Verifying Data Constraint Equivalence in FinTech Systems

Chengpeng Wang
Gang Fan
Peisen Yao
Fuxiong Pan
Charles Zhang

Prism Group, HKUST
Ant Group
Zhejiang University
Ant Group
Prism Group, HKUST
FinTech Systems

• Offer financial services to consumers or businesses
  • Mobile Payment Apps
  • Peer-to-Peer Lending
  • Personal Finance Apps

• Important to validate the correctness of financial data
Data Constraints in FinTech Systems

- A predicate over table attributes
  - Operation: numeric comparison/computation, substring matching
  - Control flow: sequencing, branch

- Examined upon huge relational tables per minute/hour

```python
1  if(acc.type == 'IN'){
2     amount = acc.in;
3  } else {
4     amount = acc.out;
5  }
6  r1 = amount > 0;
7  r2 = acc.in_id != nil;
8  r3 = acc.out_id != nil;
9  assert(r1 && r2 && r3);
```

- The amount is greater than 0
- Two accounts are not null
Equivalent Data Constraints

• Existence of equivalent data constraints
  • Over 20% of data constraints are equivalent to others in Ant Group

• Root cause
  • Unaware of existing data constraints

• Consequence
  • Waste computation resources
  • Redundant error messages
Resolving Equivalent Data Constraints

• Equivalence searching/clustering
Data Constraint Equivalence Verification

• Problem
  • Given two data constraints r1 and r2, determine whether r1 is semantically equivalent to r2.

• Challenge
  • Achieve high efficiency, soundness, and completeness simultaneously
    • Tens of thousands of data constraints can amplify the efficiency bottleneck.
    • An unsound decision procedure would result in financial loss.
    • An incomplete decision procedure would hide opportunities for optimization.
Existing Effort

• Term rewriting identifies equivalent variants
  • Ensure soundness
  • Discover restrictive forms of equivalent patterns
  • Search vast space when applying rewrite rules

• SMT-based symbolic reasoning verifies logical equivalence
  • Ensure soundness and completeness for decidable fragment
  • SMT solver targets satisfiability problem instead of logical equivalence checking
    • Invoked thousands of times, degrading the efficiency
Motivating Example

• Lexical differences in non-equivalent data constraints
  • Example: (a) and (c)
  • Pose constrain over different table attributes

• Isomorphic structures in equivalent data constraints
  • Example: (b) and (d)
  • Only differ in the order of commutative operands and assertions
EqDAC: Key Idea

- Achieve an efficient decision procedure without “deep” semantic analysis
  - (Over-approximation) Lexical difference-guided input generation refutes data constraint equivalence
  - (Under-approximation) The isomorphic structure proves data constraint equivalence
- Polynomial time!

\[ r' \] and \( r \) are divergent under a given input

\[ r' \] and \( r \) have isomorphic structures
Workflow of EqDAC

Semantic Encoding

Equivalence Reasoning
Divergence Analysis

• Concretize data variables making two formulas evaluate differently, which refutes the equivalence

\[
\varphi_1 = ((t.in > 0 \land \phi_c) \lor (t.out > 0 \land \neg \phi_c)) \land \phi_a \land \phi_o
\]

\[
\varphi_2 = ((t.out > 0 \land \neg \phi_c) \lor (t.new > 0 \land \phi_c)) \land \phi_a \land \phi_i
\]
Isomorphism Analysis

• Apply tree isomorphism algorithm to prove equivalence

```java
if (contains(t.ty, 'IN')){
    assert (t.old == t.new - t.in);
} else {
    assert (t.old == t.new + t.out);
}
assert (t.old != 0);
assert (t.iid != 0);

assert (t.iid != 0);
assert (t.toid != 0);
if (not contains(t.ty, 'IN'))
    cash = t.out + t.new;
else
    cash = t.new - t.in;
assert (cash == t.old);
```
Theoretical Result

- **Theorem 1:** Except for SMT solving, other steps of EqDAC run in polynomial time to $N$, where $N$ is the upper bound of the numbers of AST nodes for the two data constraints.

- **Theorem 2:** If the fragment of data constraints is decidable, EqDAC is sound and complete.
RQ1: Effectiveness

- Identify 26,789 equivalent pairs among 30,801 data constraints in Ant Group
  - 7,842 data constraints can be removed.
  - Error messages caused by data constraints in the same cluster can be merged
    - Extreme case: 48 equivalent data constraints in a cluster
RQ2: Efficiency

• Efficiency clustering
  • Analyze 30,801 data constraints in 2.89 h
  • Peak memory: linear to \#data constraints
  • Time cost: Quadratic to \#data constraints

• Equivalence searching
  • \#Data constraint = 30,801 – 1,000
  • Peak memory: 527.87 MB (max), 527.1 MB (avg)
  • Time cost: 2.50 sec (max), 1.22 sec (avg)
RQ3: Ablation Studies

• Equivalence searching

EqDAC
\[
\text{max: } 2.50 \text{ sec, avg: } 1.22 \text{ sec}
\]

EqDAC-NI
\[
\text{max: } 6.56 \text{ sec, avg: } 2.53 \text{ sec}
\]

EqDAC-NS
\[
\text{max: } 1.48 \text{ sec, avg: } 0.74 \text{ sec}
\]

• Equivalence clustering

<table>
<thead>
<tr>
<th>Variant</th>
<th>Time(h)</th>
<th>Mem(GB)</th>
<th>#Eq Pair</th>
<th>#Redundant</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqDAC-ND</td>
<td>OOT</td>
<td>7.27</td>
<td>141</td>
<td>53</td>
</tr>
<tr>
<td>EqDAC-NI</td>
<td>4.48</td>
<td>6.80</td>
<td>26,789</td>
<td>7,842</td>
</tr>
<tr>
<td>EqDAC-NS</td>
<td>2.13</td>
<td>3.94</td>
<td>25,952</td>
<td>7,296</td>
</tr>
<tr>
<td>EqDAC</td>
<td>2.89</td>
<td>5.01</td>
<td>26,789</td>
<td>7,842</td>
</tr>
</tbody>
</table>

EqDAC-ND: no divergence analysis
EqDAC-NI: no isomorphic analysis
EqDAC-NS: no SMT solving

Miss 37 equivalent data constraints
Conclusion

• Formulate the problem of equivalence data constraint verification
  • Equivalence reasoning upon tens of thousands of programs, i.e., data constraints

• Propose an efficient, sound, and complete decision procedure
  • Leverage lexical difference and isomorphic structures for acceleration

• Provide a fundamental component of equivalence searching and clustering
  • Avoid the redundant checking of equivalent data constraints
Thank you for your listening!
BACKUP
Syntax

• Data constraint syntax

\[\begin{align*}
\mathcal{V} & := v_d \mid x \\
\mathcal{L} & := \{l_i \mid i \geq 1\} \\
\mathcal{A} & := l \mid v_d \mid a_1 \oplus a_2 \\
\mathcal{C} & := a_1 \odot a_2 \mid x_1 \odot x_2 \mid a \odot x \mid x \odot a \mid p(v, l) \mid p(v_1, v_2) \\
\mathcal{B} & := c \mid b_1 \text{ and } b_2 \mid b_1 \text{ or } b_2 \mid \text{not } b \mid \text{ite}_b(c_0, b_1, b_2) \\
\mathcal{S} & := x = a \mid \text{assert}(b) \mid s_1; s_2 \mid \text{ite}_s(c_0, s_1, s_2) \\
\mathcal{R} & := s+ \\
\oplus & := + \mid - \mid \times \mid \div \\
\odot & := > \mid < \mid \geq \mid \leq \mid == \mid \neq \\
\mathcal{P} & := \{\text{prefixOf}, \text{suffixOf}, \text{contains}, \text{equals}\}
\end{align*}\]
Semantics

• An interpretation $I$ is a mapping which maps each data variable $v_d$ to a value in its domain.

\[ I = \{ \text{acc.type} \mapsto \text{'IN'}, \text{acc.in} \mapsto 10, \text{acc.out} \mapsto 0, \text{acc.in_id} \mapsto 1, \text{acc.out_id} \mapsto 2 \} \]

• Given a data constraint $r$, we say $I \models r$, i.e., $I$ is a model of $r$, if and only if all the assertions in $r$ hold under the interpretation $I$.

```java
if(acc.type == 'IN'){
    amount = acc.in;
} else {
    amount = acc.out;
}
r1 = amount > 0;
r2 = acc.in_id != nil;
r3 = acc.out_id != nil;
assert(r1 && r2 && r3);
```
Data Constraint Equivalence

- The data constraints \( r_1 \) and \( r_2 \) are semantically equivalent, denoted by \( r_1 \simeq r_2 \), if and only if for any interpretation \( I \), we have

\[ I \models r_1 \iff I \models r_2 \]

```
if(acc.type == 'IN'){
  amount = acc.in;
} else {
  amount = acc.out;
}
```

1. if \( \text{acc.type} == \text{'IN'} \){
2.     amount = acc.in;
3. } else {
4.     amount = acc.out;
5. }
6. \( r_1 = \text{amount} > 0 \);
7. \( r_2 = \text{acc.in.id} != \text{nil} \);
8. \( r_3 = \text{acc.out.id} != \text{nil} \);
9. assert(r1 && r2 && r3);

```
1. type = 'IN';
2. assert(acc.in_id != nil);
3. assert(acc.out_id != nil);
4. if(acc.type == type){
5.     amount = acc.in;
6. } else {
7.     amount = acc.out;
8. }
9. assert(amount > 0);
```

(a) (b)
Semantic Encoding

• Evaluate user-defined variables

\[ \varphi_1 = ((t.in > 0 \land \phi_c) \lor (t.out > 0 \land \neg \phi_c)) \land \phi_a \land \phi_o \]

\[ \varphi_2 = ((t.out > 0 \land \neg \phi_c) \lor (t.new > 0 \land \phi_c)) \land \phi_a \land \phi_i \]

\begin{align*}
\phi_a &= (t.amt > 0) \\
\phi_o &= (t.oid \neq 0) \\
\phi_i &= (t.iid \neq 0) \\
\phi_c &= \text{contains}(t.ty, 'IN')
\end{align*}
Equivalence Relation Verified by SMT Solver

• Case Study

```c
/* Data constraint 1 */
assert(t.id != t.pid);
assert(ut.oid != ut.iid);
if(t.id == ut.oid){
    assert(t.pid == ut.iid);
} else {
    assert(t.id == ut.iid);
    assert(t.pid == ut.oid);
}
```

```c
/* Data constraint 2 */
if(t.id == ut.iid){
    assert(ut.oid == t.pid);
} else {
    assert(t.id == ut.oid);
    assert(t.pid == ut.iid);
}
assert(ut.iid != ut.oid);
assert(t.pid != t.id);
```

• 1 and 2 are equivalent
• 1 and 2 are equivalent
• 1 and 2 are not equivalent
• 1 ∧ 1 and 2 ∧ 2 are equivalent