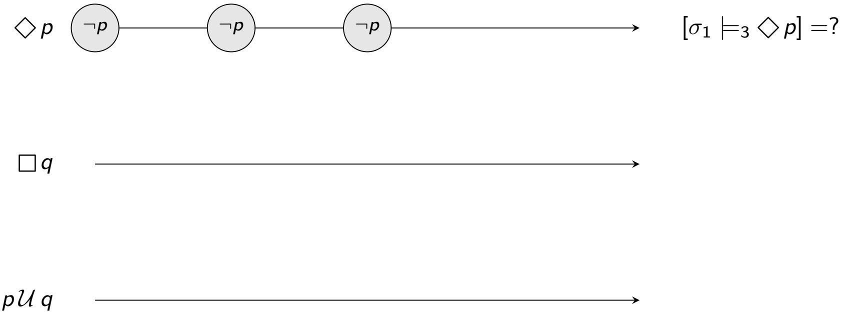
3-Valued LTL Example



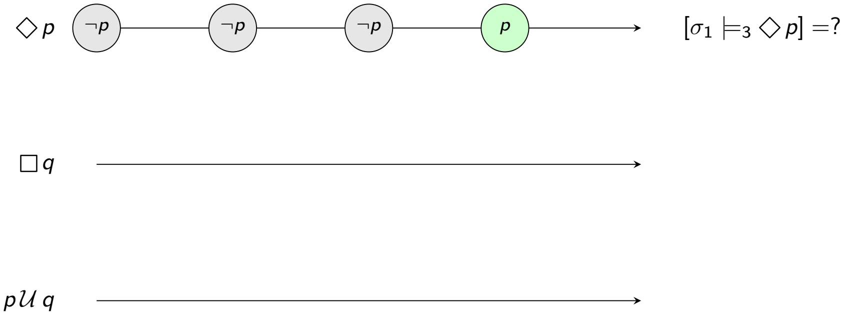
3-Valued LTL Example



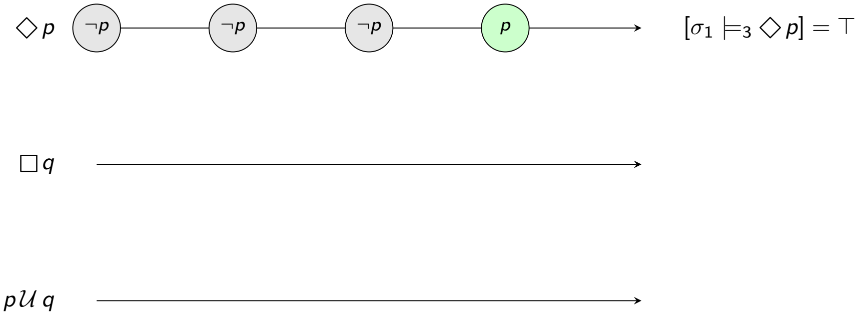
3-Valued LTL Example



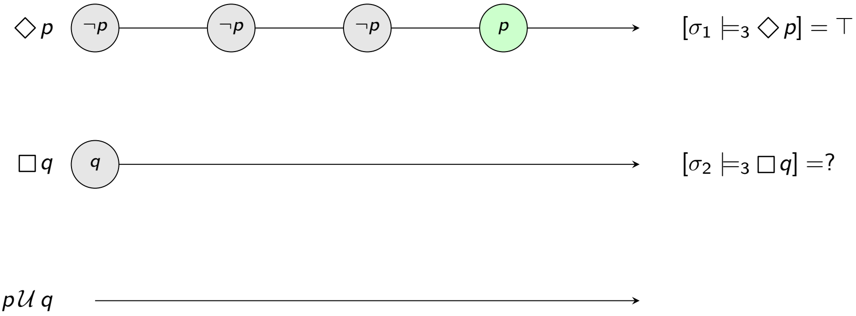
3-Valued LTL Example



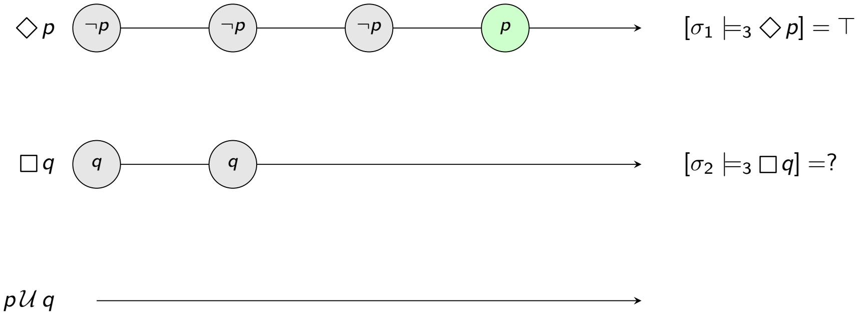
3-Valued LTL Example



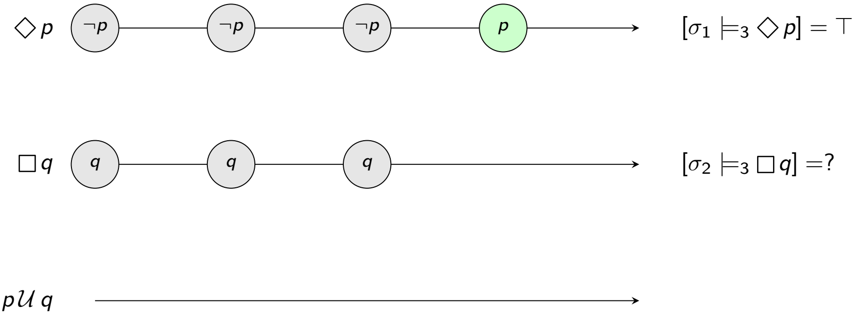
3-Valued LTL Example



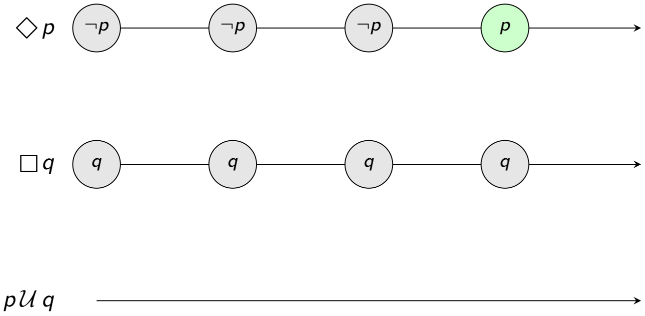
3-Valued LTL Example



3-Valued LTL Example



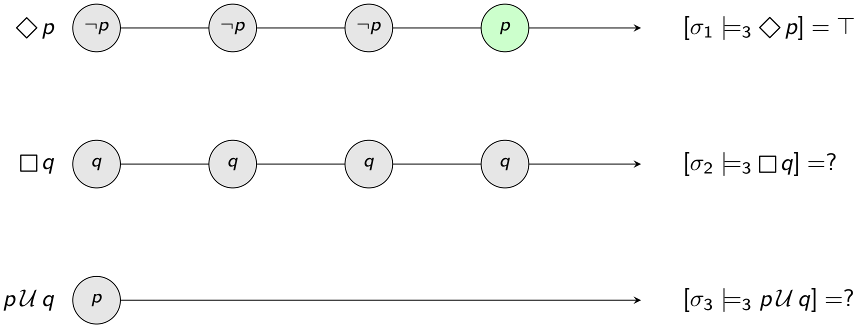
3-Valued LTL Example



$$[\sigma_1 \models_3 \Diamond p] = \top$$

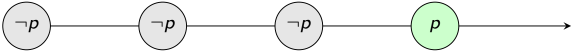
$$[\sigma_2 \models_3 \Box q] = ?$$

3-Valued LTL Example



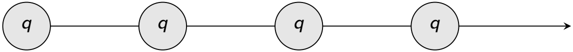
3-Valued LTL Example

$\Diamond p$



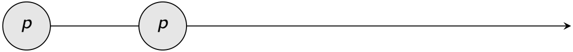
$[\sigma_1 \models_3 \Diamond p] = \top$

$\Box q$



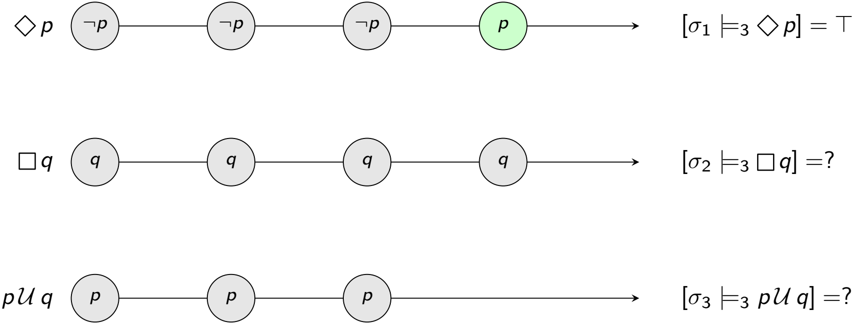
$[\sigma_2 \models_3 \Box q] = ?$

$p \mathcal{U} q$

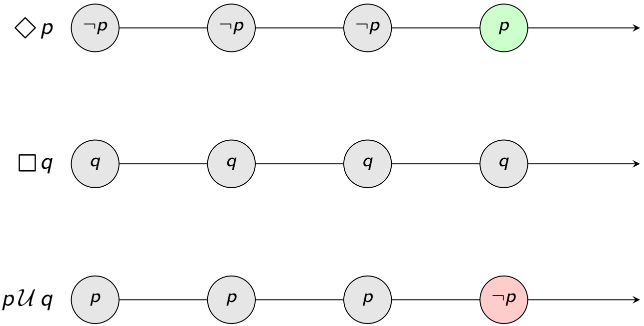


$[\sigma_3 \models_3 p \mathcal{U} q] = ?$

3-Valued LTL Example



3-Valued LTL Example



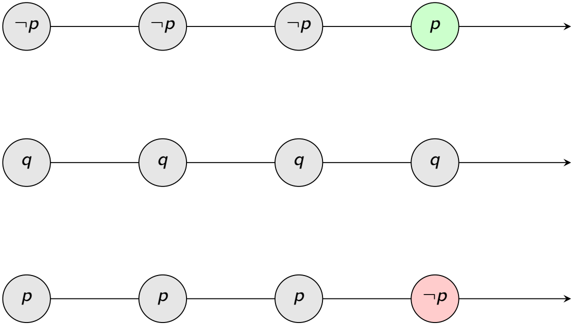
$[\sigma_1 \models_3 \Diamond p] = \top$

$[\sigma_2 \models_3 \Box q] = ?$

$[\sigma_3 \models_3 p \,\mathcal{U}\, q] = ?$

3-Valued LTL Example

$\Diamond p$, $\Box q$, $p \, \mathcal{U} \, q$



$[\sigma_1 \models_3 \Diamond p] = \top$

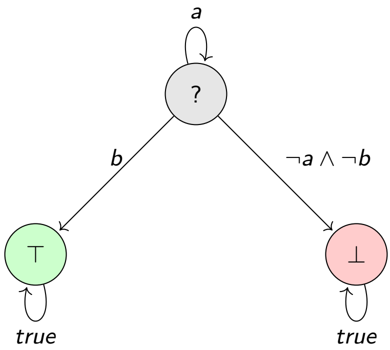
$[\sigma_2 \models_3 \Box q] = ?$

$[\sigma_3 \models_3 p \, \mathcal{U} \, q] = \bot$

LTL_3 Monitor

The *LTL_3 monitor* for a formula φ is the unique deterministic finite state machine $\mathcal{M}_\varphi = (\Sigma, Q, q_0, \delta, \lambda)$, where Q is the set of states, q_0 is the initial state, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, and $\lambda : Q \rightarrow \mathbb{B}_3$ is a function such that $\lambda(\delta(q_0, \alpha)) = [\alpha \models_3 \varphi]$, for every finite trace $\alpha \in \Sigma^*$.

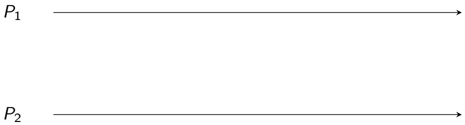
$\varphi = a\,\mathcal{U}\,b$



A. Bauer, M. Leucker, and C. Schallhart. *Runtime Verification for LTL and TLTL*. ACM Transactions on Software Engineering and Methodology (TOSEM), 20(4):14:1–14:64, 2011.

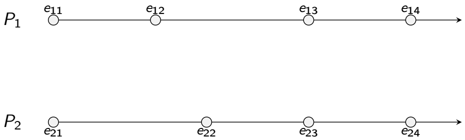
Distributed Computation

- A *distributed computation* on n processes is a tuple $(\mathcal{E}, \rightsquigarrow)$, where $\mathcal{E}$ is a set of events partially ordered by Lamport's *happened-before* ($\rightsquigarrow$) relation.



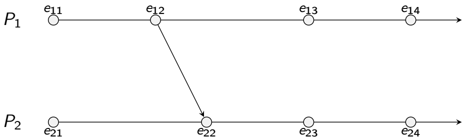
Distributed Computation

- A *distributed computation* on n processes is a tuple $(\mathcal{E}, \rightsquigarrow)$, where $\mathcal{E}$ is a set of events partially ordered by Lamport's *happened-before* $(\rightsquigarrow)$ relation.
- Each *local state* change is considered an event.



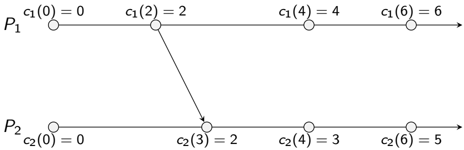
Distributed Computation

- A *distributed computation* on n processes is a tuple $(\mathcal{E}, \rightsquigarrow)$, where $\mathcal{E}$ is a set of events partially ordered by Lamport's *happened-before* ($\rightsquigarrow$) relation.
- Each *local state* change is considered an event.
- *Communication* between processes is represented by send and receive message transmissions.



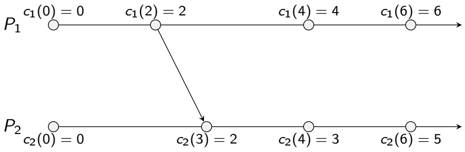
Distributed Computation

- The *local clock* (or time) of a process P_i, where $i \in [1, n]$, can be represented by an increasing function $c_i : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, where $c_i(\chi)$ is the value of the local clock at global time χ.



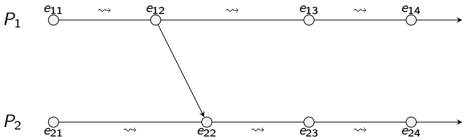
Distributed Computation

- The *local clock* (or time) of a process P_i, where $i \in [1, n]$, can be represented by an increasing function $c_i : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, where $c_i(\chi)$ is the value of the local clock at global time χ.
- For any two processes P_i and P_j, we have $\forall \chi \in \mathbb{R}_{\geq 0}.|c_i(\chi) - c_j(\chi)| < \epsilon$, with $\epsilon > 0$ being the maximum *clock skew*.



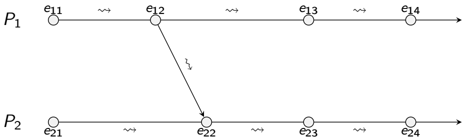
Distributed Computation

- In every process P_i, all events are totally ordered. That is, $\forall \tau, \tau' \in \mathbb{R}_+ . \forall \sigma, \sigma' \in \mathbb{Z}_{\geq 0} . (\sigma < \sigma') \rightarrow (e^i_{\tau,\sigma} \rightsquigarrow e^i_{\tau',\sigma'})$.



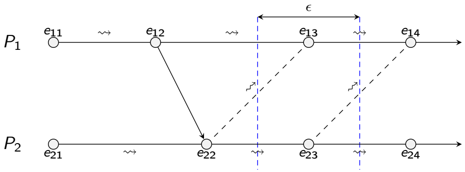
Distributed Computation

- In every process P_i, all events are totally ordered. That is, $\forall \tau, \tau' \in \mathbb{R}_+ . \forall \sigma, \sigma' \in \mathbb{Z}_{\geq 0} . (\sigma < \sigma') \rightarrow (e^i_{\tau,\sigma} \rightsquigarrow e^i_{\tau',\sigma'})$.
- If e is a message send event in a process, and f is the corresponding receive event by another process, then we have $e \rightsquigarrow f$.



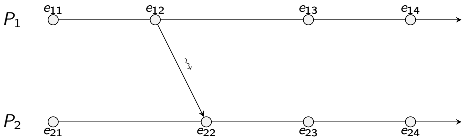
Distributed Computation

- In every process P_i, all events are totally ordered. That is, $\forall \tau, \tau' \in \mathbb{R}_{+}. \forall \sigma, \sigma' \in \mathbb{Z}_{\geq 0}. (\sigma < \sigma') \rightarrow (e^i_{\tau,\sigma} \rightsquigarrow e^i_{\tau',\sigma'})$.
- If e is a message send event in a process, and f is the corresponding receive event by another process, then we have $e \rightsquigarrow f$.
- For any two processes P_i and P_j, and any two events $e^i_{\tau,\sigma}, e^j_{\tau',\sigma'} \in \mathcal{E}$, if $\tau + \epsilon < \tau'$, then $e^i_{\tau,\sigma} \rightsquigarrow e^j_{\tau',\sigma'}$, where ϵ is the maximum *clock skew*.



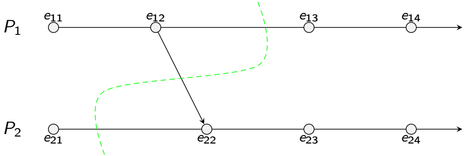
Distributed Computation

- In every process P_i, all events are totally ordered. That is, $\forall \tau, \tau' \in \mathbb{R}_+ . \forall \sigma, \sigma' \in \mathbb{Z}_{\geq 0} . (\sigma < \sigma') \rightarrow (e^i_{\tau,\sigma} \rightsquigarrow e^i_{\tau',\sigma'})$.
- If e is a message send event in a process, and f is the corresponding receive event by another process, then we have $e \rightsquigarrow f$.
- For any two processes P_i and P_j, and any two events $e^i_{\tau,\sigma}, e^j_{\tau',\sigma'} \in \mathcal{E}$, if $\tau + \epsilon < \tau'$, then $e^i_{\tau,\sigma} \rightsquigarrow e^j_{\tau',\sigma'}$, where ϵ is the maximum *clock skew*.
- If $e \rightsquigarrow f$ and $f \rightsquigarrow g$, then $e \rightsquigarrow g$.



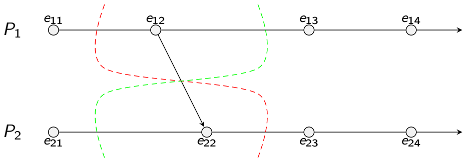
Distributed Computation

- Given a *distributed computation* $(\mathcal{E}, \rightsquigarrow)$, a subset of events $C \subseteq \mathcal{E}$ is said to form a *consistent cut* iff when C contains an event e, then it contains all events that happened-before e. Formally, $\forall e \in \mathcal{E}.(e \in C) \wedge (f \rightsquigarrow e) \rightarrow f \in C$.



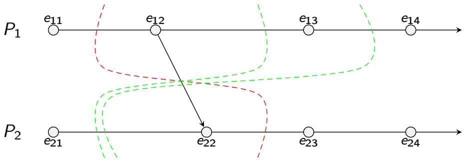
Distributed Computation

- Given a *distributed computation* $(\mathcal{E}, \rightsquigarrow)$, a subset of events $C \subseteq \mathcal{E}$ is said to form a *consistent cut* iff when C contains an event e, then it contains all events that happened-before e. Formally, $\forall e \in \mathcal{E}.(e \in C) \wedge (f \rightsquigarrow e) \rightarrow f \in C$.



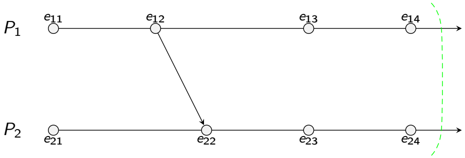
Distributed Computation

- Given a *distributed computation* $(\mathcal{E}, \rightsquigarrow)$, a subset of events $C \subseteq \mathcal{E}$ is said to form a *consistent cut* iff when C contains an event e, then it contains all events that happened-before e. Formally, $\forall e \in \mathcal{E}.(e \in C) \land (f \rightsquigarrow e) \rightarrow f \in C$.



Distributed Computation

- Given a *distributed computation* $(\mathcal{E}, \rightsquigarrow)$, a subset of events $C \subseteq \mathcal{E}$ is said to form a *consistent cut* iff when C contains an event e, then it contains all events that happened-before e. Formally, $\forall e \in \mathcal{E}.(e \in C) \wedge (f \rightsquigarrow e) \rightarrow f \in C$.
- The *frontier* of a consistent cut C, denoted $\text{front}(C)$ is the set of events that happen last in the cut.



Formal Problem Statement

- A *valid sequence* of consistent cuts is of the form $C_0 C_1 C_2 \cdots$, where for all $i \geq 0$, we define the set of all traces as:

$$\mathrm{Tr}(\mathcal{E}, \rightsquigarrow) = \left\{ \mathrm{front}(C_0)\mathrm{front}(C_1) \cdots \mid C_0 C_1 C_2 \cdots \in \mathcal{C} \right\}$$

Formal Problem Statement

- A *valid sequence* of consistent cuts is of the form $C_0 C_1 C_2 \cdots$, where for all $i \geq 0$, we define the set of all traces as:

$$\mathsf{Tr}(\mathcal{E}, \rightsquigarrow) = \left\{ \mathsf{front}(C_0)\mathsf{front}(C_1) \cdots \mid C_0 C_1 C_2 \cdots \in \mathcal{C} \right\}$$

Problem Statement

Given a finite distributed computation $(\mathcal{E}, \rightsquigarrow)$ and an LTL formula φ, compute the following:

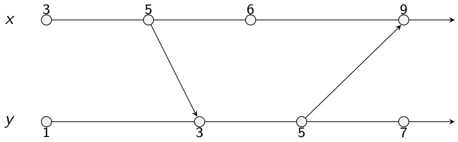
$$[(\mathcal{E}, \rightsquigarrow) \models_3 \varphi] = \left\{ [(\alpha, \rightsquigarrow) \models_3 \varphi] \mid \alpha \in \mathsf{Tr}(\mathcal{E}, \rightsquigarrow) \right\}$$

Solving the Problem

- Given a *distributed computation* $(\mathcal{E}, \rightsquigarrow)$ and an LTL formula φ, our goal is to transform the monitoring problem into an SMT problem.
- In order to ensure that all possible verdicts are explored, we generate an SMT instance for:
  1. The distributed computation $(\mathcal{E}, \rightsquigarrow)$.
  2. Each possible path in the LTL_3 monitor.
- **SMT example:** Is $\forall x.\exists y.f(x) = y + 3$ satisfiable?

SMT-based Solution (Uninterpreted Function)

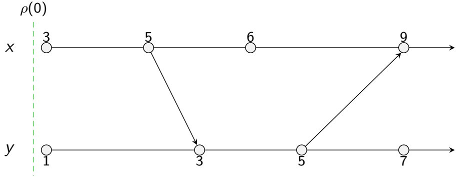
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

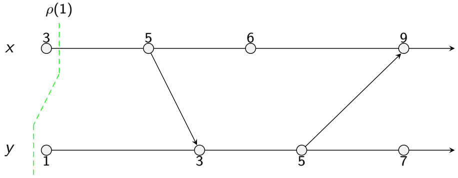
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \to 2^{\mathcal{E}}$.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

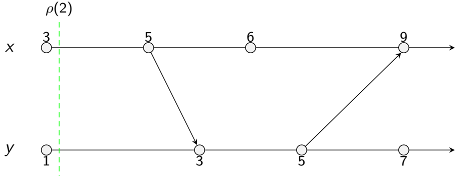
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \to 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

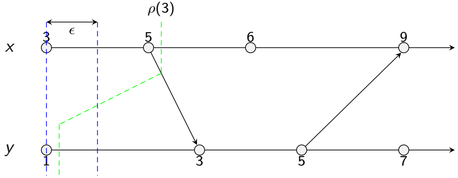
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \to 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

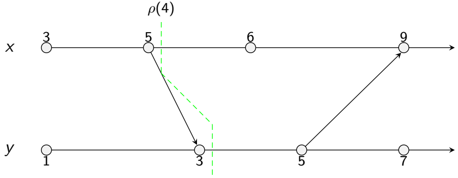
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

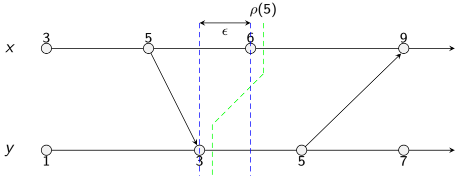
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

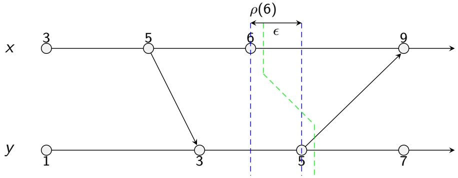
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

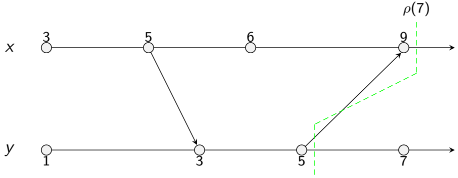
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

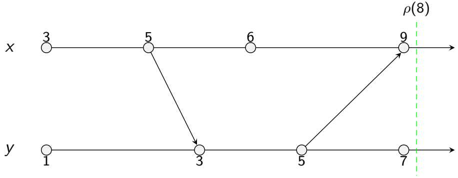
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

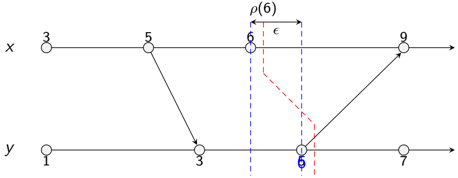
- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.



$$\varphi = \Box(x > y)$$

SMT-based Solution (Uninterpreted Function)

- In order to identify the sequence of consistent cuts whose run on the monitor starts from q_0 and ends in q_m, we introduce an *uninterpreted function* $\rho : \mathbb{Z}_{\geq 0} \rightarrow 2^{\mathcal{E}}$.
- If the SMT instance is satisfiable, then the interpretation of ρ is the sequence of consistent cuts that ends in monitor state q_m.
- Otherwise, no ordering of concurrent events results in the verdict given by state q_m.



$$\varphi = \square(x > y)$$

SMT-based Solution (Constraints over ρ)

We first identify the constraints over *uninterpreted function* ρ, whose interpretation is a sequence of consistent cuts that starts and ends in the given monitor automaton path:

1. Each element in the range of ρ is a *consistent cut*:

$$\forall i \in [0, m]. \forall e, e' \in \mathcal{E}. \Big((e' \rightsquigarrow e) \wedge (e \in \rho(i)) \Big) \rightarrow \Big(e' \in \rho(i) \Big)$$

SMT-based Solution (Constraints over ρ)

We first identify the constraints over *uninterpreted function* ρ, whose interpretation is a sequence of consistent cuts that starts and ends in the given monitor automaton path:

1. Each element in the range of ρ is a *consistent cut*:

$$\forall i \in [0, m].\forall e, e' \in \mathcal{E}.\Big(\big(e' \rightsquigarrow e\big) \land \big(e \in \rho(i)\big)\Big) \rightarrow \Big(e' \in \rho(i)\Big)$$

2. Each consistent cut in the sequence has one more event than its predecessor:

$$\forall i \in [0, m].\ |\rho(i+1)| = |\rho(i)| + 1$$

SMT-based Solution (Constraints over ρ)

We first identify the constraints over *uninterpreted function* ρ, whose interpretation is a sequence of consistent cuts that starts and ends in the given monitor automaton path:

1. Each element in the range of ρ is a *consistent cut*:

$$\forall i \in [0, m].\forall e, e' \in \mathcal{E}.\Big((e' \leadsto e) \wedge (e \in \rho(i))\Big) \rightarrow \Big(e' \in \rho(i)\Big)$$

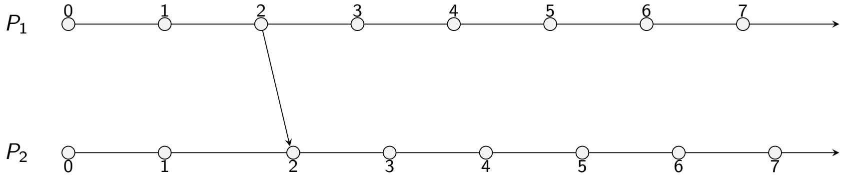
2. Each consistent cut in the sequence has one more event than its predecessor:

$$\forall i \in [0, m].\ |\rho(i+1)| \ = \ |\rho(i)| + 1$$

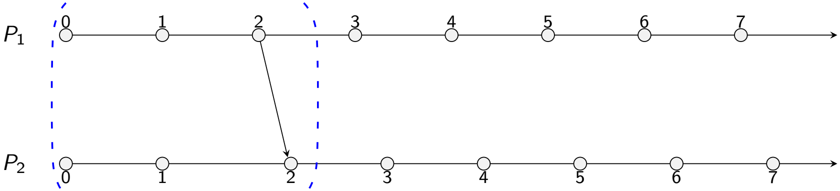
3. Each predecessor of a consistent cut is a subset of the current consistent cut:

$$\forall i \in [0, m].\ \rho(i) \subseteq \rho(i+1)$$

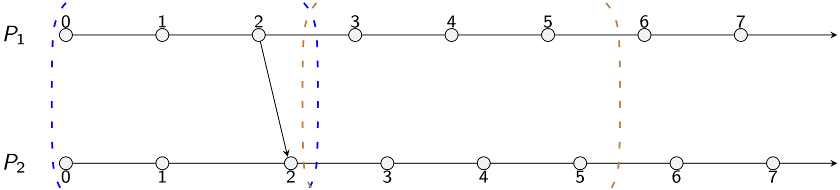
Optimization – Segmentation



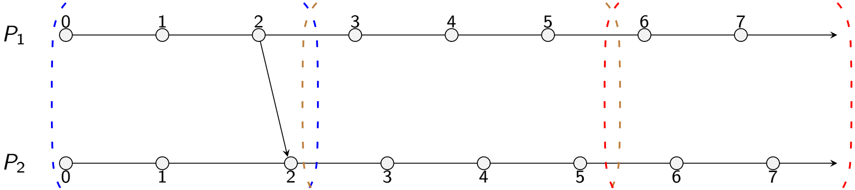
Optimization – Segmentation



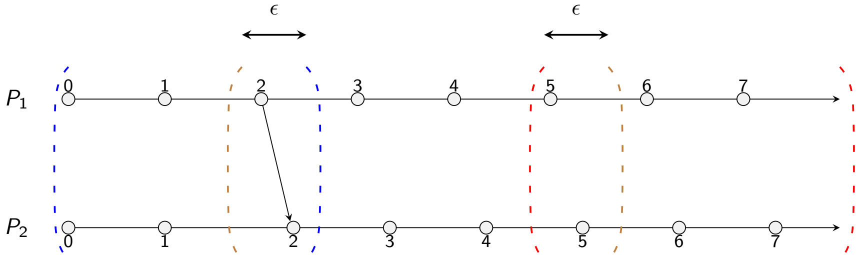
Optimization – Segmentation



Optimization – Segmentation



Optimization – Segmentation



Optimization – Exploiting Parallel Processing

	seg_1			seg_2			seg_3			seg_4		
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
q_0	T	F	F	T	T	F	T	T	T	T	T	T
$q_\top$	F	F	F	F	T	F	F	T	F	F	T	F
$q_\bot$	F	F	F	F	F	T	F	F	T	F	F	T

Figure: Reachability Matrix for $a \mathcal{U} b$

Optimization – Exploiting Parallel Processing

	seg_1			seg_2			seg_3			seg_4		
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
q_0	T	F	F	T	T	F	T	T	T	T	T	T
$q_\top$	F	F	F	F	T	F	F	T	F	F	T	F
$q_\bot$	F	F	F	F	F	T	F	F	T	F	F	T

Figure: Reachability Matrix for $a \mathcal{U} b$

Optimization – Exploiting Parallel Processing

	seg_1			seg_2			seg_3			seg_4		
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
q_0	T	F	F	T	T	F	T	T	T	T	T	T
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\top$	F	F	F	F	T	F	F	T	F	F	T	F
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\bot$	F	F	F	F	F	T	F	F	T	F	F	T

Figure: Reachability Matrix for $a \mathcal{U} b$

Optimization – Exploiting Parallel Processing

	seg_1			seg_2			seg_3			seg_4		
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
q_0	T	F	F	T	T	F	T	T	T	T	T	T
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\top$	F	F	F	F	T	F	F	T	F	F	T	F
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\bot$	F	F	F	F	F	T	F	F	T	F	F	T

Figure: Reachability Matrix for $a \mathcal{U} b$

Optimization – Exploiting Parallel Processing

	seg_1			seg_2			seg_3			seg_4		
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
q_0	T	F	F	T	T	F	T	T	T	T	T	T
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\top$	F	F	F	F	T	F	F	T	F	F	T	F
	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$	q_0	$q_\top$	$q_\bot$
$q_\bot$	F	F	F	F	F	T	F	F	T	F	F	T

Figure: Reachability Matrix for $a\mathcal{U}\,b$

Experimental setup

- Two phases
  - Data Collection
    - *Synthetic Experiments:* single core and effect of parallelization
    - *Cassandra:* moderate and extreme load scenario

[1] All LTL specification are taken from: https://matthewbdwyer.github.io/psp/patterns/ltl.html

Experimental setup

- Two phases
  - Data Collection
    - *Synthetic Experiments:* single core and effect of parallelization
    - *Cassandra:* moderate and extreme load scenario
  - Verification

[1] All LTL specification are taken from: https://matthewbdwyer.github.io/psp/patterns/ltl.html

Experimental setup

- Two phases
  - Data Collection
    - *Synthetic Experiments:* single core and effect of parallelization
    - *Cassandra:* moderate and extreme load scenario
  - Verification
- Events are *evenly spread* out over the entire length of the trace using a delay, and computation and communicating events are uniformly distributed.

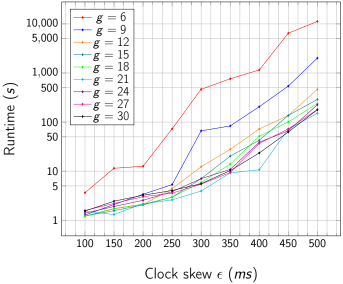
[1] All LTL specification are taken from: https://matthewbdwyer.github.io/psp/patterns/ltl.html

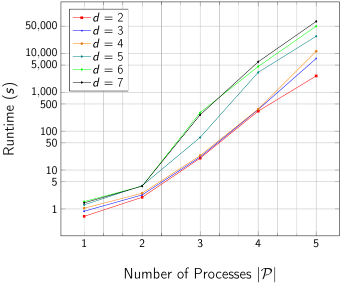
Experimental setup

- Two phases
  - Data Collection
    - *Synthetic Experiments:* single core and effect of parallelization
    - *Cassandra:* moderate and extreme load scenario
  - Verification
- Events are *evenly spread* out over the entire length of the trace using a delay, and computation and communicating events are uniformly distributed.
- *Parameters*: (1) Number of processes (2) Computation duration (3) Number of segments (4) Event rate per process per second (5) Maximum clock skew (6) Number of messages sent per second (7) Formulas under monitoring LTL[1] formulas under monitoring

[1] All LTL specification are taken from: https://matthewbdwyer.github.io/psp/patterns/ltl.html

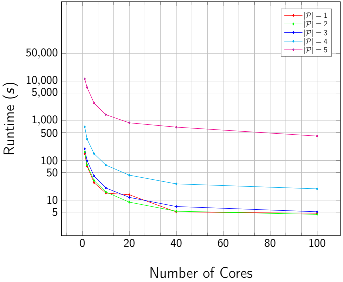
Impact of Partial Synchrony and Predicate Structure

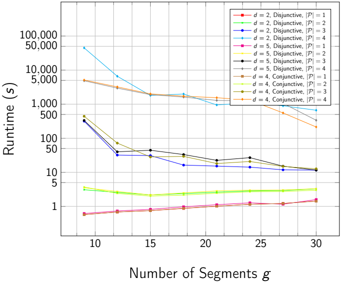




$$l = 2s \qquad r = 100/s$$

Parallelization and Segment Count





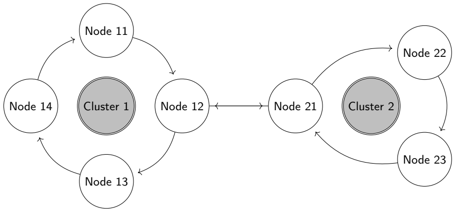
$$l = 2s \qquad r = 100/s$$

How realistic is it?

- **Extreme load scenario:** Netflix, where 1 million writes per second
- **Moderate load scenario:** Google Drive, which allows maximum 500 requests per 100 seconds per project and 100 requests per seconds per user, i.e.,5events/sec per project and a user can generate1event/sec on an average

Cassandra Setup

Cassandra is a open-source, distributed, *no-SQL* database.



- The fastest datacenter ping was received at 41*ms*.
- We use a private broadband that offers a speed of 100 Mbps with 100*ms* latency.
- Processes are capable of reading, writing, and updating all entries of the database with uniform distribution.
- Each process selects the available node at run time.

Cassandra Specification

- *Eventual consistency:*

$$\varphi_{\mathsf{rw}} = \bigwedge_{i=0}^{n} \Box \Big(\mathit{write}(i) \rightarrow \Diamond\, \mathit{read}(i) \Big)$$

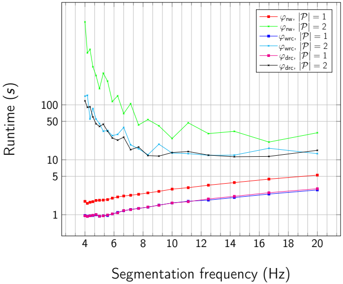
- Cassandra does not implicitly support *normalization*.

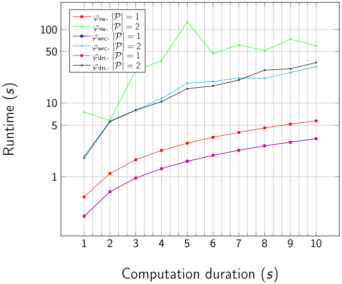
$$\text{Student}(\mathit{id}, \mathit{name}) \qquad\qquad \text{Enrollment}(\mathit{id}, \mathit{course}).$$

$$\varphi_{\mathsf{wrc}} = \neg\Big(\neg \mathit{write}(\text{Student}.\mathit{id})\ \mathcal{U}\ \mathit{write}(\text{Enrollment}.\mathit{id})\Big)$$

$$\varphi_{\mathsf{drc}} = \neg\Big(\neg \mathit{delete}(\text{Enrollment}.\mathit{id})\ \mathcal{U}\ \mathit{delete}(\text{Student}.\mathit{id})\Big)$$

Cassandra Experiments



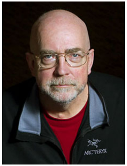


$l = 20s \qquad r = 100/s$

Outline of talk

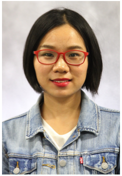
- **Distributed RV of MetricTemporal Properties for Cross-Chain Protocols** (ICDCS'22)

Maurice Herlihy



Ritam Ganguly

Yingjie Xue



Aaron Jonckheere

Parker Ljung

Benjamin Schornstein

Vulnerabilities in Blockchain Transactions

- Cryptocurrency is a 2.2 trillion US dollar market
- Smart contract is a program running on the blockchain which gets triggered automatically. In this way, the transfer of assets can be automated by the rules in the smart contracts, and human intervention cannot stop it.

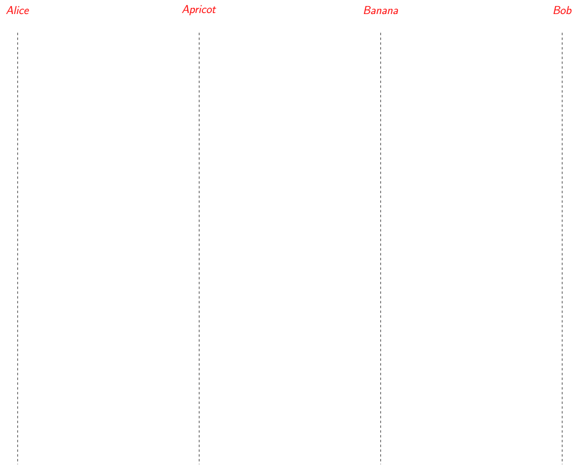
[2] https://github.com/openethereum/parity-ethereum

Vulnerabilities in Blockchain Transactions

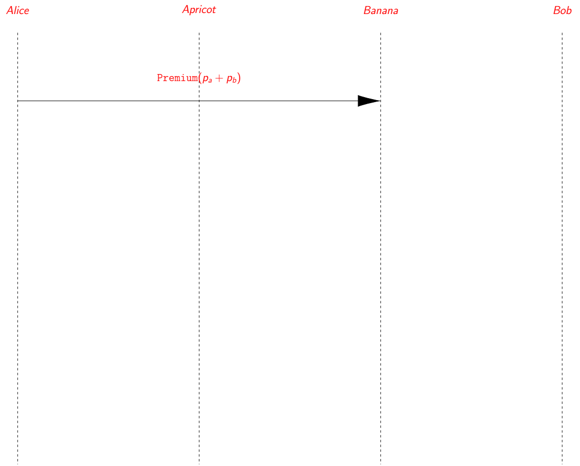
- Cryptocurrency is a 2.2 trillion US dollar market
- Smart contract is a program running on the blockchain which gets triggered automatically. In this way, the transfer of assets can be automated by the rules in the smart contracts, and human intervention cannot stop it.
- If the smart contract has bugs and does not do what is expected, then lack of human intervention may lead to massive financial losses.
- Parity Multisig Wallet smart contract [2] version 1.5 included a vulnerability which led to the loss of 30 million US dollars.

[2] https://github.com/openethereum/parity-ethereum

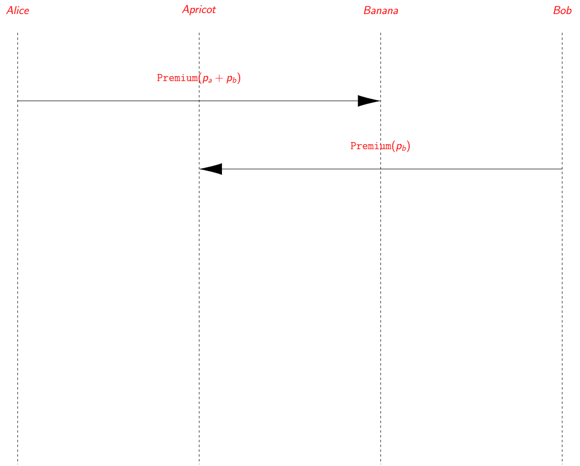
Cross Chain Transactions



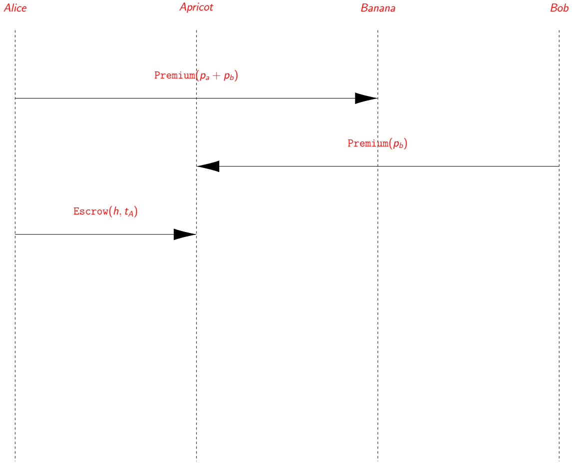
Cross Chain Transactions



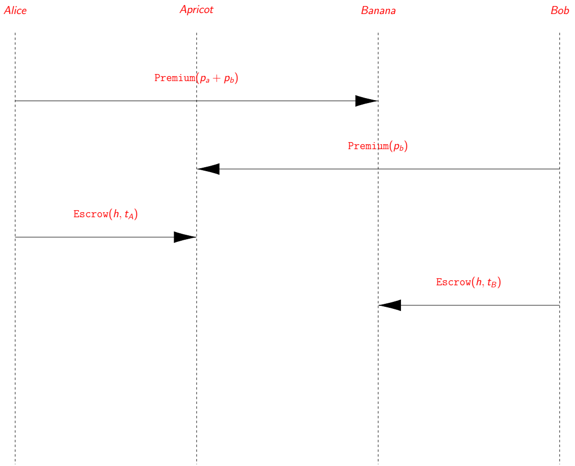
Cross Chain Transactions



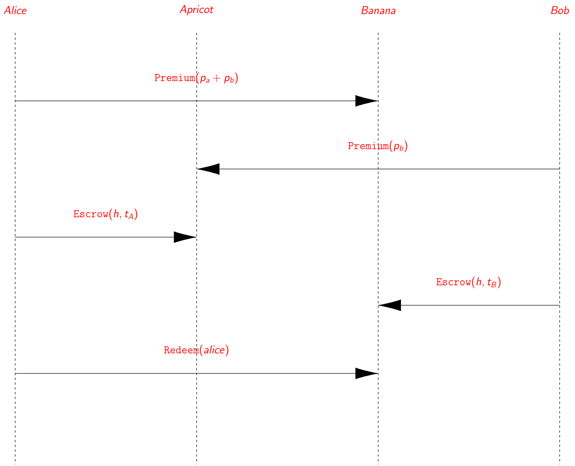
Cross Chain Transactions



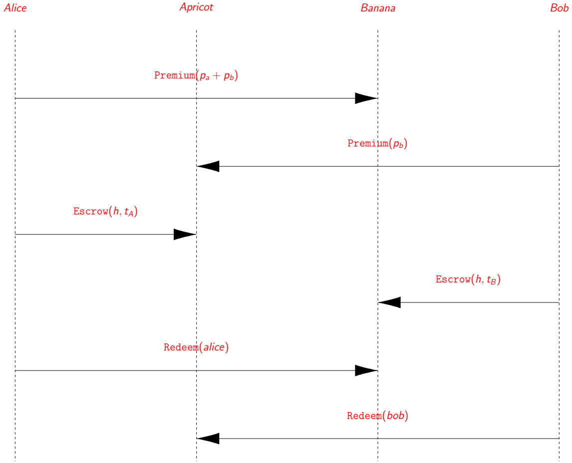
Cross Chain Transactions



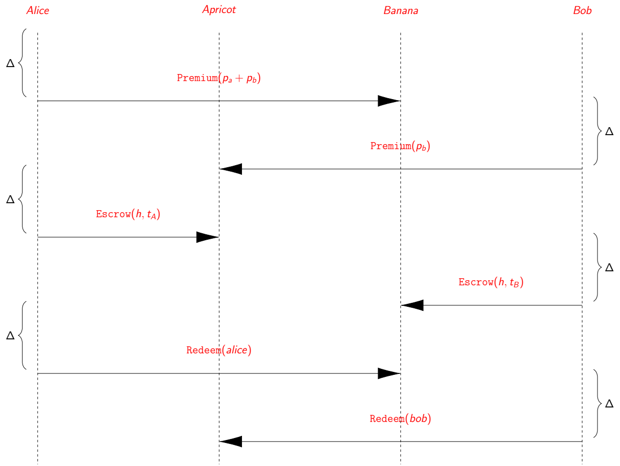
Cross Chain Transactions



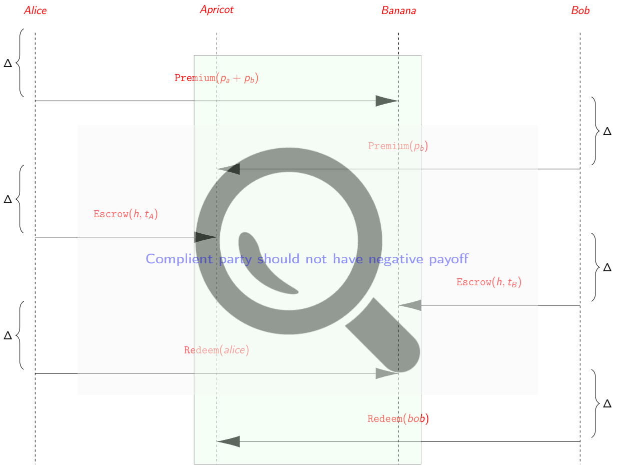
Cross Chain Transactions



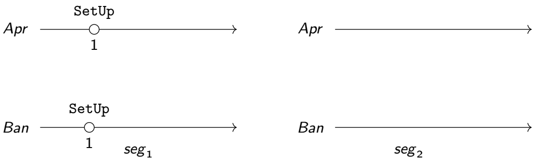
Cross Chain Transactions



Cross Chain Transactions

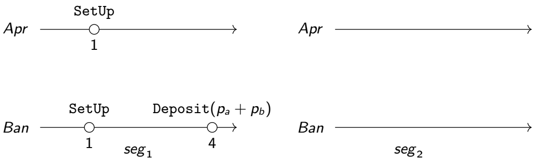


Overview of our Solution



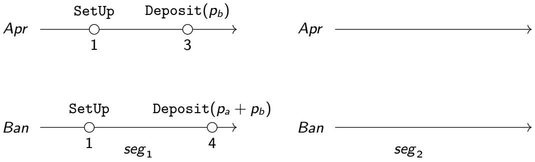
- $\varphi_{spec} = \neg\texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)}\texttt{Ban.Redeem}(alice)$

Overview of our Solution



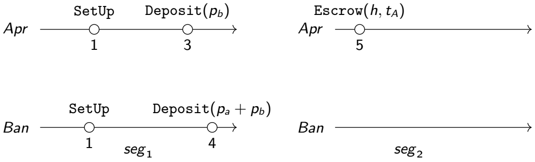
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



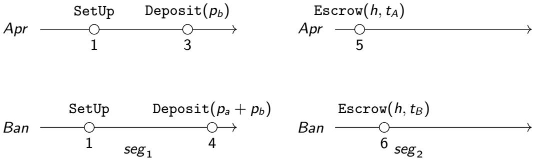
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



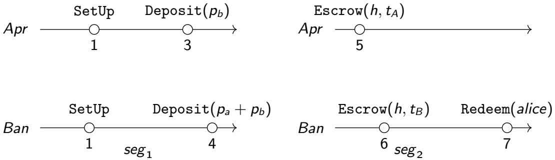
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



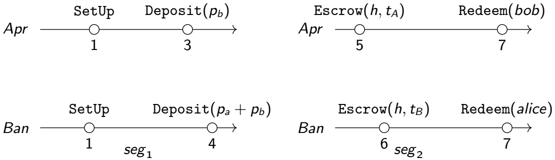
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



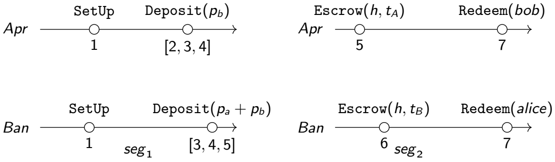
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



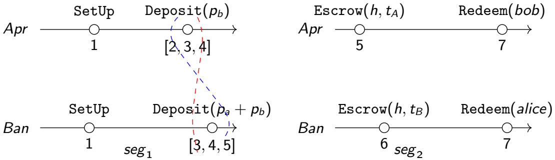
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



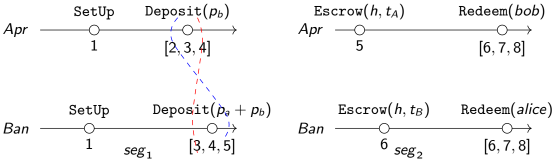
- $\varphi_{spec} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



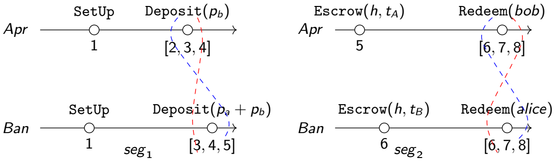
- $\varphi_{spec} = \neg\texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,8)}\texttt{Ban.Redeem}(alice)$
- $\varphi_{spec\_1} = \neg\texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,5)}\texttt{Ban.Redeem}(alice)$
- $\varphi_{spec\_2} = \neg\texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,4)}\texttt{Ban.Redeem}(alice)$
- $\varphi_{spec\_3} = \neg\texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,3)}\texttt{Ban.Redeem}(alice)$

Overview of our Solution



- $\varphi_{spec\_1} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,5)} \texttt{Ban.Redeem}(alice)$
- $\varphi_{spec\_2} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,4)} \texttt{Ban.Redeem}(alice)$
- $\varphi_{spec\_3} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,3)} \texttt{Ban.Redeem}(alice)$

Overview of our Solution



- $\varphi_{spec\_1} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,5)} \texttt{Ban.Redeem}(alice) = \texttt{true}$
- $\varphi_{spec\_2} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,4)} \texttt{Ban.Redeem}(alice) = \texttt{true}$
- $\varphi_{spec\_3} = \neg \texttt{Apr.Redeem}(bob)\ \mathcal{U}_{[0,3)} \texttt{Ban.Redeem}(alice) = \texttt{false}$

Blockchain Transactions

- We implemented two-party swap, multi-party swap, and auction[3].
- The protocols were written as smart contracts in `Solidity` and tested using `Ganache`, a tool that creates mocked `Ethereum` blockchains.

[3] Y. Xue and M. Herlihy, "Hedging against sore loser attacks in cross-chain transactions

Blockchain Transactions

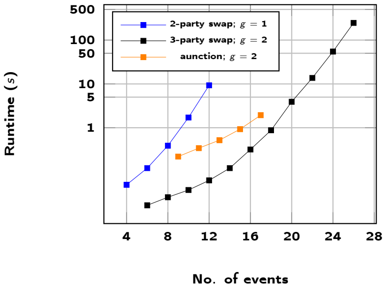
- We implemented two-party swap, multi-party swap, and auction[3].
- The protocols were written as smart contracts in Solidity and tested using Ganache, a tool that creates mocked Ethereum blockchains.
- We check the policies for liveness, safety, and ability to hedge against sore loser attacks.

$$
\begin{aligned}
\varphi_{\text{alice\_conform}} = {} & \Diamond_{[0,\Delta)} \texttt{ban.premium\_deposited(alice)} \wedge \\
& \big(\Diamond_{[0,2\Delta)} \texttt{apr.premium\_deposited(bob)} \rightarrow \\
& \quad \Diamond_{[0,3\Delta)} \texttt{apr.asset\_escrowed(alice)}\big) \wedge \\
& \big(\Diamond_{[0,4\Delta)} \texttt{ban.asset\_escrowed(bob)} \rightarrow \\
& \quad \Diamond_{[0,5\Delta)} \texttt{ban.asset\_redeemed(alice)}\big) \wedge \\
& \big(\neg \texttt{apr.asset\_redeemed(bob)}\, \mathcal{U} \\
& \quad \texttt{ban.asset\_redeemed(alice)}\big)
\end{aligned}
$$

[3] Y. Xue and M. Herlihy, "Hedging against sore loser attacks in cross-chain transactions

Blockchain Transactions

- We generate transaction logs with different values for deadline (Δ) and time synchronization constant (ϵ)
- We observe both `true` and `false` verdict when $\epsilon \gtrapprox \Delta$



Outline of talk

- **Predicate Monitoring in Distributed Cyber-physical Systems** (RV'21) – Best Paper Award
- **Predicate Monitoring in Distributed Cyber-physical Systems** (STTT'23)
- **Monitoring Signal Temporal Logic in Distributed Cyber-physical Systems** (ICCPS'23)

Houssam abbas

Anik Momtaz

Signals

- A *signal* (of some agent A) is a function $x : [a, b] \to \Re^d$, which is right-continuous, left-limited, and is not Zeno.

Signals

- A *signal* (of some agent A) is a function $x : [a, b] \rightarrow \Re^d$, which is right-continuous, left-limited, and is not Zeno.
- A *distributed signal* is a set of signals that do not share a common clock.

Signals

- A *signal* (of some agent A) is a function $x : [a, b] \rightarrow \Re^d$, which is right-continuous, left-limited, and is not Zeno.
- A *distributed signal* is a set of signals that do not share a common clock.

Signal Retiming

A *retiming* function, or simply retiming, is an increasing function $\rho : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$.

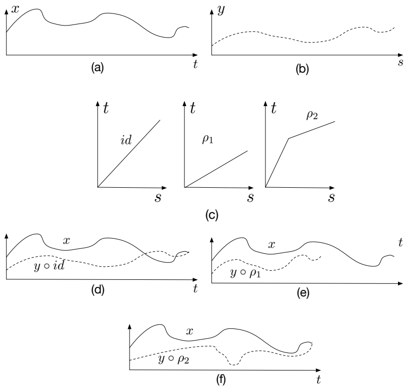
Signals

- A *signal* (of some agent A) is a function $x : [a, b] \to \Re^d$, which is right-continuous, left-limited, and is not Zeno.
- A *distributed signal* is a set of signals that do not share a common clock.

Signal Retiming

A *retiming* function, or simply retiming, is an increasing function $\rho : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$.

- An *ε-retiming* is a retiming function such that: $\forall t \in \mathbb{R}_{\geq 0} : |t - \rho(t)| < \varepsilon$. Given a distributed signal $(E, \rightsquigarrow)$ over N agents and any two distinct agents A_i, A_j, where $i, j \in [N]$, a retiming ρ from A_j to A_i is said to *respect* $\rightsquigarrow$ if we have $(e^i_t \rightsquigarrow e^j_{t'}) \Rightarrow (t < \rho(t'))$ for any two events $e^i_t, e^j_{t'} \in E$.



Retiming Functions

- *Proposition 1.* Given an STL formula φ and distributed signals $(E, \rightsquigarrow)$ over N agent, there exists a consistent cut $C \subseteq E$ that violates φ *if and only if* there exists a finite A_1-local clock value t and $N-1$ ε-retimings $\rho_n : I_n \to I_1$ that respect $\rightsquigarrow$, $2 \leq n \leq N$, such that:

$$\varphi\Big(x_1(t), x_2 \circ \rho_2^{-1}(t), \ldots, x_N \circ \rho_N^{-1}(t)\Big) = \texttt{false} \tag{1}$$

and such that $\rho_m^{-1} \circ \rho_n : I_n \to I_n$ is an ε-retiming for all $n \neq m$. Here, '$\circ$' denotes the function composition operator.

Problem Statement

Problem Statement

Given $\varepsilon > 0$, a distributed signal $(E, \rightsquigarrow)$ over N agents, and a formula φ over the N agents, find $N-1$ ε-retiming functions $\rho_2, \ldots, \rho_N$ that satisfy the hypotheses of Prop. 1 and s.t.

$$\varphi\Big(x_1(t_1), x_2(t_2), \ldots, x_N(t_N)\Big) = \texttt{false} \tag{2}$$

Problem Statement

Problem Statement

Given $\varepsilon > 0$, a distributed signal $(E, \rightsquigarrow)$ over N agents, and a formula φ over the N agents, find $N-1$ ε-retiming functions $\rho_2, \ldots, \rho_N$ that satisfy the hypotheses of Prop. 1 and s.t.

$$\varphi\Big(x_1(t_1), x_2(t_2), \ldots, x_N(t_N)\Big) = \texttt{false} \tag{2}$$

Solution: Transformation to SMT solving using *uninterpreted real functions* to find a violating retiming.

Monitoring Real Distributed CPS

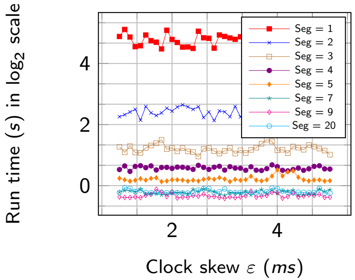


Figure: Effect of clock skew ϵ in a network of cars.

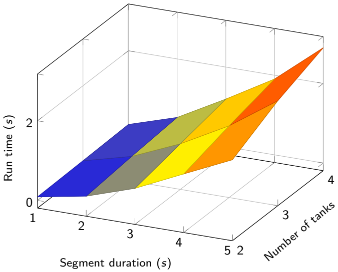
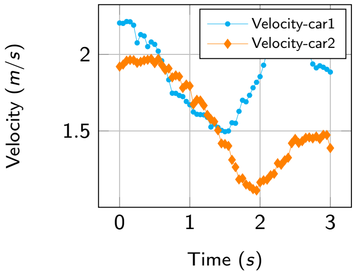
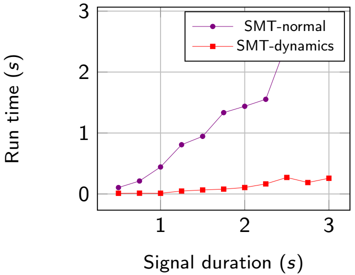


Figure: Monitoring water distribution.

Exploiting Knowledge of System Dynamics



(a) Velocity profile of two cars.



(b) Run time vs. signal duration.

$$\varphi = (v_1 > 1.6) \vee (v_2 > 1.3)$$

Knowledge of acceleration bounds

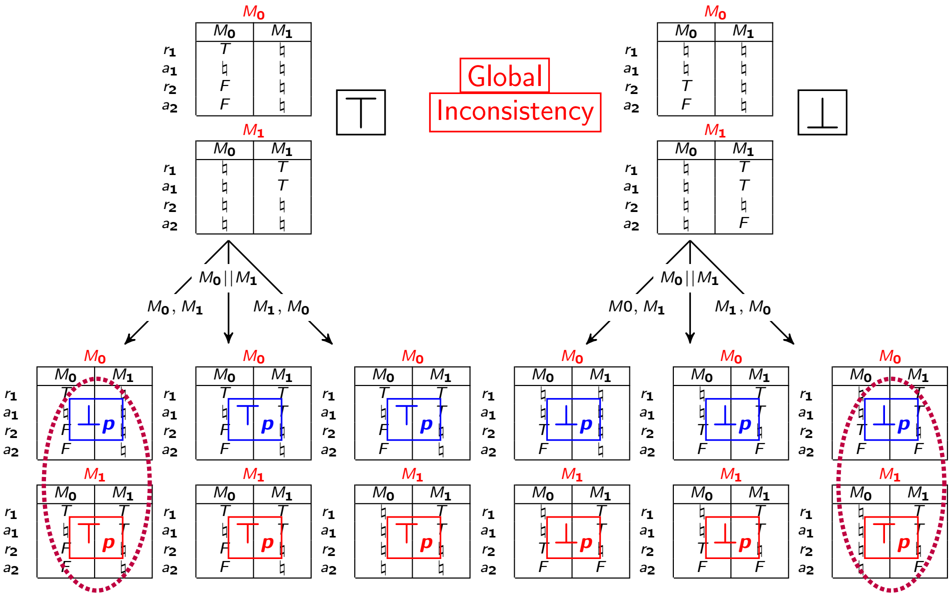
Outline of talk

- **Decentralized Asynchronous Crash-Resilient Runtime Verification** (CONCUR'16)
- **Decentralized Asynchronous Crash-Resilient Runtime Verification** (JACM'22) – Among 8 selected papers in 2022

Sergio Rajsbaum

Pierre Fraigniaud

Corentin Travsers


Global
Inconsistency
M0||M1
M0, M1
M1, M0
M0||M1
M0, M1
M1, M0

General Lower bound Results

Lemma

Not all LTL formulas can be consistently monitored by a distributed monitor with 4 truth values, even if monitors satisfy state coverage, and even if no monitor crashes.

Theorem

Not all LTL formulas can be consistently monitored by a distributed monitor with 4 truth values, even if monitors satisfy state coverage, even if no monitor crashes and even if the monitors perform an arbitrarily large number of rounds.

Alternation Number

Idea

We count the number of times that the valuation of a formula may change from (called *alternation number*).

Alternation Number

Idea

We count the number of times that the valuation of a formula may change from (called *alternation number*).

$\Box p$

Alternation Number

Idea

We count the number of times that the valuation of a formula may change from (called *alternation number*).

Alternation Number

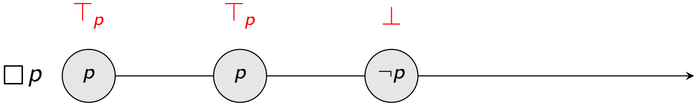
Idea

We count the number of times that the valuation of a formula may change from (called *alternation number*).

Alternation Number

Idea

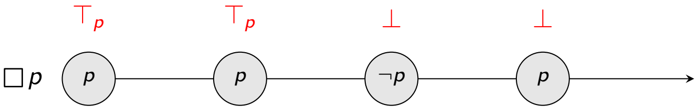
We count the number of times that the valuation of a formula may change from (called *alternation number*).



Alternation Number

Idea

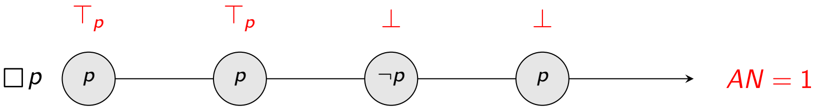
We count the number of times that the valuation of a formula may change from (called *alternation number*).



Alternation Number

Idea

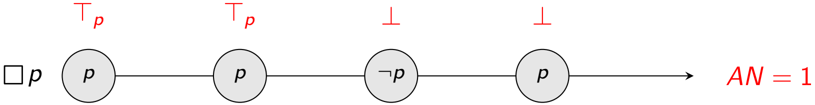
We count the number of times that the valuation of a formula may change from (called *alternation number*).



Alternation Number

Idea

We count the number of times that the valuation of a formula may change from (called *alternation number*).



Alternation number

The *alternation number* of an LTL formula φ is the following:

$$AN(\varphi) = \max\left\{A(w) \mid w \in \Sigma^*\right\}$$

where

$$A(w) = \begin{cases} A(w') + 1 & \text{if} \quad [w \models_F \varphi] \neq [w' \models_F \varphi] \\ 0 & \text{if} \quad \mathit{length}(w) = 1 \end{cases}$$

where w' denotes the longest proper prefix of w. ■

Obtaining Alternation Number

Theorem

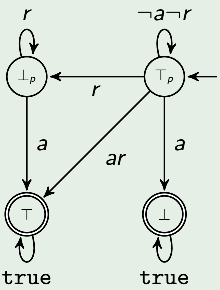
The alternation number of LTL formula φ is the length of the *longest alternating walk* of the LTL_4 monitor of φ.

Obtaining Alternation Number

Theorem

The alternation number of LTL formula φ is the length of the *longest alternating walk* of the LTL_4 monitor of φ.

Example



$$AN(\Box(\neg a \neg r) \vee [(\neg a\ \mathcal{U}\ r) \wedge \Diamond a]) = 2$$

Global Consistency

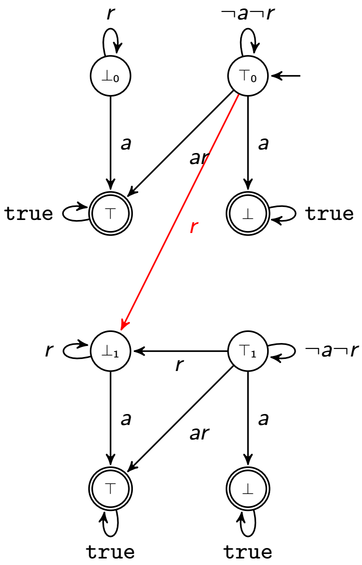
Lower Bound Theorem

In order to monitor an LTL formula φ by a wait-free distributed monitor, we need *at least* $AN(\varphi) + 1$ truth values to ensure global consistency.

Upper Bound Theorem

An LTL formula can consistently be monitored by a wait-free distributed monitor in LTL_{2k+4}, if $k \geq [\frac{1}{2}(\min(AN(\varphi), n) - 1)]$.

LTL_k Monitor Construction



Monitor for

$\square(\neg a \neg r) \vee [(\neg a \, \mathcal{U} \, r) \wedge \Diamond a]$

in LTL_6.

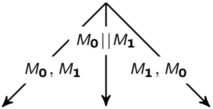
M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

$\boxed{\top}$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$



M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1

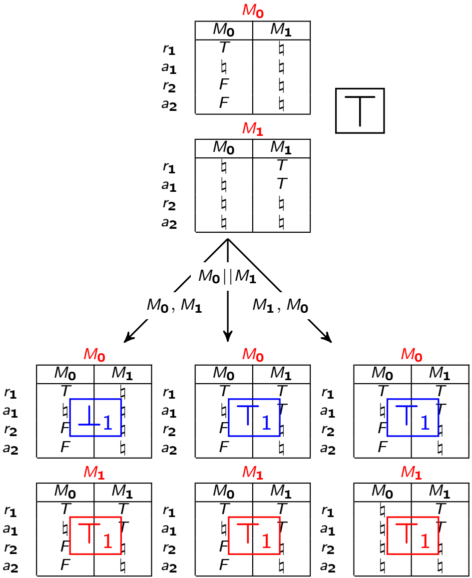
	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$


M0
M1
r1
a1
r2
a2
T
F
♮
⊤
M0 || M1
M0 , M1
M1 , M0
⊥ 1
⊤ 1

M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

$\top$

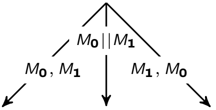
M_0

	M_0	M_1
r_1	$\natural$	$\natural$
a_1	$\natural$	$\natural$
r_2	T	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	F

$\bot$



M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

$\bot_1$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

$\top_1$

M_0

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

$\top_1$

M_1

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

$\top_1$

M_0

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

$\top_1$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

$\top_1$

Left

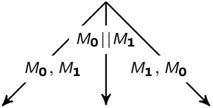
M_0

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

$\top$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$



M_0 ($\bot_1$)

	M_0	M_1
r_1	T	$\natural$
a_1	$\natural$	$\natural$
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top_1$)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0 ($\top_1$)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top_1$)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_0 ($\top_1$)

	M_0	M_1
r_1	T	T
a_1	$\natural$	T
r_2	F	$\natural$
a_2	F	$\natural$

M_1 ($\top_1$)

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	$\natural$

Right

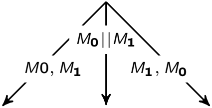
M_0

	M_0	M_1
r_1	$\natural$	$\natural$
a_1	$\natural$	$\natural$
r_2	T	$\natural$
a_2	F	$\natural$

$\bot$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	F



M_0

	M_0	M_1
r_1	$\natural$	$\natural$
a_1	$\natural$	$\natural$
r_2	T	$\natural$
a_2	F	$\natural$

M_1

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	T	$\natural$
a_2	F	F

M_0

	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	T	$\natural$
a_2	F	F

M_1

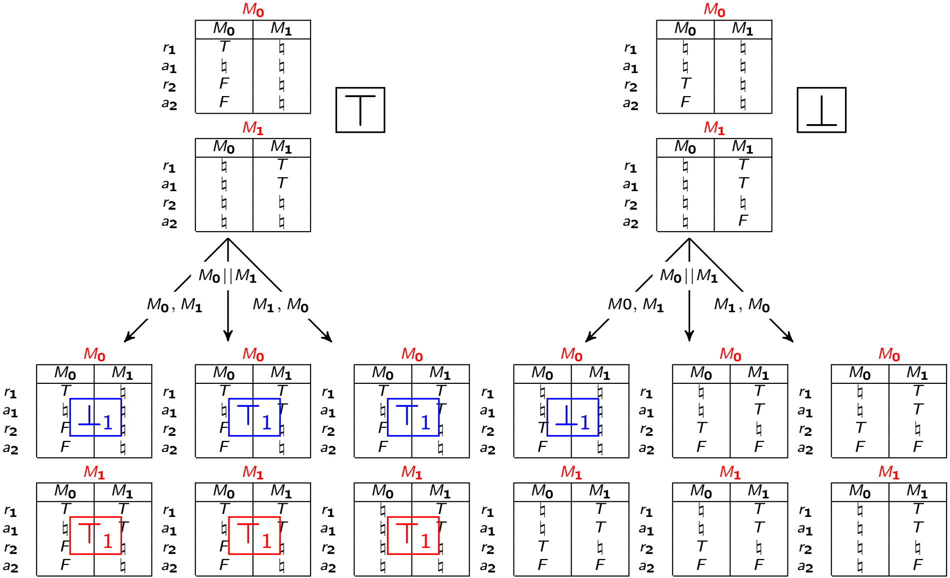
	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	T	$\natural$
a_2	F	F

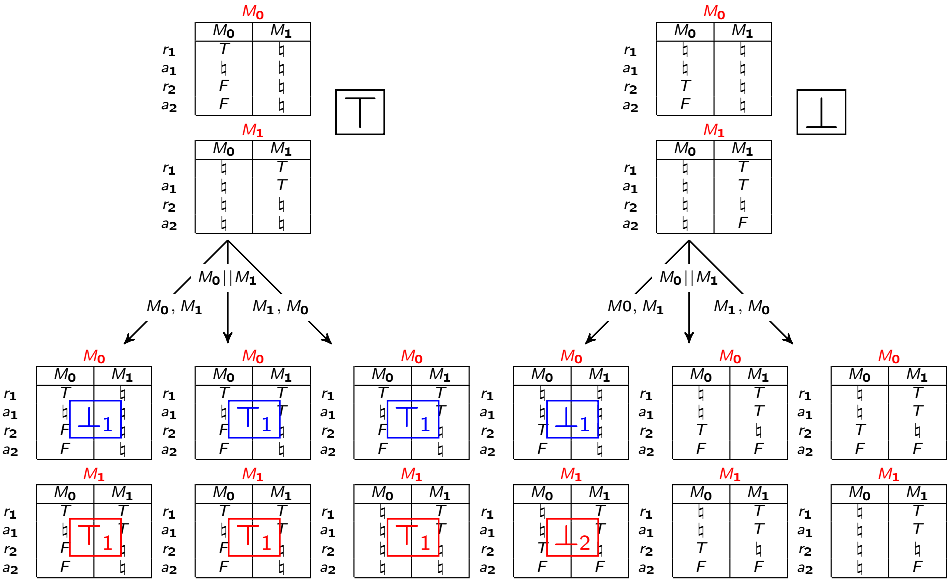
M_0

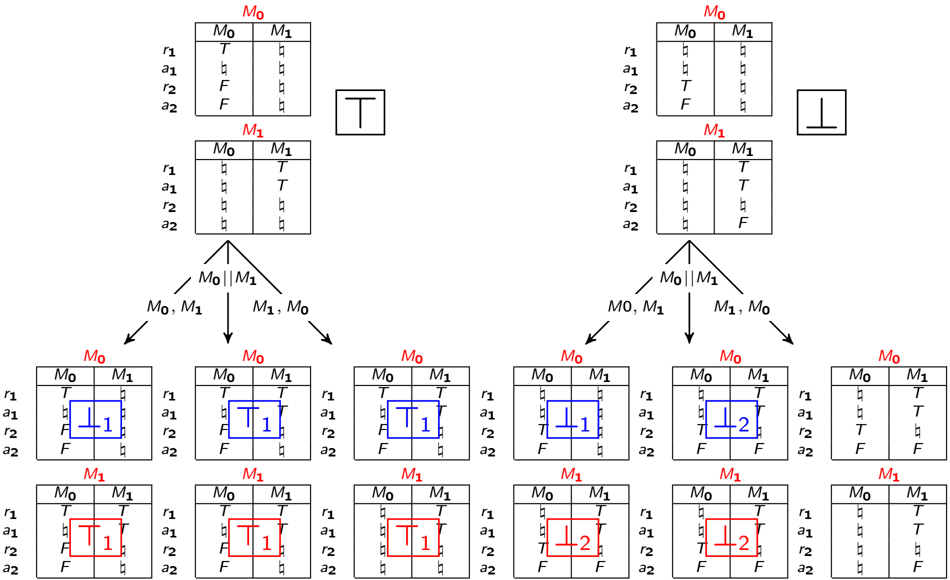
	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	T	$\natural$
a_2	F	F

M_1

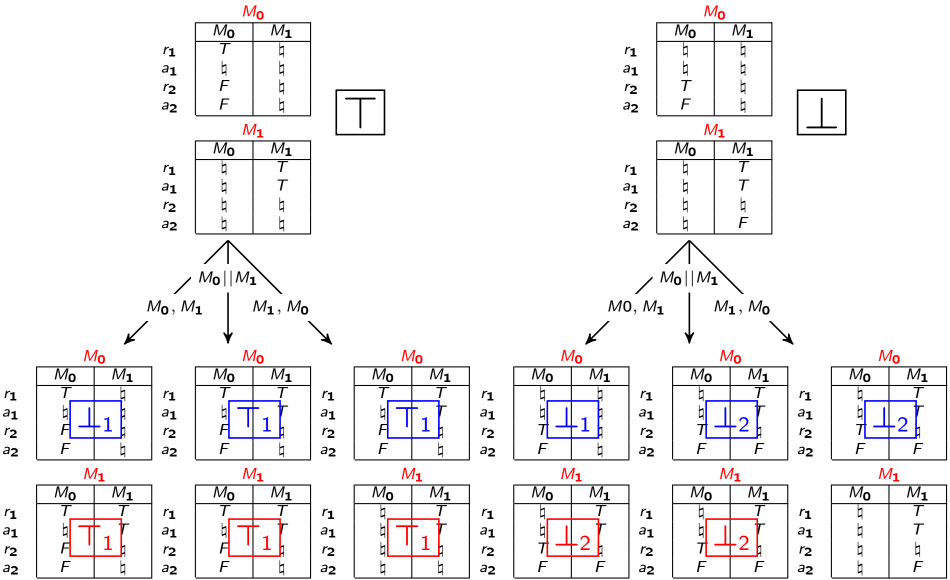
	M_0	M_1
r_1	$\natural$	T
a_1	$\natural$	T
r_2	$\natural$	$\natural$
a_2	$\natural$	F

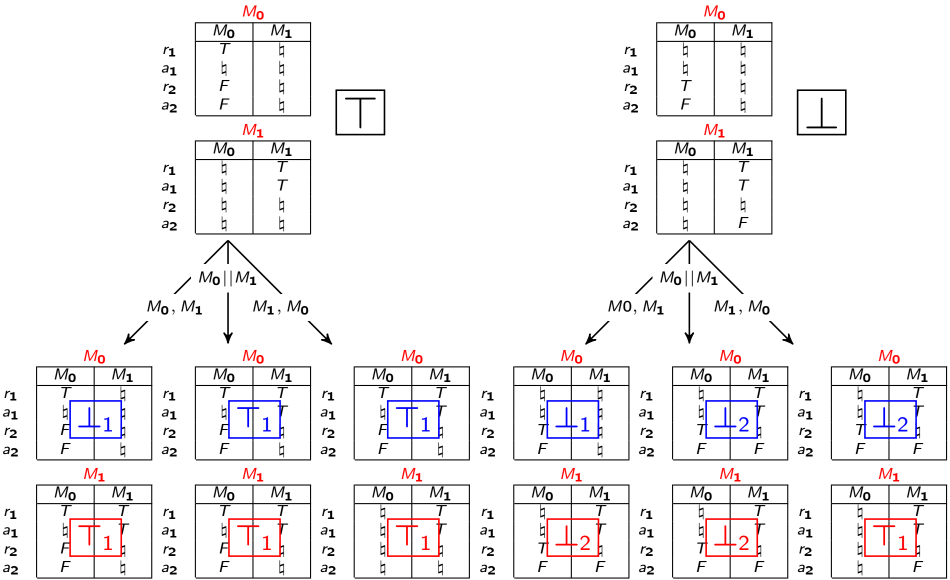

M_0
M_1
r_1
a_1
r_2
a_2
$\top$
$\bot$
$M_0 || M_1$
M_0, M_1
M_1, M_0
$M0, M_1$
$\bot_1$
$\top_1$

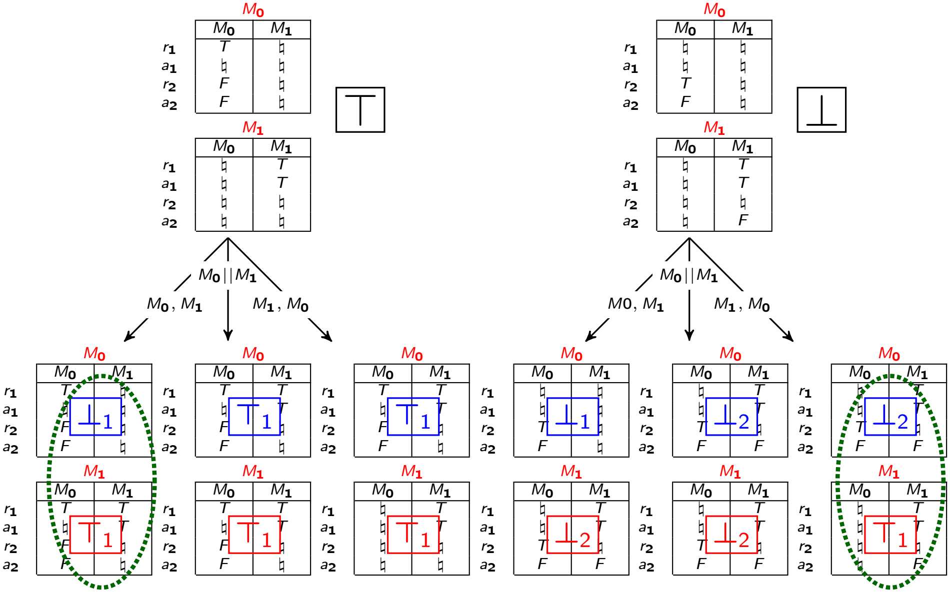

M0
M1
r1
a1
r2
a2
T
F
⊤
⊥
M0||M1
M0, M1
M1, M0
M0, M1
⊥1
⊤1
⊥2

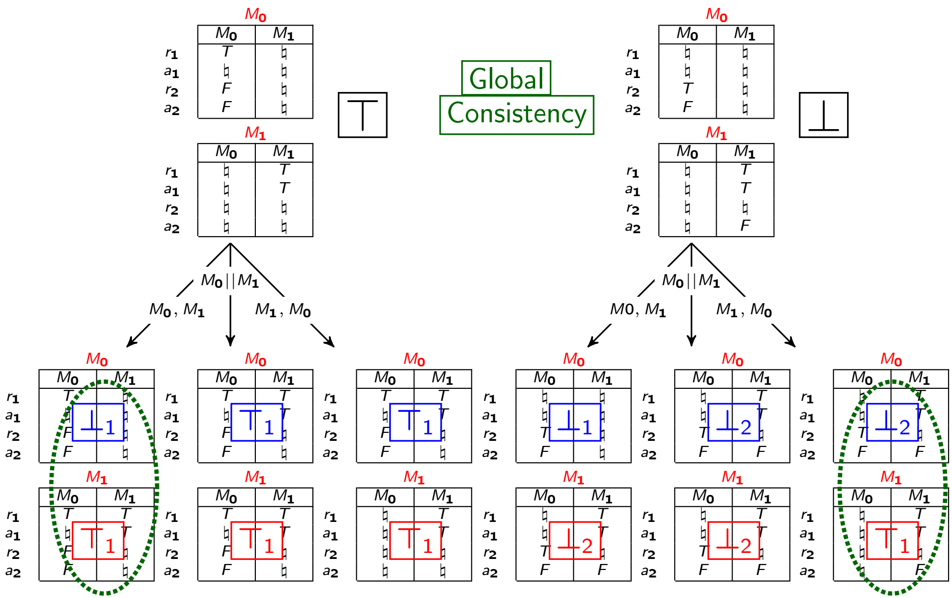


M_0
M_1
r_1
a_1
r_2
a_2
T
F
♮
⊤
⊥
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
M0, M_1
⊤1
⊥1
⊥2


M0
M1
r1
a1
r2
a2
T
F
⊤
⊥
M0 || M1
M0 , M1
M1 , M0
M0, M1
⊥1
⊤1
⊥2


M0||M1
M0, M1
M1, M0
M0, M1
M1, M0
⊤
⊥
⊥1
⊤1
⊥2
r1
a1
r2
a2


$M_0 || M_1$
M_0, M_1
M_1, M_0
$M0, M_1$
M_0
M_1
r_1
a_1
r_2
a_2
$\top$
$\bot$
$\top$ 1
$\bot$ 1
$\bot$ 2



Global
Consistency
M_0
M_1
r_1
a_1
r_2
a_2
T
F
⊤
⊥
$M_0 || M_1$
M_0 , M_1
M_1 , M_0
$M0, M_1$
⊥ 1
⊤ 1
⊥ 2

Outline of talk

Summary

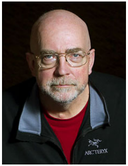
- Distributed RV under *partial synchrony*.
- *SMT*-based solution.
- *Multicore* optimization
- Monitoring *blockchains*
- Distributed RV for *analog* signals.
- *Crash-resilient* RV.

Ongoing Work

- Trade-off between *accuracy* and scalability.
  - Over/under-approximation
- *Byzantine* distributed RV.
- *Stream-based* (I/O) distributed RV for network of DNNs.
- *Private* distributed RV

Acknowledgment

Maurice Herlihy



Sergio Rajsbaum

Pierre Fraigniaud

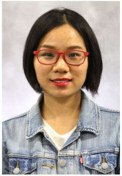
Corentin Travsers

Houssam Abbas

Ritam Ganguly

Anik Momtaz

Yingjie Xue



Commercials!

- I am looking for *PhD students* to work on:
  - Runtime monitoring
  - Information-flow security
  - Causality
- CSE@MSU has four *open faculty positions* in all areas of computer science.
- Email me: *borzoo@msu.edu*

Thank You!