Example-based Program Querying

Andy Kellens
Johan Brichau
Coen De Roover
Kim Mens
Object mutability pattern

<table>
<thead>
<tr>
<th>ImmutableObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>getContents()</td>
</tr>
<tr>
<td>setContents(String)</td>
</tr>
<tr>
<td>getDate()</td>
</tr>
<tr>
<td>setDate(Date)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blackboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>publishedObjects</td>
</tr>
<tr>
<td>publish(Object)</td>
</tr>
<tr>
<td>getLast()</td>
</tr>
</tbody>
</table>

Component 1

// ...
ImmutableObject o := new ImmutableObject();
Blackboard.publish(o);
// ...
o.setContents("contents");

Component 2

// ...
ImmutableObject o := Blackboard.getLast();
Date date := o.getDate();
// ...
o.setContents("contents");
date.setMonth(5);
Object mutability pattern

**ImmutableObject**
- contents
- date
- getContents()
- setContents(String)
- getDate()
- setDate(Date)

**Blackboard**
- publishedObjects
- publish(Object)
- getLast()

---

**Component 1**

// ...
ImmutableObject o :=
    new ImmutableObject();
Blackboard.publish(o);
// ...
o.setContents("contents");

**Component 2**

// ...
ImmutableObject o :=
    Blackboard.getLast();
Date date := o.getDate();
// ...
o.setContents("contents");
date.setMonth(5);
SOUL

- **Smalltalk Open Unification Language**
  - Prolog-based program query language
  - Hybrid (object-oriented and logic paradigms)

- **Execute queries over program implementations**
  - Complete program structure (structural reflection)

- **Adaptable and specializable reasoning semantics**
  - Open unification

- **Reason about:**
  - Smalltalk, Java
  - C and Cobol (prototypes)
Analysis in the background

Semantic Analysis

```
package foo;
class Bar { ... };

foo.Bar
import foo;
    Bar;
```

Syntactically different but semantically identical names

Static Analysis

```
return foo;
return this.foo;
return this.self().foo;
x = foo; return x;
return o.returnArgument(foo);
```

Syntactically different but possibly overlapping set of objects
Object mutability using Soul

?-c classDeclarationHasName: {ImmutableObject},
?-c hasTransitiveState: ?field,
?-creation isExpression,
?-creation equals: classInstanceCreation(?c,?,?),
?-someClass definesMethod: ?method,
?-method invokes: ?invocation calling: ?publishMethod,
?-publishMethod methodDeclarationHasName: {publish},
?-invocation methodInvocationHasArguments: <?argument>,
?-argument mayAliasWith: ?creation,
?-method invokesTransitive: ?mutatorInv after: ?invocation,
?-mutatorInv methodInvocationHasExpression: ?receiver,
or(?receiver mayAliasWith: ?creation,
  ?receiver mayAliasWith: ?field),
?-mutatorInv calls: ?mutatorMethod,
?-mutatorMethod writesTo: ?field
Object mutability using Soul

Example-based Program Querying

Figure 1. Illustration of the Object Mutability pattern

Figure 2. A direct query to detect undesired object mutations
Object mutability using Soul

```plaintext
?c classDeclarationHasName: {ImmutableObject},
?c hasTransitiveState: ?field,
?creation isExpression,
?creation equals: classInstanceCreation(?c, ?, ?c, ?, ?),
?someClass definesMethod: ?method,
?method invokes: ?invocation calling: ?publishMethod,
?publishMethod methodDeclarationHasName: {publish},
?invocation methodInvocationHasArguments: <?argument>,
?argument mayAliasWith: ?creation,
?method invokesTransitive: ?mutatorInv after: ?invocation,
?mutatorInv methodInvocationHasExpression: ?receiver,
or(?receiver mayAliasWith: ?creation,
    ?receiver mayAliasWith: ?field),
?mutatorInv calls: ?mutatorMethod,
?mutatorMethod writesTo: ?field
```
Object mutability using Soul

?c classDeclarationHasName: {ImmutableObject},
?c hasTransitiveState: ?field,

?creation isExpression,
?creation equals: classInstanceCreation(?, ?, ?c, ?, ?),

?someClass definesMethod: ?method,
?method invokes: ?invocation calling: ?publishMethod,
?publishMethod methodDeclarationHasName: {publish},
?invocation methodInvocationHasArguments: <?argument>,
?argument mayAliasWith: ?creation,

?method invokesTransitive: ?mutatorInv after: ?invocation,
?mutatorInv methodInvocationHasExpression: ?receiver,
or(?receiver mayAliasWith: ?creation,
   ?receiver mayAliasWith: ?field),
?mutatorInv calls: ?mutatorMethod,
?mutatorMethod writesTo: ?field
Object mutability using Soul

Example-based Program Querying
Object mutability using Soul

```java
ImmutableObject o := new ImmutableObject();
Blackboard.publish(o);
// ...
o.setContents("contents");

Component 1

ImmutableObject o := Blackboard.getLast();
Date date := o.getDate();
// ...
o.setContents("contents");
date.setMonth(5);

Component 2

Violation
```

Figure 1. Illustration of the Object Mutability pattern

```prolog
?c classDeclarationHasName: {ImmutableObject},
?c hasTransitiveState: ?field,
?creation isExpression,
?creation equals: classInstanceCreation(?c, ?c, ?c, ?c, ?c),
?someClass definesMethod: ?method,
?method invokes: ?invocation calling: ?publishMethod,
?publishMethod methodDeclarationHasName: {publish},
?invocation methodInvocationHasArguments: <?argument>,
?argument mayAliasWith: ?creation,
?method invokesTransitive: ?mutatorInv after: ?invocation,
?mutatorInv methodInvocationHasExpression: ?receiver,
or(?receiver mayAliasWith: ?creation,
    ?receiver mayAliasWith: ?field),
?mutatorInv calls: ?mutatorMethod,
?mutatorMethod writesTo: ?field
```

Figure 2. A direct query to detect undesired object mutations
Template Queries

```java
if jtStatement(\$statement)\{
  \$x = (\$type) \$e;
\}
```

- a new kind of condition
- multi-directionality
- unification of variables
- grammar production rule
- matching base program elements
- source code with logic variables
Object mutability using Template Query

```
?c classDeclarationHasName: {ImmutableObject},
?c hasTransitiveState: ?field,

jtExpression(?o){ new ?c(?o),

jtMethodDeclaration(?m){
  ?modList ?return ?name(?argList) {
    ?exp.publish(?o);
    ?some.?invocation(?aList); }
},

jtClassDeclaration(?){
  class ? {
    ?mod2List ?return2 ?invocation(?arg2List) {
      ?field = ?exp2;
    }
  }
}
```
Conclusion

- **Template queries:**
  - Querying by example
  - Hide complexity of semantic analysis/static analysis

- **Other examples:**
  - Concurrent modification
  - Detecting of accessors
  - Inadvert invocation on null
  - Constructors referencing uninitialized fields

- De Roover, C., Brichau, J., Noguera, C., D’Hondt, T. and Duchien, L.
  Behavioral Similarity Matching using Concrete Source Code Templates in Logic Queries
  Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Program
  Manipulation (PEPM07 - co-located with POPL07), Nice (France)

- Brichau, J., De Roover, C. and Mens, K.
  Open Unification for Program Query Languages
  Proceedings of the XXVI International Conference of the Chilean Computer Science Society
  (SCCC 2007), Iquique (Chile)