Demonstration of the VeriEQL Equivalence Checker for Complex SQL Queries

Pinhan Zhao[1*], Yang He[2*], Xinyu Wang[1], Yuepeng Wang[2]
[1]University of Michigan, USA [2]Simon Fraser University, Canada

Motivation

Why is equivalence checking of SQL queries important? This task has a wide variety of application scenarios.

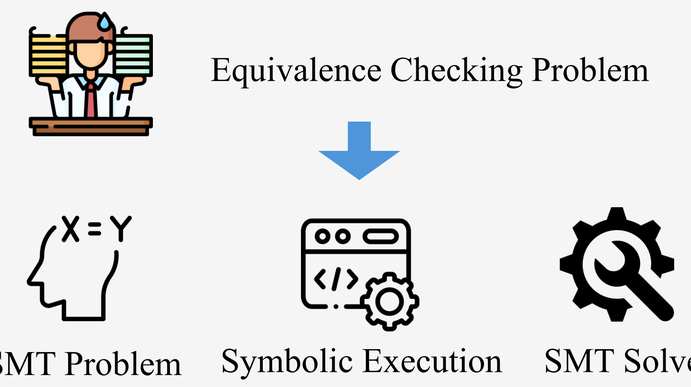
Rewrite Validation | Auto-grading | Bug Detection

Problem Statement

Bounded equivalence verification: Given two SQL queries under a database schema, VeriEQL aims to verify whether these two queries always produce identical results on all possible input databases up to a bounded size that conform to the schema.

Technique

Key idea: VeriEQL reduces the equivalence checking problem into an SMT problem using symbolic reasoning and utilizes off-the-shelf constraint solvers to determine the satisfiability of SMT formulas.



Highlights

Expressive query language. VeriEQL supports SPJ, GROUP BY, aggregate functions, three-valued semantics, set/bag operations, conditional statements, etc.

Genuine counterexample. VeriEQL refutes non-equivalent SQL queries with concrete database instances and SMT formulas.

Dialects. VeriEQL supports different SQL dialects, i.e., MySQL, MariaDB, Oracle, and PostgreSQL.

Good scalability. VeriEQL can check 70% of the 15,200 benchmarks with bound 5 in 5 minutes.

Small-Scope Hypothesis. 96% of non-equivalent benchmarks are refuted with less than 3 tuples.

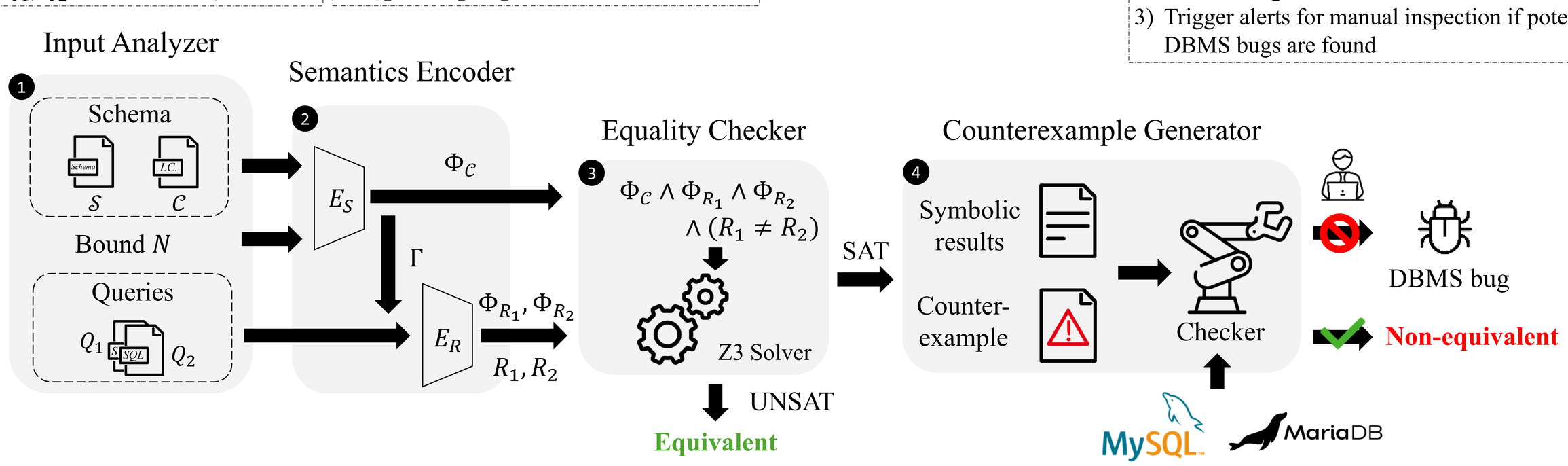
System Overview

Input Analyzer
1) Check the syntax of SQL queries Q_1, Q_2 and schemas $\mathcal{S}$, $\mathcal{C}$
2) Check the consistency between SQL queries Q_1, Q_2 and schemas $\mathcal{S}$, $\mathcal{C}$

Semantics Encoder
1) Build symbolic database Γ over bound N
2) Encode integrity constraints $\mathcal{C}$ into $\Phi_{\mathcal{C}}$
3) Symbolically execute SQL queries Q_1, Q_2 into R_1, R_2

Equality Checker
1) Check the satisfiability of $\Phi_{\mathcal{C}} \wedge \Phi_{R_1} \wedge \Phi_{R_2} \wedge (R_1 \neq R_2)$ using Z3

Counterexample Generator
1) Generate executable counterexample (i.e., SQL codes)
2) Compare symbolic results against execution results using DBMSs
3) Trigger alerts for manual inspection if potential DBMS bugs are found



Schema $\mathcal{S}$:

```
EMP: {id: int, age: int, …}
```

Integrity constraints $\mathcal{C}$:

```
PK: EMP.id
```

SQL queries:

```
Q1: SELECT id FROM EMP
    WHERE age > 30
Q2: SELECT id FROM EMP
    WHERE age >= 30
```

Symbolic table Γ ($N = 2$):

EMP

	id	age	…
t_1	t_1.id	t_1.age	…
t_2	t_2.id	t_2.age	…

Constraint formula $\Phi_{\mathcal{C}}$:

$$-2^{31} \leq t_1.\text{id} \leq 2^{31}-1 \wedge -2^{31} \leq t_1.\text{age} \leq 2^{31}-1$$
$$\wedge -2^{31} \leq t_2.\text{id} \leq 2^{31}-1 \wedge -2^{31} \leq t_2.\text{age} \leq 2^{31}-1$$
$$\wedge\, t_1.\text{id} \neq t_2.\text{id} \wedge t_1.\text{id} \neq \text{Null} \wedge t_2.\text{id} \neq \text{Null}$$

Backend DBMSs

Φ_{R_1}: $(t_1.\text{age} > 30 \rightarrow \neg\text{Del}(t_1') \wedge t_1'.\text{id} = t_1.\text{id}) \wedge (t_1.\text{age} \leq 30 \rightarrow \text{Del}(t_1'))$
$\wedge\ (t_2.\text{age} > 30 \rightarrow \neg\text{Del}(t_2') \wedge t_2'.\text{id} = t_2.\text{id}) \wedge (t_2.\text{age} \leq 30 \rightarrow \text{Del}(t_2'))$

Φ_{R_2}: $(t_3.\text{age} \geq 30 \rightarrow \neg\text{Del}(t_3') \wedge t_3'.\text{id} = t_1.\text{id}) \wedge (t_3.\text{age} < 30 \rightarrow \text{Del}(t_3'))$
$\wedge\ (t_4.\text{age} \geq 30 \rightarrow \neg\text{Del}(t_4') \wedge t_4'.\text{id} = t_2.\text{id}) \wedge (t_4.\text{age} < 30 \rightarrow \text{Del}(t_4'))$

where Del is an uninterpreted function.

Symbolic results

R_1

	id
t_1'	t_1'.id
t_2'	t_2'.id

R_2

	id
t_3'	t_3'.id
t_4'	t_4'.id

Demonstration Scenarios

VeriEQL can validate optimized SQL queries.

Q_1	`SELECT DEPTNO, COUNT(*) FILTER (WHERE JOB = 'CLERK') FROM (SELECT * FROM EMP WHERE DEPTNO = 10 UINON ALL SELECT * FROM EMP WHERE DEPTNO > 20) AS t3 GROUP BY DEPTNO`
Q_2	`SELECT DEPTNO, COALESCE(SUM(EXPR$1), 0) FROM (SELECT DEPTNO, COUNT(*) FILTER (WHERE JOB = 'CLERK') AS EXPR$1 FROM EMP WHERE DEPTNO = 10 GROUP BY DEPTNO UNION ALL SELECT DEPTNO, COUNT(*) FILTER (WHERE JOB = 'CLERK') AS EXPR$1 FROM EMP WHERE DEPTNO > 20 GROUP BY DEPTNO) AS t12 GROUP BY DEPTNO`

Q_1: the optimized query, Q_2: the original query

The testPushCountFilterThroughUnion test case of Apache Calcite.

Time to check query equivalence on different input sizes for validating query optimizations.

Size	1	2	3	4	5	6	7	8	9
Time (s)	0.2	0.4	0.6	1.0	2.4	6.6	19.7	98.5	118.2

Good scalability

VeriEQL can auto-grade queries from LeetCode.

Q_1	`WITH temp AS (SELECT DISTINCT A.customer_id, B.customer_id, B.customer_name SUM(CASE WHEN A.product_name IN ('A', 'B') THEN 1 ELSE 0 END) AS AB, SUM(CASE WHEN A.product_name = 'C' THEN 1 ELSE 0 END) AS C, FROM orders A JOIN customers B ON A.customer_id = B.customer_id GROUP BY A.customer_id) SELECT customer_id, customer_name FROM temp WHERE AB >= 2 AND C = 0`
Q_2	`SELECT customer_id, customer_name FROM customers WHERE customer_id IN (SELECT DISTINCT customer_id FROM orders WHERE product_name = 'A') AND customer_id IN (SELECT DISTINCT customer_id FROM orders WHERE product_name = 'B') AND customer_id NOT IN (SELECT DISTINCT customer_id FROM orders WHERE product_name = 'C') ORDER BY customer_id`

Q_1: a user-provided answer, Q_2: the ground-truth

Question sourced from
https://leetcode.com/problems/customers-who-bought-products-a-and-b-but-not-c/

orders

order_id	customer_id	product_name
0	0	B
1	0	B

customers

customer_id	product_name
0	Alice
1	Bob

Q_1's output

customer_id	customer_name
0	Alice

Q_2's output

customer_id	customer_name

VeriEQL can find bugs of DBSMs.

Q_1	`SELECT DISTINCT page_id AS recommended_page FROM (SELECT CASE WHEN user1_id = 1 THEN user2_id WHEN user2_id = 1 THEN user1_id ELSE NULL END AS user_id FROM friendship) AS tb1 JOIN likes AS tb2 ON tb1.user_id = tb2.user_id WHERE page_id NOT IN (SELECT page_id FROM likes WHERE user_id = 1)`
Q_2	`SELECT DISTINCT page_id AS recommended_page FROM (SELECT b.user_if, b.page_id FROM friendship a LEFT JOIN likes b ON (a.user2_id = b.user_id OR a.user1_id = b.user_id) AND (a.user1_id = 1 OR a.user2_id = 1) WHERE b.page_id NOT IN ( SELECT DISTINCT page_id FROM likes WHERE user_id = 1)) T`

Q_1: a user-provided answer, Q_2: the ground-truth

Question sourced from
https://leetcode.com/problems/page-recommendations/

friendship

user1_id	user2_id
0	1

likes

user_id	page_id
-1	0

Q_1's output

page_id

Q_2's expected output

page_id
NULL

Q_2's real output

page_id

An implementation bug in MySQL v8.0.32.
https://bugs.mysql.com/bug.php?id=110244

https://github.com/whatsmyname/VeriEQL-demo

VLDB2024 GUANGZHOU