Mining Exception-Handling Rules as Conditional Association Rules

Suresh Thummalapenta sthumma@ncsu.edu
Tao Xie xie@csc.ncsu.edu
NC STATE UNIVERSITY
Computer Science

Problem

- Programmers often reuse APIs of existing frameworks or libraries to achieve high efficiency
  - APIs throw exceptions during runtime errors
    Example: *Session* API of Hibernate framework throws *HibernateException*
  - APIs expect client applications to implement recovery actions after exceptions occur
    Example: Session API expect client application to rollback uncommitted transactions after *HibernateException* occurs
- Failure to handle exceptions results in Fatal issues:
  Example: Database lock won't be released if the transaction is not rolled back

Challenges

- **Lack of Specifications**: Automatic static or runtime verification tools require the knowledge of specifications that must be obeyed while reusing the API methods. These specifications are often not available due to lack of documentation

Example

Scenario 1

```
1.1: ...
1.2: OracleDataSource ods = null; Session session = null;
     Connection conn = null; Statement statement = null;
1.3: logger.debug("Starting update");
1.4: try {
1.5:     ods = new OracleDataSource();
1.6:     ods.setURL("jdbc:oracle:thin:scott/tiger@192.168.1.2:1521:catfish");
1.7:     conn = ods.getConnection();
1.8:     statement = conn.createStatement();
1.9:     statement.executeUpdate("DELETE FROM table1");
1.10:    connection.commit(); }
1.11:    catch (SQLException se)  {
1.13:            logger.error("Exception occurred");  }
1.14: finally {
1.15:    if(statement != null) { statement.close(); }
1.16:    if(conn != null) { conn.close(); }
1.17:    if(ods != null) { ods.close(); } }
1.18: }
```

- Defect : No rollback done when SQLException occurs
- Simple Specification:

  Connection Creation => Connection Rollback
- Formal specification:

  FCc1 FCc2 FCa => FCe1

  FCc1 -> OracleDataSource.getConnection

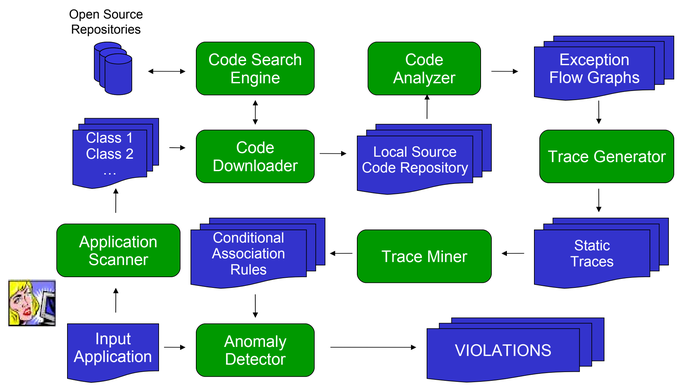
  FCc2 -> Connection.createStatement

  FCa -> Statement.executeUpdate

  FCe1 -> Connection.rollback

Approach

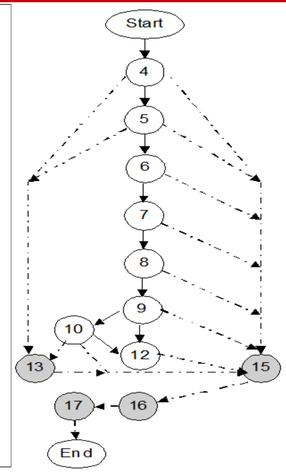
- Code Downloader: Leverage a code search engine to gather relevant code examples for all classes of the input application
- Code Analyser: Analyse code examples using heuristics as code examples are often partial and construct exception flow graphs
- Trace Generator: static traces from flow graphs
- Trace Miner: Generate conditional association rules by mining static traces
- Anomaly Detector: Detect violations of rules in the input application



To appear In Proceedings of 31st International Conference on Software Engineering (*ICSE*), Vancouver, Canada, May 16 – 24, 2009
This work is supported in part by NSF grant CCF-0725190 and ARO grant W911NF-08-1-0443.

Approach

```
2.1: Connection conn = null;
2.2: Statement stmt = null;
2.3: BufferedWriter bw = null; FileWriter fw = null;
2.3: try {
2.4:     fw = new FileWriter("output.txt");
2.5:     bw = BufferedWriter(fw);
2.6:     conn = DriverManager.getConnection("jdbc:pl:db", "ps", "ps");
2.7:     Statement stmt = conn.createStatement();
2.8:     ResultSet res = stmt.executeQuery("SELECT Path FROM Files");
2.9:     while (res.next()) {
2.10:            bw.write(res.getString(1));
2.11:    }
2.12:    res.close();
2.13: } catch(IOException ex) {  logger.error("IOException occurred");
2.14: } finally {
2.15:    if(stmt != null) stmt.close();
2.16:    if(conn != null) conn.close();
2.17:    if (bw != null) bw.close();
2.18: }
```



- " 4 -> 5 " : Normal edge
- " 5 -> 13 " : Exception edge
- A trace for Node 7

  4 -> 5 -> 6 -> 7 -> 15 -> 16 -> 17
- Trace includes 3 parts
  Normal: 4 -> 5 -> 6
  Function Call: 7
  Exception: 15 -> 16 -> 17
- Filtered trace
  6 -> 7 -> 15 -> 16

SDB_1	SDB_2
3,6,9,10	2,3,7,8
3,10,13	2,6,8
9,10,1,19	9,16,13

$SDB_{1,2}$

$3^1,6^1,9^1,10^1,2^2,3^2,7^2,8^2$
$3^1,10^1,13^1,2^2,6^2,8^2$
$9^1,10^1,1^1,19^1,9^2,16^2,13^2$

$SDB_{1,2}$

$3^1,10^1,2^2,8^2$

Association Rule

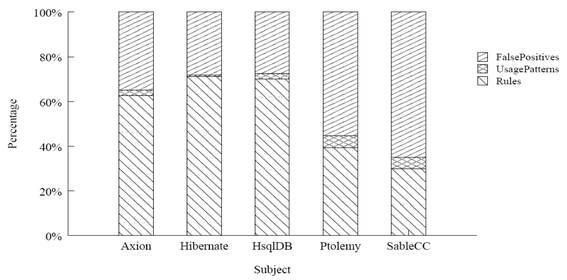
3, 10 => 2,8

Trace Miner

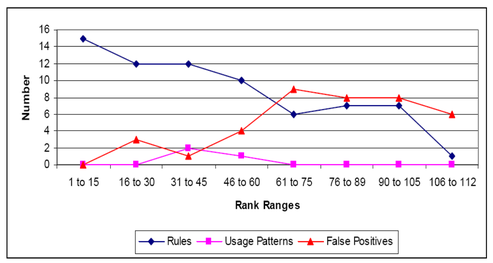
- Input: Two sequence databases SDB1 and SDB2 with 1-to-1 mapping
- Objective: Get association rules
- Annotate sequences to get a single combined database
- Apply frequent sequence mining algorithm
  [Wang and Han ICDE04]
- Transform mined sequences into association rules

Evaluation

Real Rules



Distribution of Rules for Axion



Detected Violations

Subject	#Total Violations	#Violations of first 10 rules	#Defects	#Hints	#FP
Axion 1.0M2	257	19	13	1	5
HsqlDB 1.7.1	394	62	51	0	10
Hibernate 2.0 b4	136	22	12	0	10
Sablecc 2.18.2	168	66	45	7	14
Ptolemy 3.0.2	665	95	39	1	55

- HsqlDB developers responded on the first 10 reported defects
  Accepted 7 defects
  Accepted 3 defects
- Reason given by HsqlDB developers for rejected defects
  "Although it can throw exceptions in general, it should not throw with HsqlDB, So it is fine"