Exploiting Implicit Beliefs to Resolve Sparse Usage Problem in Usage-Based Specification Mining

Samantha S. Khairunnesa  
Iowa State University

Hien N. Nguyen  
The University of Texas at Dallas

Hoan A. Nguyen  
Iowa State University

Hridesh Rajan  
Iowa State University

Supported in part by the NSF grants CCF-15-18897, CNS-15-13263, Grant No. CCF-17-23215, Grant No. CCF-17-23432, and Grant No. CNS-17-23198.
Why do We Need Specification?

- Application programming interfaces (APIs) provided by frameworks and libraries serves as building blocks in modern software development.

- API specification could help developers effectively utilize the APIs and reduce maintenance costs.

- However, currently the efforts needed to write behavioral specifications can be quite high.
How to Correctly Use an API?

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Recent Approaches

- Mining Preconditions of APIs in Large-Scale Code Corpus (Nguyen et al., FSE 2014)

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Recent Approaches

- Mining Preconditions of APIs in Large-Scale Code Corpus (Nguyen et al., FSE 2014)

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Sparse Usage Problem

- Not all important APIs are very widely-used.
- Few (potential) preconditions are present as explicit guard conditions.

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Goal: Overcome Sparse Usage Problem

```
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new IllegalArgumentException("Invalid dataset size.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Goal: Overcome Sparse Usage Problem

How to filter project-specific conditions?

How to fill missing data?
Leverage Implicit Beliefs

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data itemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Leverage Implicit Beliefs

- An implicit belief (IB) is the knowledge implicitly derived from the fact about the code.

```java
public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining

Method, $M_n$

For each $M_i$

CFG

Start

A a = new A();

b >= 0

API(a, b); foo();

Recognize Code Elements

CFG

Start

A a = new A();

b >= 0

API(a, b); foo();

Derive Implicit Beliefs

CFG

Start

A a = new A();

b >= 0

API(a, b); foo();

a != null
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining

For each $M_i$

$A \text{ a } = \text{ new A();}$

$\text{API(a, b);}$

$\text{foo();}$

For each $\text{CFG}_i$

Recognize Code Elements

$\text{A a = new A();}$

$b >= 0$

$\text{API(a, b);}$

$\text{foo();}$

Derive Implicit Beliefs

$\text{A a = new A();}$

$b >= 0$

$\text{API(a, b);}$

$\text{foo();}$

Propagate Implicit Beliefs

Extract Conditions

$b >= 0$

$a != \text{ null}$
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining

For each $M_i$

- $M_1 < API_1, \{EC_1, \ldots\}, \{IB_1, \ldots\}>$
- $M_2 < API_2, \{EC_2, \ldots\}, \{IB_2, \ldots\}>$
- $M_i < API_i, \{b>=0, \ldots\}, \{a!=null, \ldots\}>$
- $M_n < API_n, \{EC_n, \ldots\}, \{IB_n, \ldots\}>$

Extract Conditions

Start

$A a = new A()$

$b>=0$

$API(a, b);$

$foo();$

Derive Implicit Beliefs

Start

$A a = new A()$

$b>=0$

$API(a, b);$

$foo();$

Propagate Implicit Beliefs

Start

$A a = new A()$

$b>=0$

$API(a, b);$

$foo();$

$Iowa State University$
Approach to Exploit Implicit Beliefs to Improve Usage-Based Mining

<table>
<thead>
<tr>
<th>Method, $M_n$</th>
<th>For each $M_i$</th>
<th>Recognize Code Elements</th>
<th>Derive Implicit Beliefs</th>
<th>Propagate Implicit Beliefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A a = \text{new A}();$</td>
<td>$b &gt;= 0$</td>
<td>$\text{API}(a, b);$</td>
<td>$\text{foo}();$</td>
<td>$\text{foo}();$</td>
</tr>
<tr>
<td>$\text{API}(a, b);$</td>
<td>$\text{foo}();$</td>
<td>$\text{API}(a, b);$</td>
<td>$\text{foo}();$</td>
<td>$\text{foo}();$</td>
</tr>
</tbody>
</table>

For each $M_i$:

- $M_1 = \langle \text{API}_1, \{\text{EC}_1, \ldots, \}\{\text{IB}_1, \ldots\}\rangle, \ldots$
- $M_2 = \langle \text{API}_2, \{\text{EC}_2, \ldots, \}\{\text{IB}_2, \ldots\}\rangle, \ldots$
- $M_i = \langle \text{API}_i, \{b>=0, \ldots, \}\{a!=null, \ldots\}\rangle, \ldots$
- $M_n = \langle \text{API}_n, \{\text{EC}_n, \ldots, \}\{\text{IB}_n, \ldots\}\rangle, \ldots$

Mine Preconditions

Precondition Extraction

Precondition Propagation
Why Context is Important?

```java
public static double[] getOLSRegression(XYDataset data, int series) {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
}
```
Why Context is Important?

```java
public static double[] getOLSRegression(XYDataset data, int series) throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ... }
    ...
}```
Why Context is Important?

```java
public static double[] getOLSSmoothRegression(XYDataset data, int series) throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Why Context is Important?

```java
public void computeRegression(XYDataset data) {
    int series = data.getSeriesCount();
    ...
    for (int i = 0; i < series; i++) {
        regression = getOLSRegression(data, i);
        ...
    }
    ...
}

public static double[] getOLSRegression(XYDataset data, int series)
    throws RegressionException {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new RegressionException("Not enough data to calculate regression.");
    }
    ...
    for (int i = 0; i < n; i++) {
        double x = data.getXValue(series, i);
        ...
    }
    ...
```
Classification of Code Elements to Derive Implicit Beliefs
Classification of Code Elements to Derive Implicit Beliefs

- Program
  - Data
  - Computation
    - Control
Classification of Code Elements to Derive Implicit Beliefs

- Program
  - Data
    - Constant
    - Primitive Type
    - Array Type
    - Reference Type
  - Computation
  - Control
Classification of Code Elements to Derive Implicit Beliefs

Program
  - Data
    - Constant
    - Primitive Type
    - Array Type
    - Reference Type
  - Computation

Control

Constant Propagation
B1
Classification of Code Elements to Derive Implicit Beliefs

Program

Data

- Constant
  - Primitive Type
  - Array Type
  - Reference Type

Computation

- Unary Operation
- Binary Operation

Control

Constant Propagation

B1
Classification of Code Elements to Derive Implicit Beliefs

- Program
  - Data
    - Constant
    - Primitive Type
    - Array Type
    - Reference Type
  - Computation
    - Unary Operation
    - Binary Operation
- Control
- Constant Propagation
  - Negation
  - Absolute
Classification of Code Elements to Derive Implicit Beliefs

Program

Data
- Constant
- Primitive Type
- Array Type
- Reference Type

Computation
- Unary Operation
- Binary Operation

Control
- Normal Flow
- Exception Flow

Constant Propagation
- Constant

Unary Operations
- Negation
- Absolute

Iowa State University
Classification of Code Elements to Derive Implicit Beliefs

- **Data**
  - Constant
  - Primitive Type
  - Array Type
  - Reference Type

- **Computation**
  - Unary Operation
  - Binary Operation

- **Control**
  - Normal Flow
  - Exception Flow

- **Structured Control Flow**

- **Non-local Control Flow**

- **Constant Propagation**
  - B1
  - B23
  - B24

- **Unary Operation**
  - Negation
  - Absolute
Classification of Code Elements to Derive Implicit Beliefs

Program
  - Data
    - Constant
    - Primitive Type
    - Array Type
    - Reference Type
  - Computation
    - Unary Operation
    - Binary Operation
  - Control
    - Normal Flow
    - Exception Flow
  - Structured Control Flow
    - Count-controlled Loop
    - Condition-controlled Loop
    - Collection/Array-controlled Loop
  - Non-local Control Flow
  - Short Circuit Evaluation
    - Switch Case
    - Constant Propagation
      - Negation
      - Absolute

IOWA STATE UNIVERSITY
Implemented Code Elements

1. Object Instance Creation
2. Null Dereference
3. Type Comparison
4. Count Controlled Loop
5. Short Circuit Evaluation
6. Local Exception
Evaluation

RQ1: What is the relative improvement over usage-based mining approach?

RQ2: What is the impact of data size on accuracy?

RQ3: How much does each kind of implicit belief (derived from code elements) participate in relative improvement?

RQ4: How much does context sensitivity render the base usage-based approach?
# Data Collection

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects</td>
<td>14,785</td>
</tr>
<tr>
<td>Total Source Files</td>
<td>1,938,865</td>
</tr>
<tr>
<td>Total Classes</td>
<td>2,442,288</td>
</tr>
<tr>
<td>Total Methods</td>
<td>17,378,637</td>
</tr>
<tr>
<td>Total SLOCs</td>
<td>352,312,696</td>
</tr>
<tr>
<td>Total Method Calls</td>
<td>69,374,374</td>
</tr>
</tbody>
</table>
## Data Collection

<table>
<thead>
<tr>
<th>Library</th>
<th>Calls</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jfree.chart (CHART)</td>
<td>35,033</td>
<td>169</td>
</tr>
<tr>
<td>org.jfree.data (DATA)</td>
<td>16,671</td>
<td>125</td>
</tr>
<tr>
<td>org.apache.commons.math (MATH)</td>
<td>34,010</td>
<td>20</td>
</tr>
<tr>
<td>javax.swing (SWING)</td>
<td>479,373</td>
<td>20</td>
</tr>
<tr>
<td>org.eclipse.swt.widget (SWT)</td>
<td>328,174</td>
<td>56</td>
</tr>
<tr>
<td>weka.core (WEKA)</td>
<td>44,047</td>
<td>28</td>
</tr>
<tr>
<td>javax.xml (XML)</td>
<td>274,816</td>
<td>48</td>
</tr>
</tbody>
</table>
Evaluation Metrics

Precision: \[ \frac{\text{No. of True Positive Conditions}}{\text{No. of Total Mined Preconditions}} \]

Recall: \[ \frac{\text{No. of True Positive Preconditions}}{\text{No. of Expected Preconditions}} \]

F-score: \[ \frac{(2 \times \text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \]
RQ1: Relative Improvement

(a) Relative precision

(b) Relative recall.

% Relative accuracy improvement of our approach using implicit beliefs on 7 libraries.
RQ1: Relative Improvement

(a) Relative precision
% Relative accuracy improvement of our approach using implicit beliefs on 7 libraries.

X-axis: Library
Y-axis: % relative improvement

(b) Relative recall.

X-axis: Library
Y-axis: % relative improvement
RQ1: Relative Improvement

% Relative accuracy improvement of our approach using implicit beliefs on 7 libraries.

- Accuracy is improved on all libraries and overall by 32% in precision and 78% in recall.
RQ2: Impact of Data Size on Accuracy

Swing Library

F-score base  F-score combined
RQ2: Impact of Data Size on Accuracy

X-axis: progressive data size (blue bar: base, red bar: combined)
Y-axis: % gain in F-score
RQ2: Impact of Data Size on Accuracy

Swing Library

F-score base  F-score combined

<table>
<thead>
<tr>
<th>Data Size</th>
<th>F-score</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full/512</td>
<td>10%</td>
<td></td>
</tr>
<tr>
<td>Full/256</td>
<td></td>
<td>39%</td>
</tr>
<tr>
<td>Full/128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RQ2: Impact of Data Size on Accuracy

Swing Library

<table>
<thead>
<tr>
<th>Data Size</th>
<th>F-score base</th>
<th>F-score combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full/512</td>
<td>10%</td>
<td>26%</td>
</tr>
<tr>
<td>Full/256</td>
<td>10%</td>
<td>26%</td>
</tr>
<tr>
<td>Full/128</td>
<td>10%</td>
<td>26%</td>
</tr>
<tr>
<td>Full/64</td>
<td>30%</td>
<td>44%</td>
</tr>
<tr>
<td>Full/32</td>
<td>40%</td>
<td>53%</td>
</tr>
<tr>
<td>Full/16</td>
<td>50%</td>
<td>61%</td>
</tr>
<tr>
<td>Full/8</td>
<td>60%</td>
<td>61%</td>
</tr>
<tr>
<td>Full/4</td>
<td>70%</td>
<td>61%</td>
</tr>
<tr>
<td>Full/2</td>
<td>80%</td>
<td>61%</td>
</tr>
<tr>
<td>Full</td>
<td>90%</td>
<td>61%</td>
</tr>
</tbody>
</table>
RQ2: Impact of Data Size on Accuracy

- Impact of data size is non-linear.
- With progressive data, F-score saturates after a certain point.
RQ3: How much does Each Code Element Participate in Relative Improvement?

- Two trends are observed:
  - Precision and recall increase for each library.
    - True positive preconditions increase the recall.
    - Absence of false positive preconditions achieves improvement in terms of precision.
  - Recall increases for all libraries, precision decreases for few libraries.
    - The addition of more true positive preconditions gives better recall.
    - In the cases of the libraries DATA and SWING, we see a decrease in precision because of mining stronger conditions related to some APIs.
RQ4: Context Sensitivity Rendering to Usage-Based Approach

% Comparison of 1-Level Control Flow Analysis (1-CFA), Beliefs with respect to baseline approach in terms of precision and recall for 7 different libraries of interest.

- Both types of components further increases the accuracy.
- The preconditions mined by these two different types of components incorporate some overlapping preconditions.
Conclusion

The notion of implicit beliefs and their usage for precondition mining.

Exploit Implicit Belief to Infer Specifications.

Samantha S. Khairunnesa, sammy@iastate.edu
Tien N. Nguyen, tien.n.nguyen@utdallas.edu
Hoan A. Nguyen, hoan@iastate.edu
Hridesh Rajan, hridesh@iastate.edu

Supported in part by the NSF grants CCF-15-18897, CNS-15-13263, Grant No. CCF-17-23215, Grant No. CCF-17-23432, and Grant No. CNS-17-23198.
Conclusion

The notion of implicit beliefs and their usage for precondition mining.

A catalog of 35 code elements in total that can be used to derive implicit beliefs.

Exploit Implicit Belief to Infer Specifications.

Samantha S. Khairunnesa, sammy@iastate.edu
Tien N. Nguyen, tien.n.nguyen@utdallas.edu
Hoan A. Nguyen, hoan@iastate.edu
Hridesh Rajan, hridesh@iastate.edu

Supported in part by the NSF grants CCF-15-18897, CNS-15-13263, Grant No. CCF-17-23215, Grant No. CCF-17-23432, and Grant No. CNS-17-23198.
Conclusion

The notion of implicit beliefs and their usage for precondition mining.

A catalog of 35 code elements in total that can be used to derive implicit beliefs.

Relative Improvement in Precision: 32%
Relative Improvement in Recall: 78%

Exploit Implicit Belief to Infer Specifications.

Samantha S. Khairunnesa, sammy@iastate.edu
Tien N. Nguyen, tien.n.nguyen@utdallas.edu
Hoan A. Nguyen, hoan@iastate.edu
Hridesh Rajan, hridesh@iastate.edu

Supported in part by the NSF grants CCF-15-18897, CNS-15-13263, Grant No. CCF-17-23215, Grant No. CCF-17-23432, and Grant No. CNS-17-23198.