Command-form Coverage for Testing DB Applications

Alessandro Orso
William G.J. Halfond
Georgia Institute of Technology

Supported by NSF awards CCR- 0205422 and CCR-0306372 to GA Tech
A Database Application

User Interface → Application → Database
A Database Application

ResultSet srchBook(String searchString, int searchType, bool showRating, bool grpByRating, bool grpByISBN) {
  String[] srchFields =
  {"title", "author", "isbn"};
  String queryStr =
  "SELECT title, author, description";
  if (showRating)
    queryStr += ", avg(rating) ";
  queryStr += "FROM books WHERE ";
  if (searchType==2)
    queryStr += srchFields[searchType] + " = " + searchString;
  else
    queryStr += searchFields[searchType] + " = " + searchString + "’ ";
  if (grpByRating)
    queryStr += "GROUP BY rating ";
  else if (grpByISBN)
    queryStr += " GROUP BY isbn ";
  return db.executeQuery(queryStr);
}
Faults in Generated DB Commands

1. Misspelled column name “tiitle,”
2. Missing delimiter for a concatenation
3. Lack of “GROUP BY” clause for grouping function
4. Missing delimiter unless one specific line is executed
Faults in Generated DB Commands

1. Misspelled column name “tiitle,”
2. Missing delimiter for a concatenation
3. Lack of “GROUP BY” clause for grouping function
4. Missing delimiter unless one specific line is executed

```java
ResultSet srchBook(String searchString, int searchType, boolean showRating, boolean grpByRating, boolean grpByISBN) {
    String[] srchFields = {
        "tiitle", "author", "isbn"};
    String queryStr =
        "SELECT title, author, description";
    if (showRating)
        queryStr += ", avg(rating) ";
    queryStr += "FROM books WHERE ";
    if (searchType==2)
        queryStr += srchFields[searchType] + 
            " = " + searchString;
    else
        queryStr += searchFields[searchType] + 
            " = " + searchString + ";
    if (grpByRating)
        queryStr += "GROUP BY rating ";
    else if (grpByISBN)
        queryStr += " GROUP BY isbn ";
    return db.executeQuery(queryStr);
}
```
Faults in Generated DB Commands

1. Misspelled column name “tiitle,”
2. Missing delimiter for a concatenation
3. Lack of “GROUP BY” clause for grouping function
4. Missing delimiter unless one specific line is executed

```java
ResultSet srchBook(String searchString, int searchType, boolean showRating, boolean grpByRating, boolean grpByISBN) {
    String[] srchFields = {
        "title", "author", "isbn"};
    String queryStr = 
        "SELECT title, author, description";
    if (showRating)
        queryStr += ", avg(rating) ";
    queryStr += "FROM books WHERE ";
    if (searchType==2)
        queryStr += srchFields[searchType] + 
            " = " + searchString;
    else
        queryStr += searchFields[searchType] + 
            " = '" + searchString + ";"
    if (grpByRating)
        queryStr += "GROUP BY rating ";
    else if (grpByISBN)
        queryStr += " GROUP BY isbn ";
    return db.executeQuery(queryStr);
}
```
Faults in Generated DB Commands

1. Misspelled column name “tiitle,”
2. Missing delimiter for a concatenation
3. Lack of “GROUP BY” clause for grouping function
4. Missing delimiter unless one specific line is executed

```java
ResultSet `srchBook(String searchString, int searchType, boolean showRating, boolean grpByRating, boolean grpByISBN) {
    String[] srchFields = {
        "tiitle", "author", "isbn"};
    String queryStr =
        "SELECT title, author, description";
    if (showRating)
        queryStr += ", avg(rating) ";
    queryStr += "FROM books WHERE ";
    if (searchType == 2)
        queryStr += srchFields[searchType] + 
            " = " + searchString;
    else
        queryStr += srchFields[searchType] + 
            " = '" + searchString + ";"
    if (grpByRating)
        queryStr += "GROUP BY rating ";
    else if (grpByISBN)
        queryStr += " GROUP BY isbn ";
    return db.executeQuery(queryStr);
```

Alex Orso – ASE 2006 – September 2006
Traditional Testing

ResultSet `srchBook(String searchString, int searchType, bool showRating, bool grpByRating, bool grpByISBN) {
String[] srchFields =
{"title", "author", "isbn"};
String queryStr =
"SELECT title, author, description";
if (showRating)
    queryStr += ", avg(rating) ";
queryStr += "FROM books WHERE ";
if (searchType==2)
    queryStr += srchFields[searchType] + 
    " = " + searchString;
else
    queryStr += srchFields[searchType] + 
    " = '" + searchString + "' ";
if (grpByRating)
    queryStr += "GROUP BY rating ";
else if (grpByISBN)
    queryStr += " GROUP BY isbn ";
return db.executeQuery(queryStr);
}
Traditional Testing

ResultSet srchBook(String searchString, int searchType, boolean showRating, boolean grpByRating, boolean grpByISBN) {
    String[] srchFields = 
        {"title", "author", "isbn"};
    String queryStr = 
        "SELECT title, author, description";
    if (showRating)
        queryStr += ", avg(rating) ";
    queryStr += "FROM books WHERE ";
    if (searchType==2)
        queryStr += srchFields[searchType] + 
            " = " + searchString;
    else
        queryStr += srchFields[searchType] + 
            " = " + searchString + ");
    if (grpByRating)
        queryStr += "GROUP BY rating ";
    else if (grpByISBN)
        queryStr += " GROUP BY isbn ";
    return db.executeQuery(queryStr);
}
Traditional Testing

Test Cases
1. ("0123", 2, false, false, true)
2. ("Poe", 1, false, false, false)

Queries Generated
1. SELECT title, author, description FROM books WHERE isbn = 0123 GROUP BY isbn
2. SELECT title, author, description FROM books WHERE author = 'Poe'

Faults Revealed
1. #4
2. #4
Traditional Testing

Test Cases
1. ("0123", 2, false, false, true)
2. ("Poe", 1, false, false, false)
3. ("Poe", 1, true, true, false)

Queries Generated
1. SELECT title, author, description FROM books WHERE isbn = 0123 GROUP BY isbn
2. SELECT title, author, description FROM books WHERE author = 'Poe'
3. SELECT title, author, description, avg(rating) FROM books WHERE author = 'Poe' GROUP BY rating

Faults Revealed
1. #4
2. #4
3. None
Traditional Testing

Test Cases
1. ("0123", 2, false, false, true)
2. ("Poe", 1, false, false, false)
3. ("Poe", 1, true, true, false)

Queries Generated
1. SELECT title, author, description FROM books WHERE isbn = 0123 GROUP BY isbn
2. SELECT title, author, description FROM books WHERE author = 'Poe'
3. SELECT title, author, description, avg(rating) FROM books WHERE author = 'Poe' GROUP BY rating

Faults Revealed
1. #4
2. #4
3. None
Outline

• Motivation and background
• Command-form coverage
• DITTO coverage tool
• Empirical evaluation
• Conclusion and future work
Given a DB application:

**Database command form**: Equivalence class that groups database commands, generated by the application, that differ only in the possible value of their indeterminate parts

**Indeterminate part**: Part of a command form that cannot be determined statically (substrings that correspond to user input)
Given a DB application:

**(Database) command form:** Equivalence class that groups database commands, generated by the application, that differ only in the possible value of their indeterminate parts

Example:

- `SELECT title, author, description FROM books WHERE author = 'Poe'`
- `SELECT title, author, description FROM books WHERE author = 'Capote'`
- `SELECT title, author, description FROM books WHERE author = 'Dante'`
- `=> SELECT title, author, description FROM books WHERE author = '<*>'`
Using the Criterion

1. Compute the command forms
2. Collect coverage information at runtime
3. Determine/report coverage information
1. Compute Command Forms

a. Perform string analysis on the application => char-level NFAs for each query string at each DB interaction point

b. Group SQL keywords and operators in NFAs and determinize => SQL command-form models (DFAs)

c. Assign unique ID to each command form
1. Compute Command Forms

a. Perform string analysis on the application => char-level NFAs for each query string at each DB interaction point

b. Group SQL keywords and operators in NFAs and determinize => SQL command-form models (DFAs)

c. Assign unique ID to each command form
public ResultSet searchBooks(String searchString, int searchType, boolean showRating, boolean groupByRating, boolean groupByISBN) {

    1. String[] searchFields = {"title", "author", "isbn"};
    2. String queryStr = "SELECT title, author, description";
    3. if (showRating) {
        4. queryStr += ", avg(rating) ";
    }
    ...
    14. return database.executeQuery(queryStr);

[Christensen, Møller, and Schwartzbach 2003]
1. Compute Command Forms

a. Perform string analysis on the application => char-level NFAs for each query string at each DB interaction point

b. Group SQL keywords and operators in NFAs and determinize => SQL command-form models (DFAs)

c. Assign unique ID to each command form
Build Command-form Models

Group SQL keywords/operators => SQL command-form models
Build Command-form Models

Group SQL keywords/operators => SQL command-form models

By construction, a path in the model identifies a command form (concatenation of transition labels)

=> The complete set of command forms (i.e., requirements) is given by the set of paths in all models
1. Compute Command Forms

a. Perform string analysis on the application => char-level NFAs for each query string at each DB interaction point

b. Group SQL keywords and operators in NFAs and determinize => SQL command-form models (DFAs)

c. Assign unique ID to each command form
Assign Command-form IDs

Assign unique ID to each command form

- Efficient path-profiling technique => edge labels
- Sum of edge labels along a path gives unique ID for the path (i.e., for the corresponding command form)
  - No need to enumerate all forms
  - Straightforward computation of coverage

[Ball and Larus 1996]
2. Collect Coverage Information

At runtime: Match dynamically-generated queries to command forms (i.e., to paths in the command-form models)

```
SELECT title, author, description,
avg(rating) FROM books WHERE author = 'Poe' GROUP BY rating
```
2. Collect Coverage Information

At runtime: Match dynamically-generated queries to command forms (i.e., to paths in the command-form models)

```
SELECT title, author, description
FROM books
WHERE author = 'Poe'
GROUP BY rating
```

Query:
2. Collect Coverage Information

At runtime: Match dynamically-generated queries to command forms (i.e., to paths in the command-form models)

Query:
```
SELECT title, author, description, avg(rating) FROM books WHERE author = 'Poe' GROUP BY rating
```

Command form:
```
SELECT title, author, description, avg(rating) FROM books WHERE author = '<*>' GROUP BY rating
```
3. Coverage Analysis and Feedback

Coverage = \frac{\text{number of command forms covered}}{\text{total number of command forms}}
The DITTO Coverage Tool

Database-Interaction Testing Tool

Mode 1
- Requirements Generator
  - String Analyzer
  - NFA Models
  - SQL-Model Generator
- Path Analyzer
  - SQL Command Form Models
  - Command Form Information
- Coverage Analyzer
  - Coverage Data
  - Coverage Report

Mode 2
- Instrumenter
  - Instrumented Database Application
- Testers
  - Database Application

Mode 3
- Coverage Monitor
  - Coverage Data
Empirical Evaluation

• **Study 1**: Perform a proof-of-concept evaluation on a commercial application and test suite

• **Study 2**: Investigate whether command-form coverage provides for a more thorough testing of database applications than traditional approaches
Study 1 — Feasibility

• Is the approach feasible?
• What is the command-form coverage achieved by the existing test suite?

Subject: Bookstore
• 27 servlets, ~17 KLOC

Test cases: Test suite from related work
• ~7,000 test cases

Results:
• DITTO was able to compute command forms and measure command-form coverage for the test suite
• Command-form coverage between 1% and 13%
=> Initial evidence that command-form coverage cannot be trivially achieved
Study 2 — Usefulness

• Is command-form coverage useful?
• Does it provide something more than traditional testing?

• Compare with a traditional criterion (branch coverage)
• Ideally, compare fault detection capability, but
  • few data points for real faults
  • difficult to seed faults in an unbiased way

=> Indirect comparison through estimation
Study 2: Protocol

Estimate number of command forms covered by a branch-adequate test suite for Bookstore (B)

- Compute total number of command forms for B
- Identify subset B’ of B involved in building command forms; backward slices from DB interaction points
- Estimate the number of test cases needed to cover all branches in B’; cyclomatic complexity (overestimate)
- Assume each test case covers one command form (overestimate)
- Compare estimated number of command forms covered and total number of command forms
## Results for Study 2

<table>
<thead>
<tr>
<th>Servlet</th>
<th># DIP</th>
<th># command forms</th>
<th>Estimated # comm. forms covered by branch-adequate test suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyInfo</td>
<td>1</td>
<td>6</td>
<td>all</td>
</tr>
<tr>
<td>BookDetail</td>
<td>4</td>
<td>1583</td>
<td>150</td>
</tr>
<tr>
<td>AdminBooks</td>
<td>1</td>
<td>617</td>
<td>31</td>
</tr>
<tr>
<td>OrdersGrid</td>
<td>1</td>
<td>394</td>
<td>26</td>
</tr>
<tr>
<td>ShoppingCart</td>
<td>2</td>
<td>20</td>
<td>all</td>
</tr>
<tr>
<td>AdminMenu</td>
<td>1</td>
<td>1</td>
<td>all</td>
</tr>
<tr>
<td>MembersGrid</td>
<td>1</td>
<td>162</td>
<td>21</td>
</tr>
</tbody>
</table>
Related Work

Specific coverage for DB applications
• Chan and Cheung, 1999
• Kapfhammer and Soffa, 2003
• Suárez-Cabal and Tuya, 2004
• Willmor and Embury, 2005

Static checking of DB applications
• Christensen, Møller, and Schartzbachthe, 2003
• Gould, Su, and Devanbu, 2004

Other paradigms
• McClure and Krüger, 2005
• Cook and Rai, 2005

Test case generation for DB applications
• Frankl et al., 2000, 2004, 2005
• Zhang, Xu, and Cheung, 2001
Conclusion and Future Work

Conclusion

• Technique to adequately test DB applications (in particular, interactions application-DB)
• Approach based on command-form coverage
• DITTO tool that implements the approach
• Initial evaluation
  • Approach is feasible
  • Approach is potentially useful

Future work

• More extensive empirical studies
  • More subjects
  • Direct comparison with other criteria
• Improvement of the technique by leveraging info about the DB (e.g., DB schema)
Questions?