Query-aware Test Generation Using a Relational Constraint Solver

SHADI ABDUL KHALEK, BASSEM ELKARABLIEH,
YAI O. LALEYE, SARFRAZ KHURSHID

ASE’08

PRESENTED BY: YUSHAN ZHANG

Blackbox DBMS Testing

Query → DBMS → Results
Automat DBMS Testing

Requires the generation of:

1. **Test queries** for a given database schema
2. A set of test **databases**
3. **Oracles** to verify the result of query execution
Reality...manual work

The developers will create the regression tests (regular) by:

1. Generating the databases
2. Writing queries
3. Running the queries and compare it to the ground truth (oracle)

If there is a bug:

1. Add crash-triggering queries
2. Add the correct results for the queries (oracle)
3. Add the databases
Oracle...?

There are 1000 randomly generated rows in the table.

What is the correct #row returned?

```
SELECT DISTINCT id
FROM student
WHERE (id=1 OR (id>=3 AND id<=5));
```

Easy! At most 4!
Oracle...?

There are 10000 randomly generated rows in the table.

What is the correct #row returned?

```
SELECT DISTINCT id
FROM student
WHERE (id=1 OR (id>=3 AND id<=5)));
WHERE id > 200;
```

Could you still tell the correct number?
Generating data with oracle?

Given the schema for a database $D$ and a query $Q$:

$$SELECT \text{DISTINCT id FROM student WHERE (id=1 OR (id>=3 AND id<=5));}$$

The tool generates:

1. Data to fill the table
2. Oracle to verify the result

Data is the oracle!

QUERY-AWARE generation!
ADUSA Overview

- SQL input (schema, query)
  - SQL2Alloy Translator
  - Alloy Specification
  - Alloy Analyzer

- Input Test data
  - Alloy2SQL Translator
  - Alloy Instances

- DBMS (Query processor)
  - Expected Query result
  - Check
  - True
  - False
  - Counter Example

- Actual Query result

=> Solve the constraints

=> Insert the instances

=> Check for consistency
SQL => Alloy

Leverage the Alloy specification language for the transformation:

The query:

```sql
SELECT DISTINCT id FROM student WHERE (id=1 OR (id>=3 AND id<=5));
```

Becomes:

```alloy
fun query () : Int {
    (select |where{from{student.rowe}})
}

fun select {rows: Int -> varchar} : Int {
    (rows.varchar)
}

fun where{rows: Int -> varchar} : Int -> varchar {
    (x1: rows.varchar, x2: x1.rows | condition[x1])
}

pred condition(x1: Int) {
    (eq[x1, 1] or (gte[x2, 3] and lte[x1, 5] ))
}

fun from{rows: Int -> varchar} : Int -> varchar {
    (rows)
}
```

The Alloy Analyzer solves the constraints and generates instances.
Alloy => SQL

Alloy returns the following instances:

```
varchar: {varchar$0, varchar$1}
student:{student$0}
rows: {(1, varchar$1), (2, varchar$0), (4, varchar$1)}
$query: {1, 4}
```

Transformed into SQL insertions:

```
INSERT INTO student VALUES (1, varchar$1)
INSERT INTO student VALUES (2, varchar$0)
INSERT INTO student VALUES (4, varchar$1)
```
ADUSA Overview

=> Solve the constraints
=> Insert the instances
=> Verify the result
Experiments

1. With Oracle 11g (commercial)
   ◦ Verify the correctness of ADUSA

2. With MySQL (open source, multi-platform)
   ◦ Test ability in reproducing bugs

3. With HSQLDB (open source)
   ◦ Test ability to detect injected bugs
Database Schema and Queries

Q1: SELECT id
    FROM student
    WHERE name = 'John'

Q2: SELECT s.id, s.name
    FROM student s, enrollment e
    WHERE s.id = e.id

Q3: SELECT name
    FROM student NATURAL JOIN department

Q4: SELECT COUNT(*)
    FROM student NATURAL JOIN course NATURAL JOIN department
Experiment with Oracle 11g

<table>
<thead>
<tr>
<th>Query</th>
<th>Scope</th>
<th># of instances</th>
<th>Inconsistency</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>4</td>
<td>No</td>
<td>358</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15</td>
<td>No</td>
<td>1,193</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>75</td>
<td>No</td>
<td>6,020</td>
</tr>
<tr>
<td>Q2</td>
<td>2</td>
<td>182</td>
<td>No</td>
<td>56,108</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>≥1,000</td>
<td>No</td>
<td>59,592</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>≥1,000</td>
<td>No</td>
<td>78,577</td>
</tr>
<tr>
<td>Q3</td>
<td>2</td>
<td>83</td>
<td>No</td>
<td>6,887</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>≥1,000</td>
<td>No</td>
<td>33,160</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>≥1,000</td>
<td>No</td>
<td>39,707</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>200</td>
<td>Yes</td>
<td>10,522</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>Yes</td>
<td>229</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>Yes</td>
<td>≤1</td>
</tr>
</tbody>
</table>

**TABLE I.** Results for executing ADUSA on Oracle11g using 4 test queries.

*Scope: #varchar values*
Experiment with MySQL

Reproduce Bug 13371 in MySQL 4.0; Scope for int is 4.

CREATE TABLE keytest (  
id int(10) NOT NULL default '1',  
PRIMARY KEY (id)  
)

SELECT *  
FROM keytest  
WHERE (id=4 OR (id>=1 AND id<=3) OR (id>=2 AND id<=5))

253/826 counter examples (databases), < 10s
Experiment with HSQLDB

1. Run the query with ADUSA => 1108 instances, < 10s
2. Modify the source code to retrieve partial data => fault injection
3. Use ADUSA to verify the execution => use the generated instances

```sql
SELECT sid
FROM student s, course c
WHERE s.id = c.cid
```

499/1108 counter examples.
Summary

- Automatic DBMS testing need the generation of the query, database and oracle.
- ADUSA generates the database which is also the oracle.
  - It translates SQL $\leftrightarrow$ Alloy
  - It uses Alloy analyzer to solve the constraints
  - It executes the SQL and checks for inconsistency
Reflections

1. Gives a good insight to solve the oracle problem: ‘should-exist’ result
2. SQL <=> Alloy translation is hard to extend, mainly involves manual work
3. How to retrieve the queries?
   - Translates the SQL grammar and enumerate [the author’s later work]
   - Random generation [arxiv’20, ETH Zurich]