Automated Generation of Database Mocks with Symbolic Execution

C. Cornejo[1], A. Borda[1], N. Aguirre[1], M. Frias[2], P. Ponzio[1] & G. Regis[1]
[1]CONICET and University of Rio Cuarto, Argentina
[2]University of Texas at El Paso, USA

4th International KLEE Workshop on Symbolic Execution, April 15-16 2024, Lisbon, Portugal

Software applications and databases

- Most software applications need to persist information, typically via the interaction with databases
- Testing some methods within such applications thus requires producing database states, besides the direct inputs of the method under test
    - In particular, exercising certain program paths may depend on specific data stored in the database

```
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                             + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                          + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                        +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                        + currBook.getId() + “,"
                        + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Software applications and databases

- Most software applications need to persist information, typically via the interaction with databases
- Testing some methods within such applications thus requires producing database states, besides the direct inputs of the method under test
    - In particular, exercising certain program paths may depend on specific data stored in the database

```java
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                             + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                          + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                         +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                         + currBook.getId() + “,"
                         + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Software applications and databases

- Most software applications need to persist information, typically via the interaction with databases
- Testing some methods within such applications thus requires producing database states, besides the direct inputs of the method under test
    - In particular, exercising certain program paths may depend on specific data stored in the database

```java
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                                .executeQuery("SELECT id FROM shelf WHERE id="
                                              + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                          + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                         +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                         + currBook.getId() + “,"
                         + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Software applications and databases

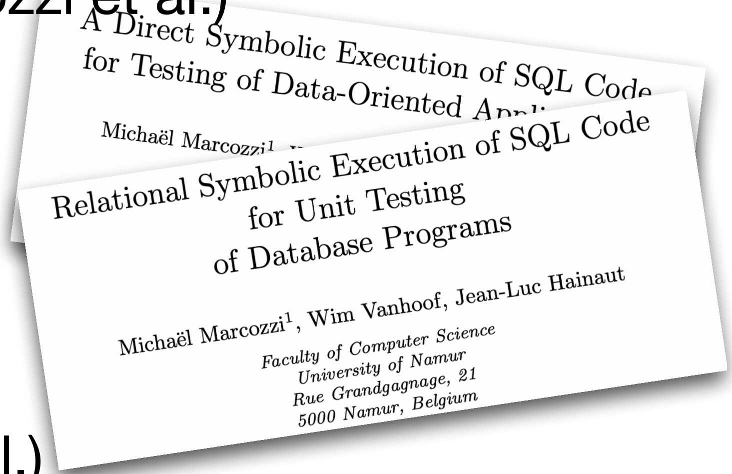
- Most software applications need to persist information, typically via the interaction with databases
- Testing some methods within such applications thus requires producing database states, besides the direct inputs of the method under test
    - In particular, exercising certain program paths may depend on specific data stored in the database

```java
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                      + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                  + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                        +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                        + currBook.getId() + “,"
                        + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

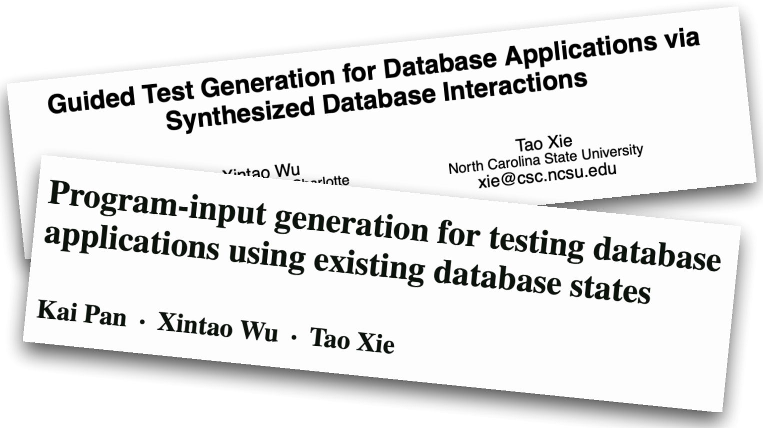
Symbolic execution can help us produce inputs, including database contents, to exercise such paths

Related (previous) Work

- Symbolic execution for DB applications (M. Marcozzi et al.)
  - Implemented for Java + JDBC
  - Ad hoc SE engine
  - **Z3** as backend solver



- Concolic execution for DB applications (T. Xie et al.)
  - Implemented for C# + .NET SqlClient
  - Realized as a PEX extension
  - **Z3** as backend solver



M. Marcozzi, W. Vanhoof, J.-L. Hainaut: *Towards testing of full-scale SQL applications using relational symbolic execution.* CSTVA 2014
M. Marcozzi, W. Vanhoof, J.-L. Hainaut: *Relational symbolic execution of SQL code for unit testing of database programs*. Sci. Comput. Program. (2015)
K. Pan, X. Wu, T. Xie: *Program-input generation for testing database applications using existing database states*. Autom. Softw. Eng. 22(4) (2015)
K. Pan, X. Wu, T. Xie: *Guided test generation for database applications via synthesized database interactions*. ACM Trans. Softw. Eng. Methodol. (2014)

Is Z3 the “right” solver for relational constraints?

Is Z3 the "right" solver for relational constraints?

- Database constraints are inherently relational

Is Z3 the "right" solver for relational constraints?

- Database constraints are inherently relational
- Previous works use Z3 for "relational" constraint solving
  - Essentially, each relation $R \subseteq T_1 \times T_2 \times \ldots \times T_k$ is encoded as uninterpreted function $f_R : T_1 \times T_2 \times \ldots \times T_k \rightarrow$ Bool, and relational constraints as constraints on these functions

Is Z3 the "right" solver for relational constraints?

- Database constraints are inherently relational
- Previous works use Z3 for "relational" constraint solving
  - Essentially, each relation $R \subseteq T_1 \times T_2 \times \ldots \times T_k$ is encoded as uninterpreted function $f_R : T_1 \times T_2 \times \ldots \times T_k \rightarrow$ Bool, and relational constraints as constraints on these functions
- But relational constraint solving in an area on its own
  - *Are we missing relational constraint solving advances by using "non-relational" SMT?*

State-of-the-art in Relational Constraint Solving

State-of-the-art in Relational Constraint Solving

- Bounded relational constraint solving
    - Alloy/KodKod
    - Based on SAT, implements many optimizations

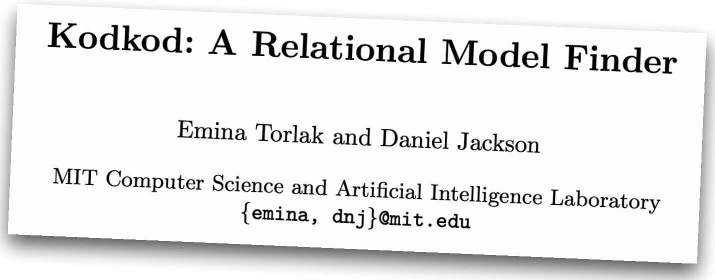
Kodkod: A Relational Model Finder

Emina Torlak and Daniel Jackson

MIT Computer Science and Artificial Intelligence Laboratory
{emina, dnj}@mit.edu

State-of-the-art in Relational Constraint Solving

- Bounded relational constraint solving
    - Alloy/KodKod
    - Based on SAT, implements many optimizations
- Relational constraint solving based on set theory
    - {log}
    - Complete for a theory of finite relations



Kodkod: A Relational Model Finder

Emina Torlak and Daniel Jackson

MIT Computer Science and Artificial Intelligence Laboratory
{emina, dnj}@mit.edu

Solving quantifier-free first-order constraints over finite sets and binary relations

Maximiliano Cristiá · Gianfranco Rossi

State-of-the-art in Relational Constraint Solving

- Bounded relational constraint solving
  - Alloy/KodKod
  - Based on SAT, implements many optimizations
- Relational constraint solving based on set theory
  - {log}
  - Complete for a theory of finite relations
- Relational constraint solving based on an algebraic theory of finite relations
  - Implemented on CVC4
  - Complete for a language over many sorted finite relations (including transitive closure)

Kodkod: A Relational Model Finder

Emina Torlak and Daniel Jackson

MIT Computer Science and Artificial Intelligence Laboratory
{emina, dnj}@mit.edu

Solving quantifier-free first-order constraints over finite sets and binary relations

Maximiliano Cristiá · Gianfranco Rossi

Relational Constraint Solving in SMT

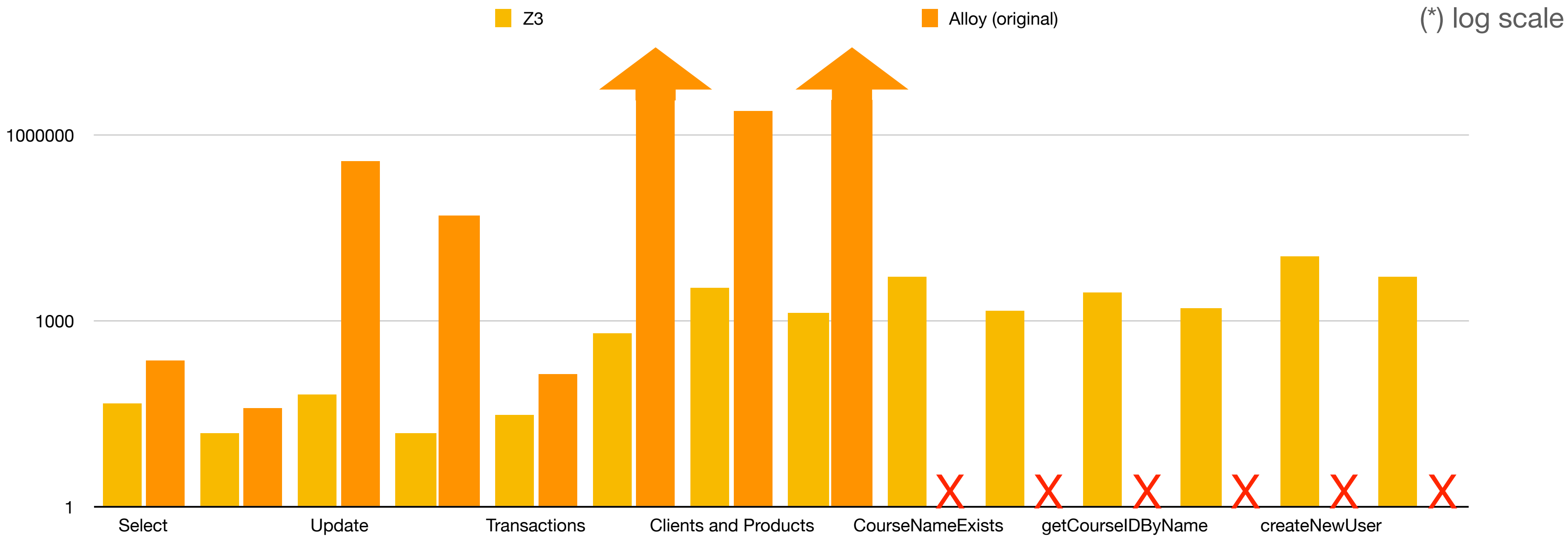
Baoluo Meng[1], Andrew Reynolds[1], Cesare Tinelli[1], and Clark Barrett[2]

[1] Department of Computer Science, The University of Iowa
[2] Department of Computer Science, Stanford University

“Non-relational” SMT vs Relational Constraint Solving

- No comprehensive comparison of existing approaches
- Some studies compare some specific techniques
  - e.g., Alloy vs non-relational SMT, Alloy vs a relational decision procedure, …
  - Some studies are outdated, and use restricted datasets, among other issues

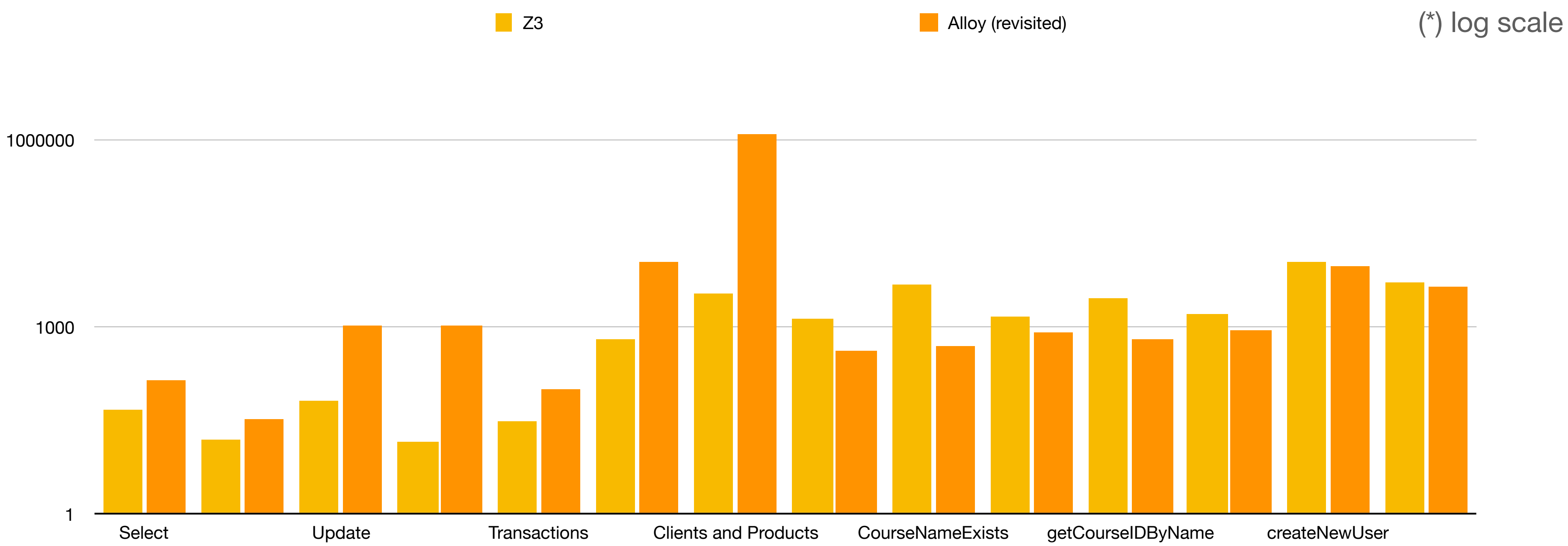
Database application constraints: “Non-relational” SMT vs Relational Constraint Solving



M. Marcozzi, W. Vanhoof, J.-L. Hainaut: *Relational symbolic execution of SQL code for unit testing of database programs*. Sci. Comput. Program. (2015)

Database application constraints: “Non-relational” SMT vs Relational Constraint Solving

Our revisited study



Opportunities with relational constraint solving (in the context of DB Applications)

Opportunities with relational constraint solving (in the context of DB Applications)

- Efficiency
  - SAT bounded relational solving is more efficient for satisfiable constraints (*)

(*) Baoluo Meng, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett: *Relational Constraint Solving in SMT*. CADE 2017

Opportunities with relational constraint solving (in the context of DB Applications)

- Efficiency
  - SAT bounded relational solving is more efficient for satisfiable constraints (*)
- Effectiveness
  - E.g., {log} generates finite representation of all models of a relational formula
    - Kind of “symbolic” interpretation of a formula

(*) Baoluo Meng, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett: *Relational Constraint Solving in SMT*. CADE 2017

Opportunities with relational constraint solving (in the context of DB Applications)

- Efficiency
  - SAT bounded relational solving is more efficient for satisfiable constraints (*)
- Effectiveness
  - E.g., {log} generates finite representation of all models of a relational formula
    - Kind of “symbolic” interpretation of a formula
- Room for combined techniques for relational constraint solving

(*) Baoluo Meng, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett: *Relational Constraint Solving in SMT*. CADE 2017

Interaction between SMT and SAT in DB constraint solving

```
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                             + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                          + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                         +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                         + currBook.getId() + “,"
                         + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Standard SMT

Path condition

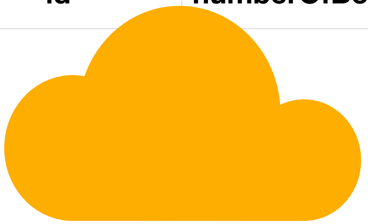
theShelf = x0 $\wedge$ success = true $\wedge$ shelves $\neq \emptyset \wedge \#Shelf > 1$

Relational SAT

Symbolic DB

Shelf

id	numberOfBooks



id is primary key
numberOfBooks > 0

Interaction between SMT and SAT in DB constraint solving

```
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                             + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                             + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                         +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                         + currBook.getId() + “,"
                         + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Standard SMT

Path condition

theShelf = x0 ∧ success = true ∧ shelves ≠ ∅ ∧ $\#Shelf > 1$

Relational SAT

Symbolic DB

Shelf

id	numberOfBooks
x0	n0
y0	n1

U

id is primary key
numberOfBooks > 0

Interaction between SMT and SAT in DB constraint solving

```
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                             + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                          + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id =“
                         +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES (“
                         + currBook.getId() + “,"
                         + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Standard SMT

Path condition

$theShelf = x0 \wedge success = true \wedge shelves \neq \emptyset \wedge \#Shelf > 1$

$y0 \neq x0 \wedge n0 > 0 \wedge n1 > 0$

Relational SAT

Symbolic DB

Shelf

id	numberOfBooks
x0	n0
y0	n1

U

id is primary key
numberOfBooks > 0

Interaction between SMT and SAT in DB constraint solving

```
public List<Integer> addBooks (Connection con, List<Book> newBooks) {
    if (con == null || newBooks == null) throw new IllegalArgumentException();
    int i = 0;
    List<Integer> addedBooks = new ArrayList<Integer>();
    while (i < newBooks.size()) {
        Book currBook = newBooks.get(i);
        int theShelf = shelfForBook(currBook.getId());
        boolean success = true;
        ResultSet shelves = con.createStatement()
                               .executeQuery("SELECT id FROM shelf WHERE id="
                                      + theShelf);
        if (!shelves.next()) {
            con.createStatement().execute("INSERT INTO shelf VALUES ("
                                     + theShelf + ",1)");
        }
        else {
            con.createStatement()
               .execute("UPDATE shelf SET numberOfBooks=numberOfBooks+1 WHERE id ="
                        +theShelf);
        }
        try {
            con.createStatement()
               .execute("INSERT INTO book VALUES ("
                        + currBook.getId() + ","
                        + theShelf +")");
        }
        catch (SQLException e) {
            success = false;
        };
        if (success) {
            con.commit();
            addedBooks.add(currBook.getId());
        }
        else {
            con.rollback();
        }
        i++;
    }
    return addedBooks;
}
```

Standard SMT

Path condition

$\text{theShelf} = \text{x0} \wedge \text{success} = \text{true} \wedge \text{shelves} \neq \emptyset \wedge \#Shelf > 1$

$\text{y0} \neq \text{x0} \wedge \text{n0} > 0 \wedge \text{n1} > 0 \wedge \text{n0'} = \text{n0} + 1$

Relational SAT

Symbolic DB

Shelf

id	numberOfBooks
x0	n0’
y0	n1

U

id is primary key
numberOfBooks > 0

Remarks

- Testing and test generation of db applications based on symbolic execution needs to handle database constraints in combination with path constraints
- Solely using standard SMT prevents us from exploiting advances in relational constraint solving
- Solving techniques that combine standard SMT with relational constraint solving can have an important impact in db application testing