CLOTHO: DIRECTED TEST GENERATION FOR WEAKLY CONSISTENT DATABASE SYSTEMS
Kia Rahmani
Kartik Nagar
Benjamin Delaware
Suresh Jagannathan
Transactional support

Database Program

TXN  TXN  TXN

TXN  TXN
TRADITIONAL DATABASE PROGRAMMING

- Transactional support
- Highly structured relational data
- Transactional support
- Highly structured relational data
- Clients invoke transactions
- Transactional support
- Highly structured relational data
- Clients invoke transactions
- Structured query language for data retrieval/modification
TRADITIONAL DATABASE PROGRAMMING

- Transactional support
- Highly structured relational data
- Clients invoke transactions
- Structured query language for data retrieval/modification
- Queries processed and responded by the DBMS
ACID guarantees
- ACID guarantees
  - Atomicity
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
ACID guarantees
  ✓ Atomicity
TRADITIONAL DATABASE PROGRAMMING

- **ACID guarantees**
  - **Atomicity**

"All or None"
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency

“Single Copy of Data”
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation

 execution order
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation

"No Interference"
ACID guarantees

- Atomicity
- Consistency
- Isolation
- Durability
TRADITIONAL DATABASE PROGRAMMING

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

“Permanent Commits”
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Serializability
ACID guarantees

- Atomicity
- Consistency
- Isolation
- Durability

Serialization facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - **Atomicity**
  - **Consistency**
  - **Isolation**
  - **Durability**

- **Serializability** facilitates program design and reasoning

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Execution order**

```
TXN (arg)
SELECT pay_cnt AS v
WHERE id=arg
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
TXN (arg)
SELECT pay_cnt AS v
WHERE id=arg
UPDATE pay_cnt=v+1
WHERE id=arg
```
ACID guarantees
- Atomicity
- Consistency
- Isolation
- Durability

Serializability facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```sql
SELECT pay_cnt AS v
WHERE id=arg
UPDATE pay_cnt=v+1
WHERE id=arg
```

**Execution order:**
- SELECT `pay_cnt` AS `v`
- WHERE `id=arg`
- UPDATE `pay_cnt`=v+1
- WHERE `id=arg`

```
TXN (arg)
```

```
TXN (arg)
```
ACID guarantees
- Atomicity
- Consistency
- Isolation
- Durability

Serializability facilitates program design and reasoning.
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning
ACID guarantees
- Atomicity
- Consistency
- Isolation
- Durability

Serializable facilitates program design and reasoning
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning

```
<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

TXN (arg)
```

```
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TXN (arg)
```

```
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning.

**Example:**

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

TXN (arg)

```
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
SELECT pay_cnt AS v
WHERE id=arg
```

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Final pay_cnt = # of TXN invocation**
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - **Atomicity**
  - **Consistency**
  - **Isolation**
  - **Durability**

- **Serializability** facilitates program design and reasoning

- Requires heavy synchronization
SERIALIZABILITY GUARANTEES

- ACID guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Serializability facilitates program design and reasoning

- Requires heavy synchronization
SERIALIZABILITY GUARANTEES

- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Serializability** facilitates program design and reasoning

- Requires heavy synchronization

  - Unacceptable cost for web-scale applications
ACID guarantees
- Atomicity
- Consistency
- Isolation
- Durability

Serializability facilitates program design and reasoning

Requires heavy synchronization

Weaker guarantees are offered in favor of higher performance

Not Isolated!

Witness each other’s presence!
EXAMPLE: A SERIALIZABILITY ANOMALY
Unexpected behaviors can occur under weak guarantees
Unexpected behaviors can occur under weak guarantees
Unexpected behaviors can occur under weak guarantees.

### Example: A Serializability Anomaly

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TXN (arg)**

```
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```
Unexpected behaviors can occur under weak guarantees
Unexpected behaviors can occur under weak guarantees
### Unexpected behaviors can occur under weak guarantees

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**SQL Execution Order**

1. `SELECT pay_cnt AS v WHERE id=arg`
2. `UPDATE pay_cnt=v+1 WHERE id=arg`
3. `SELECT pay_cnt AS v WHERE id=arg`
EXAMPLE: A SERIALIZABILITY ANOMALY

- Unexpected behaviors can occur under weak guarantees

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```
TXN (arg)
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
SELECT pay_cnt AS v
WHERE id=arg
```

```
UPDATE pay_cnt=v+1
WHERE id=arg
```

execution order
Unexpected behaviors can occur under weak guarantees.
EXAMPLE: A SERIALIZABILITY ANOMALY

- Unexpected behaviors can occur under weak guarantees
- Assumed program invariants can be violated
WEAKLY CONSISTENT REPLICATED DATABASE SYSTEMS
Data is geo-replicated in highly-available DBMSs
WEAKLY CONSISTENT REPLICATED DATABASE SYSTEMS

- Data is geo-replicated in highly-available DBMSs
- Worldwide synchronization is **extremely** costly
WEAKLY CONSISTENT REPLICATED DATABASE SYSTEMS

- Data is geo-replicated in highly-available DBMSs
- Worldwide synchronization is extremely costly
- Strongly synchronized data cannot be available
WEAKLY CONSISTENT REPLICATED DATABASE SYSTEMS

- Data is geo-replicated in highly-available DBMSs
- Worldwide synchronization is extremely costly
- Strongly synchronized data cannot be available
WEAKLY CONSISTENT REPLICATED DATABASE SYSTEMS

- Data is geo-replicated in highly-available DBMSs
- Worldwide synchronization is extremely costly
- Strongly synchronized data cannot be available
- Weak consistency semantics are very popular
Data is geo-replicated in highly-available DBMSs

Worldwide synchronization is extremely costly

Strongly synchronized data cannot be available
WEAKLY CONSISTENT REPLICAED DATABASE SYSTEMS

- Data is geo-replicated in highly-available DBMSs
- Worldwide synchronization is extremely costly
- Strongly synchronized data cannot be available
- Weak consistency semantics are very popular
- Serializability is rarely assumed by default [Bailis et.al]

<table>
<thead>
<tr>
<th>Database</th>
<th>Default</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actian Ingres 10.0/10S [1]</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Aerospike [2]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>Clustrix CLX 4100 [4]</td>
<td>RR</td>
<td>RR</td>
</tr>
<tr>
<td>Greenplum 4.1 [8]</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>IBM DB2 10 for z/OS [5]</td>
<td>CS</td>
<td>S</td>
</tr>
<tr>
<td>IBM Informix 11.50 [9]</td>
<td>Depends</td>
<td>S</td>
</tr>
<tr>
<td>MySQL 5.6 [12]</td>
<td>RR</td>
<td>S</td>
</tr>
<tr>
<td>MemeSQL 1b [10]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>Oracle 11g [14]</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>Oracle Berkeley DB [7]</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Postgres 9.2.2 [15]</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>SAP HANA [16]</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>ScaleDB 1.02 [17]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>VoltDB [18]</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

TESTING: FUNDAMENTAL CHALLENGES
Triggering anomalies requires determining many parameters.
Triggering anomalies requires determining many parameters.
Testing: Fundamental Challenges

- Triggering anomalies requires determining many parameters
- Initial database state

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```
SELECT pay_cnt AS v
WHERE id=arg
UPDATE pay_cnt=v+1
WHERE id=arg
```
Triggering anomalies requires determining many parameters

- Initial database state
- Input arguments
Triggering anomalies requires determining many parameters:
- Initial database state
- Input arguments
- Execution order

TESTING: FUNDAMENTAL CHALLENGES

```
<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

SELECT pay_cnt AS v
WHERE id=arg
TXN (arg)  //arg=1
UPDATE pay_cnt=v+1
WHERE id=arg
```

```
SELECT pay_cnt AS v
WHERE id=arg
TXN (arg)  //arg=1
UPDATE pay_cnt=v+1
WHERE id=arg
```
Triggering anomalies requires determining many parameters:
- Initial database state
- Input arguments
- Execution order
- Network delays

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```
TXN (arg) //arg=1
SELECT pay_cnt AS v
WHERE id=arg
UPDATE pay_cnt=v+1
WHERE id=arg
```

---

**TESTING: FUNDAMENTAL CHALLENGES**
Triggering anomalies requires determining many parameters

- Initial database state
- Input arguments
- Execution order
- Network delays

**TABLE**

<table>
<thead>
<tr>
<th>id</th>
<th>pay_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**SQL QUERY**

```sql
SELECT pay_cnt AS v
WHERE id=arg
```

**UPDATE**

```
UPDATE pay_cnt=v+1
WHERE id=arg
```
Triggering anomalies requires determining many parameters:
- Initial database state
- Input arguments
- Execution order
- Network delays

Exponential state space!
BLACKBOX TESTING
BLACKBOX TESTING

- Independent of application semantics
- Independent of application semantics
- Independent of database specific guarantees
BLACKBOX TESTING

- Independent of application semantics
- Independent of database specific guarantees
- Not reproducible
- Independent of application semantics
- Independent of database specific guarantees
- Not reproducible
- Each database may offer multiple guarantees
- Independent of application semantics
- Independent of database specific guarantees
- Not reproducible
- Each database may offer multiple guarantees
- Time and resource consuming!
BLACKBOX TESTING

- Independent of application semantics
- Independent of database specific guarantees
- Not reproducible
- Each database may offer multiple guarantees
- Time and resource consuming!
- No guarantees
State of the art cloud-based testing framework using **Jepsen** and **OLTPBench**
BLACKBOX TESTING IN ACTION

- State of the art cloud-based testing framework using *Jepsen* and *OLTPBench*

- TPC-C benchmark
BLACKBOX TESTING IN ACTION

- State of the art cloud-based testing framework using *Jepsen* and *OLTPBench*
- TPC-C benchmark
State of the art cloud-based testing framework using Jepsen and OLTPBench

TPC-C benchmark
State of the art cloud-based testing framework using *Jepsen* and *OLTPBench*

- TPC-C benchmark
- 21 application-level invariants were analyzed
BLACKBOX TESTING IN ACTION

- State of the art cloud-based testing framework using *Jepsen* and *OLTPBench*
- TPC-C benchmark
- 21 application-level invariants were analyzed
- Only **14 out of 21** invariants were broken at best

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Broken?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
</tr>
</tbody>
</table>

33% of invariants are assumed to be preserved
WHITE-BOX ANALYSIS
Systematic assessment of anomalous executions **within** a given program
Systematic assessment of anomalous executions **within** a given program

SELECT pay_cnt AS v
WHERE id=arg

UPDATE pay_cnt=v+1
WHERE id=arg

TXN (arg)

SELECT pay_cnt AS v
WHERE id=arg

UPDATE pay_cnt=v+1
WHERE id=arg

TXN (arg)
Systematic assessment of anomalous executions within a given program

Data dependencies among database operations
Systematic assessment of anomalous executions within a given program

Data dependencies among database operations

Execution properties (e.g. order) affect dependent operations
Systematic assessment of anomalous executions within a given program

Data dependencies among database operations

Execution properties (e.g. order) affect dependent operations

WHITE-BOX ANALYSIS

SELECT pay_cnt AS v
WHERE id=arg

UPDATE pay_cnt=v+1
WHERE id=arg

TXN (arg)

SELECT pay_cnt AS v
WHERE id=arg

UPDATE pay_cnt=v+1
WHERE id=arg

Witnesses the update

TXN (arg)
Systematic assessment of anomalous executions within a given program

Data dependencies among database operations

Execution properties (e.g. order) affect dependent operations

Cyclic dependencies between transactions correspond to anomalous executions

\[
\text{SELECT pay\_cnt AS v} \\
\text{WHERE id=arg} \\
\text{TXN (arg)} \\
\text{UPDATE pay\_cnt=v+1} \\
\text{WHERE id=arg}
\]

Does **NOT** witness the update

**WHITE-BOX ANALYSIS**

Does **NOT** witness the update
- Systematic assessment of anomalous executions within a given program
- Data dependencies among database operations
- Execution properties (e.g. order) affect dependent operations
- Cyclic dependencies between transactions correspond to anomalous executions

**Goal:** statically construct valid execution and database instances with cyclic dependencies
FORMAL EXECUTION MODEL
Transactions are arbitrarily invoked
Transactions are arbitrarily invoked

<table>
<thead>
<tr>
<th>TXN</th>
<th>TXN</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op’1</td>
</tr>
<tr>
<td>op2</td>
<td>op’2</td>
</tr>
</tbody>
</table>
Formal Execution Model

- Transactions are arbitrarily invoked
- An **Operation** from an arbitrary transaction is executed at a random partition
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random **partition**
• Transactions are arbitrarily invoked

• An **Operation** from an arbitrary transaction is executed at a random **partition**
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random partition
- Transactions are arbitrarily invoked
- An **Operation** from an arbitrary transaction is executed at a random **partition**
- Operations create a set of read and write **effects** upon execution in the partition
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random **partition**

Operations create a set of read and write **effects** upon execution in the partition

A relations on the set of effects
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random **partition**

Operations create a set of read and write **effects** upon execution in the partition

A relations on the set of effects

**visibility:** *causal* precedence between effects
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random partition

Operations create a set of read and write **effects** upon execution in the partition

A relations on the set of effects
  - **visibility**: *causal* precedence between effects
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random partition

Operations create a set of read and write effects upon execution in the partition

A relations on the set of effects

- **visibility**: causal precedence between effects
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random **partition**

Operations create a set of read and write **effects** upon execution in the partition

A relation on the set of effects
- **visibility**: *causal* precedence between effects
Transactions are arbitrarily invoked

An **Operation** from an arbitrary transaction is executed at a random **partition**

Operations create a set of read and write **effects** upon execution in the partition

A relations on the set of effects
   - **visibility**: *causal* precedence between effects
Transactions are arbitrarily invoked.

An **Operation** from an arbitrary transaction is executed at a random **partition**.

Operations create a set of read and write **effects** upon execution in the partition.

A relation on the set of effects:

- **visibility**: *causal* precedence between effects
- Only within a partition!
Operation-level dependencies
  - write dependency (WW)

UPDATE X

UPDATE X
Operation-level dependencies
- write dependency (WW)
- read dependency (WR)
Operation-level dependencies
- write dependency (WW)
- read dependency (WR)
- read anti-dependency (RW)
A language of axiomatic relations encoded as a decidable fragment of first order logic (FOL)
A language of axiomatic relations encoded as a decidable fragment of first order logic (FOL)
Finding bounded anomalies against a database abstraction is reduced to finding satisfying assignments to a formula \( \varphi \)
A language of axiomatic relations encoded in a decidable fragment of first order logic (FOL)

Finding bounded anomalies against a database abstraction is reduced to finding satisfying assignments to a formula

Valid assignments are constrained by five conjuncts

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}\rightarrow} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]
\( \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \)
A set of constraints which must be satisfied by any execution of any program
A set of constraints which must be satisfied by any execution of any program.

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]

UPDATE X=1

SELECT X  //X=0
A set of constraints which must be satisfied by any execution of any program

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]

\[ \text{UPDATE } X=1 \rightarrow \text{SELECT } X \quad //X=0 \]

**WR** induces the same read/written values.
A set of constraints which must be satisfied by **any execution** of **any** program.

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]

**WR** induces the **same** read/written values.
\( \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \)
\( \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \)

- Includes a set of user-defined constraints on records
Includes a set of user-defined constraints on records

- e.g. "all customer records must be older than 21"
Includes a set of user-defined constraints on records
  e.g. "all customer records must be older than 21"
Includes database-specific consistency and isolation constraints
- Includes a set of user-defined constraints on records
  - e.g. "all customer records must be older than 21"
- Includes database-specific consistency and isolation constraints

<table>
<thead>
<tr>
<th>Guarantee</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causal Visibility</td>
<td>$\Psi_{cv} \equiv \forall \eta_1 \eta_2 \eta_3. \text{vis}(\eta_1, \eta_2) \land \text{vis}(\eta_2, \eta_3) \Rightarrow \text{vis}(\eta_1, \eta_3)$</td>
</tr>
<tr>
<td>Causal Consistency</td>
<td>$\Psi_{cc} \equiv \forall \eta_1 \eta_2. \Psi_{cv} \land (\text{st}(\eta_1, \eta_2) \Rightarrow \text{vis}(\eta_1, \eta_2) \lor \text{vis}(\eta_2, \eta_1))$</td>
</tr>
<tr>
<td>Read Committed</td>
<td>$\Psi_{rc} \equiv \forall \eta_1 \eta_2 \eta_3. \text{st}(\eta_1, \eta_2) \land \text{vis}(\eta_1, \eta_3) \Rightarrow \text{vis}(\eta_2, \eta_3)$</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>$\Psi_{rr} \equiv \forall \eta_1 \eta_2 \eta_3. \text{st}(\eta_1, \eta_2) \land \text{vis}(\eta_3, \eta_1) \Rightarrow \text{vis}(\eta_3, \eta_2)$</td>
</tr>
<tr>
<td>Linearizable</td>
<td>$\Psi_{lin} \equiv \text{ar} \subseteq \text{vis}$</td>
</tr>
<tr>
<td>Strictly Serial</td>
<td>$\Psi_{ser} \equiv \Psi_{rc} \land \Psi_{rr} \land \Psi_{lin}$</td>
</tr>
</tbody>
</table>
Includes a set of user-defined constraints on records
  e.g. "all customer records must be older than 21"
Includes database-specific consistency and isolation constraints

<table>
<thead>
<tr>
<th>Guarantee</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causal Visibility</td>
<td>$\Psi_{cv} \equiv \forall \eta_1 \eta_2 \eta_3. \text{vis}(\eta_1, \eta_2) \land \text{vis}(\eta_2, \eta_3) \Rightarrow \text{vis}(\eta_1, \eta_3)$</td>
</tr>
<tr>
<td>Causal Consistency</td>
<td>$\Psi_{cc} \equiv \forall \eta_1 \eta_2. \Psi_{cv} \land (\text{st}(\eta_1, \eta_2) \Rightarrow \text{vis}(\eta_1, \eta_2) \lor \text{vis}(\eta_2, \eta_1))$</td>
</tr>
<tr>
<td>Read Committed</td>
<td>$\Psi_{rc} \equiv \forall \eta_1 \eta_2 \eta_3. \text{st}(\eta_1, \eta_2) \land \text{vis}(\eta_1, \eta_3) \Rightarrow \text{vis}(\eta_2, \eta_3)$</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>$\Psi_{rr} \equiv \forall \eta_1 \eta_2 \eta_3. \text{st}(\eta_1, \eta_2) \land \text{vis}(\eta_3, \eta_1) \Rightarrow \text{vis}(\eta_3, \eta_2)$</td>
</tr>
<tr>
<td>Linearizable</td>
<td>$\Psi_{lin} \equiv \text{ar} \subseteq \text{vis}$</td>
</tr>
<tr>
<td>Strictly Serial</td>
<td>$\Psi_{ser} \equiv \Psi_{rc} \land \Psi_{rr} \land \Psi_{lin}$</td>
</tr>
</tbody>
</table>

Only executions valid for the database abstraction are constructed
\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land (\varphi_{\text{DEP}} \rightarrow \varphi_{\text{DEP}}) \land \varphi_{\text{ANOMALY}} \]
Necessary conditions to establish a dependency relation between two operation instances
Necessary conditions to establish a dependency relation between two operation instances

ϕ ≡ ϕ_{CONTEXT} ∧ ϕ_{DB} ∧ ϕ_{DEP→} ∧ ϕ_{→DEP} ∧ ϕ_{ANOMALY}
Necessary conditions to establish a dependency relation between two operation instances

- There is a mutually accessed record

\[ \varphi \equiv \varphi_{CONTEXT} \land \varphi_{DB} \land \varphi_{DEP} \land \varphi_{ANOMALY} \]
Necessary conditions to establish a valid dependency relation between two operation instances

- There is a mutually accessed record
- Both operations are simultaneously reached by the control flow
**Necessary** conditions to establish a valid dependency relation between two operation instances

- There is a mutually accessed record
- Both operations are simultaneously reached by the control flow
\( \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{DB} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}} \)
Sufficient conditions to establish a dependency relation between two operation instances

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP} \rightarrow} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]
Sufficient conditions to establish a dependency relation between two operation instances

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}} \]
Sufficient conditions to establish a dependency relation between two operation instances

- If there is a mutually accessed record

\[
\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP} \rightarrow} \land \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}}
\]
Sufficient conditions to establish a dependency relation between two operation instances

- If there is a mutually accessed record
- and both operations are reached

\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \land \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}} \]

```
if (A==true) {
    UPDATE X
}
```

```
if (A==true) {
    SELECT X
}
```
Sufficient conditions to establish a dependency relation between two operation instances

- If there is a mutually accessed record
- and both operations are reached
- and the update is visible to the select

\[
\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}}
\]
Sufficient conditions to establish a dependency relation between two operation instances

- If there is a mutually accessed record
- and both operations are reached
- and the update is visible to the select

Operations must be dependent by \( WR \)

\[
\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}}
\]
\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]
\[ \varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}} \]

- Enforces the existence of an anomaly
- Enforces the existence of an anomaly
- Parametrized over three variables: $i, j$ and $k$
Enforces the existence of an anomaly

Parametrized over three variables: $i, j$ and $k$

Bounds on the state space
Enforces the existence of an anomaly

Parametrized over three variables: $i$, $j$ and $k$

Instantiates $i$ serially executed transactions,
Enforces the existence of an anomaly
Parametrized over three variables: \(i\), \(j\) and \(k\)
Instatiates \(i\) serially executed transactions,
leading to \(j\) concurrent transactions
Enforces the existence of an anomaly
- Parametrized over three variables: $i, j$ and $k$
- Instantiates $i$ serially executed transactions,
- leading to $j$ concurrent transactions
- that form a dependency cycle of length $k$
TESTING: FUNDAMENTAL CHALLENGES (REVISITED)

- Rich and precise encoding
Rich and precise encoding
Rich and precise encoding

Triggering anomalies requires determining:

- Initial database state
- Input arguments
- Execution order
- Network delays
Rich and precise encoding
Triggering anomalies requires determining:
- Initial database state
- Input arguments
- Execution order
- Network delays

Concrete database instances

TESTING: FUNDAMENTAL CHALLENGES (REVISITED)
Rich and precise encoding
Triggering anomalies requires determining:
- Initial database state
- Input arguments
- Execution order
- Network delays

Concrete database instances
Transaction instances
Rich and precise encoding

Triggering anomalies requires determining:
- Initial database state
- Input arguments
- Execution order
- Network delays

Concrete database instances

Transaction instances

Control-flow sensitive
Rich and precise encoding

Triggering anomalies requires determining:

- Initial database state ✔
- Input arguments ✔
- Execution order ✔
- Network delays
Rich and precise encoding

Triggerring anomalies requires determining:
- Initial database state ✓
- Input arguments ✓
- Execution order ✓
- Network delays ✓

Testing: Fundamental Challenges (Revisited)
CLOTHO: BUG DETECTION MECHANISM
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for java programs
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for Java programs
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for Java programs
- Compiles programs down to an abstract representation
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for java programs
- Compiles programs down to an abstract representation
- FOL encoding engine, backed by Z3 SMT solver
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for java programs
- Compiles programs down to an abstract representation
- FOL encoding engine, backed by Z3 SMT solver
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for java programs
- Compiles programs down to an abstract representation
- FOL encoding engine, backed by Z3 SMT solver
- Efficient search algorithm
CLOTHO: BUG DETECTION MECHANISM

- Static analysis engine for java programs
- Compiles programs down to an abstract representation
- FOL encoding engine, backed by Z3 SMT solver
- Efficient search algorithm
- Returns annotated code containing concrete anomalies
Directed test framework
Directed test framework
- automated step-by-step replaying of annotated buggy programs
Directed test framework
- automated step-by-step replaying of annotated buggy programs
- synchronized drivers
Directed test framework
- automated step-by-step replaying of annotated buggy programs
- synchronized drivers
- managed connection throttler in a cluster of database nodes
EMPIRICAL RESULTS: APPLICABILITY
EMPIRICAL RESULTS: APPLICABILITY

- 7 benchmarks of various complexity and different properties were analyzed
EMPIRICAL RESULTS: APPLICABILITY

- **7 benchmarks** of various complexity and different properties were analyzed
- Serializability anomalies were found and successfully replayed in 5 application

![Bar chart showing the number of anomalies across different benchmarks](chart)
EMPIRICAL RESULTS: APPLICABILITY

- **7 benchmarks** of various complexity and different properties were analyzed.

- Serializability anomalies were found and successfully replayed in 5 applications.

![Bar chart showing number of anomalies per application]

- ~25m per application (avg)
- 17 anomalies per application (avg)
EMPIRICAL RESULTS: COMPARISON TO BLACKBOX TESTING

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
</tr>
</tbody>
</table>
### Case study: TPC-C

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
</tr>
</tbody>
</table>
Case study: TPC-C

Anomalies were studied and mapped to invariant violations

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
</tr>
</tbody>
</table>
Case study: TPC-C

Anomalies were studied and mapped to invariant violations

All invariants were broken as a result of at least one serializability anomaly

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
<th>CLOTHO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
- **Case study: TPC-C**

- Anomalies were studied and mapped to invariant violations

- **All invariants were broken** as a result of at least one serializability anomaly

- Only 3 serializability anomalies did not result in any invariant violation

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
<th>CLOTHO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
SUMMARY
SUMMARY

- CLOTHO: an end-to-end directed testing framework for weakly consistent database programs
CLOTHO: an end-to-end directed testing framework for weakly consistent database programs

The problem of finding serializability anomalies is reduced to finding satisfying assignments to a formula
CLOTHO: an end-to-end directed testing framework for weakly consistent database programs

The problem of finding serializability anomalies is reduced to finding satisfying assignments to a formula

Applicable on many benchmark applications
CLOTHO: an end-to-end directed testing framework for weakly consistent database programs

The problem of finding serializability anomalies is reduced to finding satisfying assignments to a formula

Applicable on many benchmark applications

Outperforms state of the art blackbox testing techniques
THANK YOU!

QUESTIONS?

TOOL AVAILABLE
- Includes transaction instances, arguments
- Accompanied by a test configuration file specifying execution order and networking details

```java
@Parameters(10)
public void payment ... {
  ...
  @Sched(node="B", order=1)
  rs = stmt.executeQuery();
  ...
  @Sched(node="B", order=2)
  stmt.executeUpdate();
}
```

```plaintext
# initialize:
INSERT INTO
    CUST(c_id, c_pay_cnt)
VALUES (10, 50);
# schedule:
@T1@partitions{A,B}: Ins1-01
@T2@partitions{A,B}: Ins2-01
@T3@partitions{A,B}: Ins1-02
@T4@partitions{A,B}: Ins2-02
```

A1_In2.java  A1.conf
Rules specify the necessary conditions for establishing a dependency relation between two database operation instances.

\[q \equiv \text{SELECT } f \text{ AS } x \text{ WHERE } \phi\]

\[q' \equiv \text{UPDATE SET } f = v \text{ WHERE } \phi'\]

\[\text{txn}(q) = t \quad \text{txn}(q') = t' \quad t \neq t'\]

\[\mu_{q,q'}^{\text{RW}} = \exists r. [\phi]_t^B \land [\phi']_{t'}^B \land \text{Alive}(r, q) \land \text{Alive}(r, q') \land [\Lambda(q)]_t^B \land [\Lambda(q')]_{t'}^B\]
Rules specify the sufficient conditions for establishing a dependency relation between two database operation instances.
STRUCTURALLY SIMILAR ANOMALIES

- All share the same transaction instances and the same edges between them:

  $\text{txn1} \xrightarrow{\text{RW}} \xleftarrow{\text{WR}} \text{txn2}$
for $t \in [2, \max_t]$ do
    $c \leftarrow 3$
    while $c \leq \max_c$ do
        $\varphi_{NEG} \leftarrow \text{EncNeg}(cycles)$
        $\text{new}_cyc \leftarrow \text{isSAT}(\exists \mathbf{t}_1, \ldots, \mathbf{t}_t \cdot \varphi^c_{\text{CYCLE}}(\mathbf{t}_1, \ldots, \mathbf{t}_t) \land \varphi_{\text{DB}} \land \varphi_{\text{APP}} \land \varphi_{\text{NEG}})$
        if $\text{new}_cyc = \text{UNSAT}$ then $c \leftarrow c + 1$; continue;
        $\text{cycles} \leftarrow \text{cycles} \cup \{\text{new}_cyc\}$
        $\varphi_{\text{STCT}} \leftarrow \text{EncStruct}(\text{new}_cyc)$
        do
            $\varphi_{\text{NEG}} \leftarrow \text{EncNeg}(\text{cycles})$
            $\text{new}_cyc \leftarrow \text{isSAT}(\exists \mathbf{t}_1, \ldots, \mathbf{t}_t \cdot \varphi^c_{\text{CYCLE}}(\mathbf{t}_1, \ldots, \mathbf{t}_t) \land \varphi_{\text{DB}} \land \varphi_{\text{APP}} \land \varphi_{\text{NEG}} \land \varphi_{\text{STCT}})$
            if $\text{new}_cyc = \text{UNSAT}$ then break else $\text{cycles} \leftarrow \text{cycles} \cup \{\text{new}_cyc\}$;
        while true;
    for $\text{cyc} \in \text{cycles}$ do
        for $p \in [0, \max_p]$ do
            $\varphi_{\text{PATH}} \leftarrow \text{EncPath}(\text{cyc})$
            $\text{new}_\text{anml} \leftarrow \text{isSAT}(\exists \mathbf{t}_1, \ldots, \mathbf{t}_p \cdot \varphi_{\text{PATH}})$
            if $\text{new}_\text{anml} \neq \text{UNSAT}$ then $\text{anoms} \leftarrow \text{anoms} \cup \{\text{new}_\text{anml}\}$; break;
EFFECT OF OPTIMIZATIONS IN SEARCH ALGORITHM

Number of anomalies found within the same given time period
RELATED WORKS

- [Kaki et al. 2018], [Nagar et al. 2018]
  - Do not incorporate their techniques into a full test-and-reply environment

- [Brutschy et al. 2018]
  - Does not suit query-based models where dependences between two operations cannot be decided locally, but are reliant on other operations

- [Warszawski and Bailis 2017]
  - Does not consider how to help determining if applications executing on storage systems that expose guarantees weaker than serializability are actually **correct**