Automated Reasoning of Database Queries

Shumo Chu, University of Washington
with Konstantin Weitz, Chenglong Wang, Daniel Li, Brendan Murphy,
Jared Roesch, Alvin Cheung, Dan Suciu
• SQL: a language supported by all relational databases

• Restricted abstraction enabling powerful optimizations

• 30 years of research results in many optimizations based on semantic equivalent rewrites

Lack tools that can reason about SQL equivalences
Automated Solver for SQL:
Q1 = Q2?
Automated Solver for SQL:
∀ possible input, \( Q1 = Q2 \)?
Automated Solver for SQL: ∀ possible input, Q1 = Q2?

• Correctness of query rewrite

Big Data Landscape
Automated Solver for SQL: 
∀ possible input, Q1 = Q2?

- Correctness of query rewrite
- Semantics layer of data systems

Big Data Landscape
Automated Solver for SQL:

∀ possible input, Q1 = Q2?

• Correctness of query rewrite
• Semantics layer of data systems
• Auto grading on SQL assignments

Big Data Landscape
Challenges

• Checking equivalences of two FO sentences over finite models is undecidable 😞

• Rich language features
  • Aggregation and grouping
  • Index and integrity constraints
  • Correlated subqueries
Undecidability ≠ No Proof

Interactive Theorem Prover that validates mechanized proofs
Undecidability ≠ No Proof

The model of an inequivalence is usually not large

Interactive Theorem Prover that validates mechanized proofs

Constraint Solver for Model Checking
Construct a SQL Solver

- Cosette: An (almost) automated solver for SQL by combining interactive theorem proving and constraints solving (PLDI 17, SIGMOD17, CIDR17, VLDB18)
The Search for Formal SQL Semantics
What is SQL?

- More than 1,700 pages natural language description
- Insufficient for a rigorous SQL semantics
What is SQL?

- Data model: relations
  - A relation is an unordered collection of tuples
- Schema: set of attributes, each tuple in a relation must have same schema
- SQL query: \( Q(R_1, R_2, \ldots, R_k) = R \)
Relational Algebra

Relational Algebra \((\times, \sigma, \Pi, \cup, -)\)

- \(\times\): e.g. \(\text{Courses} \times \text{Student} = \text{SELECT } * \text{ FROM Course, Student}\)
- \(\sigma\): e.g. \(\sigma_{\text{len} > 100}(\text{Movie}) = \text{SELECT } * \text{ FROM Movie WHERE len > 100}\)
- \(\Pi\): e.g. \(\Pi_{\text{first}}(\text{Name}) = \text{SELECT first FROM Name}\)
- \(\cup\): e.g. \(R \cup S = (\text{SELECT } * \text{ FROM R}) \text{ UNION ALL } (\text{SELECT } * \text{ FROM S})\)
Relational Algebra

Relational Algebra \( (x, \sigma, \Pi, u, -) \) \( \rightarrow \) UCQ

- \( x \): e.g. Courses \( \times \) Student = SELECT \* FROM Course, Student
- \( \sigma \): e.g. \( \sigma_{\text{len} > 100}(\text{Movie}) \) = SELECT \* FROM Movie WHERE len > 100
- \( \Pi \): e.g. \( \Pi_{\text{first}}(\text{Name}) \) = SELECT first FROM Name
- \( u \): e.g. \( R \cup S = (\text{SELECT \* FROM R}) \cup \text{ALL (SELECT \* FROM S)} \)
Relational Algebra

- More operators needed
  - \( \delta \): e.g. \( \delta(R) = \text{SELECT DISTINCT } * \text{ FROM } R \)
  - \( \gamma \): e.g. \( \gamma_{\text{name}, \text{count}(*)}\text{Person} = \text{SELECT } \text{name, count}(*) \text{ FROM Person GROUP BY } \text{name} \)
- Constraints: keys, foreign keys ...
- Lacking “meta-theory”
Relational Algebra

- More operators needed
  - $\delta$: e.g. $\delta(R) = \text{SELECT DISTINCT } * \text{ FROM } R$
  - $\gamma$: e.g. $\gamma_{\text{name, count}(*)} \text{Person} = \text{SELECT name, count(*) FROM } \text{Person GROUP BY name}$
- Constraints: keys, foreign keys ...
- Lacking "meta-theory"

Leaking Abstraction: What is the semantics of RA?
Relation as List

Existing Coq Formalization [Malecha et al., POPL 10]

- Two queries are equivalent if returning the **same** result for **all possible** input relations
- Need to reason about two lists are equivalent up to permutations
Q1 = SELECT *
FROM (R UNION ALL S)
WHERE b

Q2 = (SELECT * FROM R WHERE b)
UNION ALL
(SELECT * FROM S WHERE b)
Q1 = \textbf{SELECT} *  
\textbf{FROM} (R \textbf{UNION ALL} S) 
\textbf{WHERE} b

Q2 = (\textbf{SELECT} * \textbf{FROM} R \textbf{WHERE} b) 
\textbf{UNION ALL} 
(\textbf{SELECT} * \textbf{FROM} S \textbf{WHERE} b)

Q1 = Q2 ?

Induction on R:
Assume Q1 == Q2 when R has N tuples
Then when R is of size N+1:
...

Induction on S:
Assume Q1 == Q2 when S has N tuples
Then when S is of size N+1:
...

\textbf{WHY SO MANY INDUCTIONS?}
Q1 = SELECT * 
   FROM (R UNION ALL S) 
   WHERE b

Q2 = (SELECT * FROM R WHERE b) 
   UNION ALL 
   (SELECT * FROM S WHERE b)

Q1 = Q2 ?

Induction on R:
Assume Q1 = Q2 when R has N tuples
Then when R is of size N+1:
   ...

Induction on S:
Assume Q1 = Q2 when S has N tuples
Then when S is of size N+1:
   ...

400 Line of Coq for a simple rewrite, very limited
rewrites has been proven
Univalent SQL Semantics
[Chu et al., PLDI 17]
Relation as Function

$R : \text{Relation} \rightarrow [R] : \text{Tuple} \rightarrow \mathbb{N}$
Relation as Function

R: Relation $\rightarrow [R]: Tuple \rightarrow \mathbb{N}$

b: Predicate $\rightarrow [b]: Tuple \rightarrow \{0, 1\}$

```
SELECT * FROM R
WHERE b
```

$Q(t) = [R] t \times [b] t$

```
R UNION ALL S
```

$Q(t) = [R] t + [S] t$
Q1 = SELECT *
    FROM (R UNION ALL S)
    WHERE b

Q2 = (SELECT * FROM R WHERE b)
    UNION ALL
    (SELECT * FROM S WHERE b)
Q1 = SELECT *
FROM (R UNION ALL S)
WHERE b

Q1(t) = ([R](t) + [S](t)) x [b](t)

Q2 = (SELECT * FROM R WHERE b)
    UNION ALL
    (SELECT * FROM S WHERE b)

Q2(t): [R](t) x [b](t) + [S](t) x [b](t)
Q1 = SELECT *
    FROM (R UNION ALL S)
    WHERE b

Q2 = (SELECT * FROM R WHERE b)
    UNION ALL
    (SELECT * FROM S WHERE b)

Q1(t) = ([R](t) + [S](t)) \times [b](t)

Q2(t): [R](t) \times [b](t) + [S](t) \times [b](t)

Q1 = Q2 ?

Algebraic Reasoning

Distrib. Reflex. ...

QED
Problem 1: Projection

\[ \text{[SELECT first FROM Name]} = ? \]

<table>
<thead>
<tr>
<th>Name</th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Michael</td>
<td>Shulman</td>
</tr>
<tr>
<td></td>
<td>Michael</td>
<td>Jordan</td>
</tr>
<tr>
<td></td>
<td>Alex</td>
<td>Aiken</td>
</tr>
<tr>
<td></td>
<td>Alex</td>
<td>Krizhevsky</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
</tr>
<tr>
<td>Alex</td>
</tr>
</tbody>
</table>
Problem 1: Projection

\[ SELECT \text{first} \text{FROM Name} \] = ?

Add up multiplicities:
\[ Q(“Michael”) = 2 \]
\[ Q(“Alex”) = 2 \]
Problem 1: Projection

\[
[SELECT \text{first} \text{FROM Name}] = ?
\]

\[
Q(t') = \sum_{t: \text{Tuple}} [t.\text{first} = t'.\text{first}] \times \llbracket \text{Name} \rrbracket(t)
\]
Problem 1: Projection

\[ \text{SELECT } \text{first FROM Name} = \? \quad 0 \text{ or } 1 \]

\[ Q(t') = \sum_{t: \text{Tuple}} [t.\text{first} = t'.\text{first}] \times [\text{Name}](t) \]

Issues:
- Sum can potentially be infinite
- Need to reason about equality with algebraic expressions (now with sums ... )
Problem 2: Duplicate Elimination

\[
\text{[SELECT DISTINCT first FROM Name]} = ?
\]

Proposal: A step functions over \( \mathbb{N} \): e.g. \( S(0) = 0, S(x) = 1 \)

Issues: Back to inductive proof again 😞
Univalent Semantics

• HoTT Type instead of $\mathbb{N}$

• Dependent Pair Type ($\Sigma$-Type) to represent summation

• Squash Type to represent duplicate elimination

• Bonus: Unifies types and propositions
Introduction of $\theta$-semijoin:

$$R_1 \bowtie_{\theta} R_2 \equiv R_1 \bowtie_{\theta} (R_2 \times_{\theta} R_1)$$

Pushing $\theta$-semijoin through join:

$$(R_1 \bowtie_{\theta_1} R_2) \times_{\theta_2} R_3 \equiv (R_1 \bowtie_{\theta_1} R'_2) \times_{\theta_2} R_3$$

$$R'_2 = E_2 \times_{\theta_1 \wedge \theta_2} (R_1 \bowtie R_3)$$

Pushing $\theta$-semijoin through aggregation:

$$g \mathcal{F}_f(R_1) \times_{c_1=c_2} R_2 \equiv g \mathcal{F}_f(R_1 \times_{c_1=c_2} R_2)$$
Introduction of \( \theta \)-semijoin:

\[
R_1 \Join_\theta R_2 \equiv R_1 \Join_\theta (R_2 \bowtie_\theta R_1)
\]

Pushing \( \theta \)-semijoin through join:

\[
(R_1 \Join_{\theta_1} R_2) \bowtie_{\theta_2} R_3 \equiv (R_1 \Join_{\theta_1} R'_2) \bowtie_{\theta_2} R_3
\]

\[R'_2 = E_2 \bowtie_{\theta_1 \land \theta_2} (R_1 \Join R_3)\]

Pushing \( \theta \)-semijoin through aggregation:

\[
\text{if } g \in \mathcal{F}_f(R_1) \bowtie_{c_1=c_2} R_2 \equiv \text{if } \mathcal{F}_f(R_1 \bowtie_{c_1=c_2} R_2)
\]
Can we do better?
Axiomatic Foundations and Semi-Decision Procedures

[Chu et al., VLDB 18]
**Idea 1:** projection requires summation

\[
\text{[SELECT } \text{first FROM Name]} \equiv \\
\text{[Q]}(t') = \sum_t \text{[t.name = t'.name]} \times \text{[R]}(t)
\]

**Solution:** new \* infinitary operator \(\sum_t\)
Idea 2: **DISTINCT** converts bags to sets

\[
\text{[SELECT DISTINCT first FROM Name]} \equiv \sum_t [t.\text{first}=t'.\text{first}] \times [\text{Name}(t)]
\]

**Solution:** add the **squash**\(^*\) operator \(\|...\|\)

**Intuition:** \(\|\emptyset\| = 0\) and \(\|x\| = 1\) otherwise
Idea 3: non-monotone operators e.g. EXCEPT, NOT, EXIST

Solution: add a $\text{not}(\ldots)$ operator

Intuition: $\text{not}(0) = 1$, $\text{not}(x) = 0$ otherwise
Definition: An **Unbounded-semiring** is a structure \((U, 0, 1, +, \times, \|\ldots\|, \text{not}, (\sum_{D \in \mathfrak{D}})\) satisfying a list of axioms
U-Semirings

Definition: An **Unbounded-semiring** is a structure \((U, 0, 1, +, \times, \|\ldots\|, \not, (\sum_D)_{D \in \Sigma})\) satisfying a list of axioms.
Example: Axiomization of Sum

\[ \sum_t (f_1(t) + f_2(t)) = \sum_t f_1(t) + \sum_t f_2(t) \]

\[ \sum_{t_1} \sum_{t_2} f(t_1, t_2) = \sum_{t_2} \sum_{t_1} f(t_1, t_2) \]

\[ x \times \sum_t f(t) = \sum_t x \times f(t) \]

\[ \| \sum_t f(t) \| = \| \sum_t \| f(t) \| \| \]

Distributivity

Commutativity

Associativity

Idempotency on Squash
Axiomization of Integrity Constraints

- The **key constraint** on $R_k$ is the identity:

$$[t.k = t'.k] \times R(t) \times R(t') = [t = t'] \times R(t)$$
Axiomization of Integrity Constraints

- The **key constraint** on $R.k$ is the identity:

$$[t.k = t'.k] \times R(t) \times R(t') = [t = t'] \times R(t)$$
Axiomization of Integrity Constraints

- The **key constraint** on \( R.k \) is the identity:

\[
[t.k = t'.k] \times R(t) \times R(t') = [t = t'] \times R(t)
\]

- The **foreign key constraint** from \( S.fk \) to \( R.k \) is:

\[
S(t') = S(t') \times \sum_t R(t) \times [t.k = t'.fk]
\]
Proving Equivalences with ICs

-- Q1
SELECT ip.np, itm.type, itm.itemno
FROM (SELECT DISTINCT itp.itemno AS itn,
       itp.np AS np
       FROM price WHERE price.np > 1000) ip, itm
WHERE ip.itn = itm.itemno;

-- Q2
SELECT DISTINCT price.np, itm.type, itm.itemno
FROM price, itm
WHERE price.np > 1000 AND itp.itemno = itm.itemno;

itemno is a key of itm
Proving Equivalences with ICs

\[ Q_1(t) = \sum_{t_1, t_2} [t_1.np = t.np] \times [t_2.type = t.type] \times [t_2.itemno = t.itemno] \times [t_1.itn = t_2.itemno] \times [t.np > 1000] \times \text{price}(t) \times \text{itm}(t_2) \]

\[ Q_2(t) = \sum_{t_1, t_2} [t_2.type = t.type] \times [t_2.itemno = t.itemno] \times [t_1.itemno = t.itemno] \times [t_2.itemno = t_1.itemno] \times [t_1.np = t.np] \times [t_1.np > 1000] \times \text{price}(t_1) \times \text{itm}(t_2) \]
Proving Equivalences with ICs

\[ Q_1(t) = \sum_{t_1, t_2} [t_1.np = t.np] \times [t_2.type = t.type] \times [t_2.itemno = t.itemno] \times [t_1.itn = t_2.itemno] \times \sum_{t'} [t'.itemno = t_1.itn] \times [t'.np = t_1.np] \times [t'.np > 1000] \times \text{price}(t') \times \text{itm}(t_2) \]

Rewrite Using U-Semiring Axioms

\[ Q_1(t) = \sum_{t_2, t'} [t_2.type = t.type] \times [t_2.itemno = t.itemno] \times [t'.itemno = t.itemno] \times [t'.np = t.np] \times [t'.np > 1000] \times \text{price}(t') \times \text{itm}(t_2) \]

\[ Q_2(t) = \sum_{t_1, t_2} [t_2.type = t.type] \times [t_2.itemno = t.itemno] \times [t_1.itemno = t.itemno] \times [t_2.itemno = t_1.itemno] \times [t_1.np = t.np] \times [t_1.np > 1000] \times \text{price}(t_1) \times \text{itm}(t_2) \]
Algorithm: Main Idea

Check $[Q_1] = [Q_2]$ by generalizing two known cases:

- UCQ under set semantics:
  - Check for homomorphisms $Q_1 \leftrightarrow Q_2$
- UCQ under bag semantics:
  - Check for isomorphisms $Q_1 \rightarrow Q_2$
- “Chase” the axiomatization of constraints
What about inequivalent queries?
Finding Counterexamples using Constraint Solver

[Chu et al., CIDR17]
Cosette Architecture

- Cosette: An (almost) automated solver for SQL by combining interactive theorem proving and constraints solving (PLDI 17, SIGMOD17, CIDR17, VLDB18)

SQL: Q1 = Q2?

- Proposition Generator
  - Propositions
  - Proof Search and Validation
  - Equivalent

- Constraint Generator
  - Constraints
  - Model Checker for SQL
  - Not Equal: Counter Examples
Cosette Architecture

- Cosette: An (almost) automated solver for SQL by combining interactive theorem proving and constraints solving (PLDI 17, SIGMOD 17, CIDR 17, VLDB 18)

SQL: \( Q_1 = Q_2? \)

- Proposition Generator
  - Propositions
  - Proof Search and Validation
  - Equivalent

- Constraint Generator
  - Constraints
  - Model Checker for SQL
  - Not Equal: Counter Examples
Finding Counter Examples with SMT Solver

SQL: Q1 = Q2?

Constraint Generator

Constraints

SMT Solver

Not Equal: Counter Examples
Finding Counter Examples with SMT Solver

SQL: Q1 = Q2?

Constraint Generator

Equal: Increase Model Size

Constraints

SMT Solver

Not Equal: Counter Examples
Encoding SQL in Solvers

- A tuple as a list

\[
\text{Tuple} := \text{List } <\text{Integer}>\]
Encoding SQL in Solvers

- A tuple as a list
  
  \[
  \text{Tuple} \equiv \text{List} \langle \text{Integer} \rangle
  \]

- A relation as tuples tagged with multiplicity
  
  \[
  \text{Relation} \equiv \text{List} \langle \text{Pair}<\text{Tuple}, \text{Integer}> \rangle
  \]

- A SQL query as constraints over symbolic values
Encoding SQL in Solvers

- A tuple as a list
  
  $$\text{Tuple} := \text{List } \langle \text{Integer} \rangle$$

- A relation as tuples tagged with multiplicity
  
  $$\text{Relation} := \text{List } \langle \text{Pair}<\text{Tuple}, \text{Integer} \rangle \rangle$$

- A SQL query as constraints over symbolic values
Performance Optimizations

- Maximize concrete evaluation/ Minimize Symbolic Execution
- Symmetry breaking
Cosette

Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!

Example: Count Bug.

```sql
schema s(pnum:int, shipdate:int);
schema p(pnum:int, qoh:int);
table parts(p);
table supply(s);

-- Kim, W. ACM Trans. Database System 1982
-- found incorrect 3 years later

query q1
'select x.pnum as xp
from parts x
where x.qoh = (select count(y.shipdate) as cnt
  from supply y
  where y.pnum = x.pnum AND y.shipdate < 10)';

query q2
'select x.pnum as xp
from parts x,
(select y.pnum as suppnum, count(y.shipdate) as ct
  from supply y where y.shipdate < 10
  group by y.pnum) temp
where x.qoh = temp.ct AND x.pnum = temp.suppnum';

verify q1 q2;
```
Cosette

Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!

Example: Count Bug.

```sql
1. schema s(pnum:int, shipdate:int);
2. schema p(pnum:int, qoh:int);
3. table parts(p);
4. table supply(s);
5. -- Kim, W. ACM Trans. Database System 1982
6. -- found incorrect 3 years later
7. query q1
8. select x.pnum as xp
9. from parts x
```
Cosette

Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!

Example: Count Bug.

```
schema s(pnum:int, shipdate:int);
schema p(pnum:int, qoh:int);
table parts(p);
table supply(s);

-- Kim, W. ACM Trans. Database System 1982
-- found incorrect 3 years later

query q1
  'select x.pnum as xp
  from parts x
  where x.qoh = (select count(y.shipdate) as cnt 
                      from supply y 
                      where y.pnum = x.pnum AND y.shipdate < 10)';

query q2
  'select x.pnum as xp
  from parts x, (Select y.pnum as suppnum, count(y.shipdate) as ct 
                from supply y where y.shipdate < 10 
                group by y.pnum) temp 
  where x.qoh = temp.ct AND x.pnum = temp.suppnum';

verify q1 q2;
```
Evaluating Cosette

- **Bug**: 3 real-world optimizer bugs
- **XData**: query and mutant pairs collected from XData, a test generation framework
- **Exams**: a set of questions from the undergraduate data management class
- **Rules**: 68 query rewrite rules from database literatures and real-world optimizers
Evaluating Cosette

- **Bug**: 3 real-world optimizer bugs
- **XData**: query and mutant pairs collected from XData, a test generation framework
- **Exams**: a set of questions from the undergraduate data management class
- **Rules**: 68 query rewrite rules from database literatures and real-world optimizers
### Evaluating Cosette

#### Dataset Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total #</th>
<th>Average time taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bugs</td>
<td>3</td>
<td>8.3s</td>
</tr>
<tr>
<td>XData</td>
<td>9</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>Exams</td>
<td>5</td>
<td>1.3s</td>
</tr>
</tbody>
</table>

#### Equivalence Evaluation

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Equiv?</th>
<th>Total Number</th>
<th>Automatically Decided</th>
<th>Manually proved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>Yes</td>
<td>68</td>
<td>62</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg. Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt; 20 s</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exams</td>
<td>Yes</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg. Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt; 1 s</td>
<td></td>
</tr>
</tbody>
</table>
## Evaluating Cosette

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total #</th>
<th>Average time taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bugs</td>
<td>3</td>
<td>8.3s</td>
</tr>
<tr>
<td>XData</td>
<td>9</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>Exams</td>
<td>5</td>
<td>1.3s</td>
</tr>
</tbody>
</table>

**Rule No. 29:**
400 LOC in [Malecha et al., POPL 10] to 15 LOC in Cosette

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Equiv?</th>
<th>Total Number</th>
<th>Automatically Decided</th>
<th>Manually proved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>No.</td>
<td>Avg. Time</td>
</tr>
<tr>
<td>Rules</td>
<td>Yes</td>
<td>68</td>
<td>62</td>
<td>&lt; 20 s</td>
</tr>
<tr>
<td>Exams</td>
<td>Yes</td>
<td>4</td>
<td>3</td>
<td>&lt; 1 s</td>
</tr>
</tbody>
</table>
Cosette in Action

- Full stack: solver core, web service, online demo, automated grader
- Deployed for UW 344/544 automated grading since 2017
- SIGMOD 2017 Best Demo
- Top 1 Trending Racket Project in GitHub (July, 2017)
Conclusions and Takeaways

**Cosette**: The first practical SQL solver

- A new axiomatic semantics for SQL
- Semi-decision procedure for UCQ under bag/set with ICs
- Integrated interactive theorem proving and constraints solving techniques
- Automated reasoning brought by **Formal Methods + Domain Specific Semantics** leads to more reliable, more optimized future data systems

- Website: cosette.cs.washington.edu