Declarative Visitors to Ease Fine-grained Source Code Mining with Full History on Billions of AST Nodes

Robert Dyer, Hridesh Rajan, and Tien Nguyen
{rdyer,hridesh,tien}@iastate.edu

Iowa State University

The research and educational activities described in this talk was supported in part by the US National Science Foundation (NSF) under grants CCF-13-49153, CCF-13-20578, TWC-12-23828, CCF-11-17937, CCF-10-17334, and CCF-10-18600.
Why mine software repositories?

Learn from the past

Keep doing what works

Empirical validation

To find better designs

Spot (anti-)patterns

Inform the future

What is actually practiced
Consider a task to answer

"How many bug fixes add checks for null?"
Has repository?

- Yes
  - Access repository
  - mine revisions

- No

Foreach project

Output count of all null checks

Find null checks in each source

mine source code

Fixes bug?

Find all Java source files
A solution in Java...

class AddNullCheck {
    static void main(String[] args) {
        /* create and submit a Hadoop job */
    }
    static class AddNullCheckMapper extends Mapper<Text, BytesWritable, Text, LongWritable> {
        static class DefaultVisitor {
            /* define default tree traversal */
        }
        void map(Text key, BytesWritable value, Context context) {
            final Project p = /* read from input */
            new DefaultVisitor() {
                boolean preVisit(Expression e) {
                    if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
                        for (Expression exp : e.expressions)
                            if (exp.kind == ExpressionKind.LITERAL && exp.literal.equals("null")) {
                                context.write(new Text("count"), new LongWritable(1));
                                break;
                            }
                        }
                    }.visit(p);
                }
            }
        }
    }
    static class AddNullCheckReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
        void reduce(Text key, Iterable<LongWritable> vals, Context context) {
            int sum = 0;
            for (LongWritable value : vals)
                sum += value.get();
            context.write(key, new LongWritable(sum));
        }
    }
}

Full program
over 140 lines of code

Uses JSON, SVN, and Eclipse JDT libraries

Uses Hadoop framework

Explicit/manual parallelization
A better solution...

```java
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

Full program **8 lines of code!**

Automatically parallelized!

**No external libraries** needed!

Analyzes **28.8 million** source files in about **15 minutes**!

(only 32 **microseconds** each!)
A better solution...

```java
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

Solution utilizes the Boa framework [Dyer-etal-13]

⇒ This talk: Domain-specific language features for source code mining ⇐

Related Works

- **OO Visitors**
  - GoF, hierarchical, visitor combinators, visitor pattern libraries, recursive traversals

- **DJ, Demeter/Java**

- **Source/program query languages**
  - PQL, JQuery, CodeQuest
Declarative Visitors in Boa

http://boa.cs.iastate.edu/
Basic Syntax

\[ id := \texttt{visitor} \{ \\
\quad \texttt{before \ id:T} \rightarrow \texttt{statement} \\
\quad \texttt{after \ id:T} \rightarrow \texttt{statement} \\
\ldots \\
\}; \]

Execute \texttt{statement} either \texttt{before} or \texttt{after} visiting the children of a node of type \texttt{T}
Basic Syntax

visit(startNode, id);

Starts a visit at the specified startNode using the visitor with the name id
Depth-First Traversal

Provides a default, depth-first traversal strategy

A -> B -> C -> D -> E
Type Lists and Wildcards

visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}

Single type (with identifier)

Attributes of the node available via identifier
Type Lists and Wildcards

```plaintext
visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}
```

Type list (no identifier)

Executes `statement` when visiting nodes of type `T2`, `T3`, or `T4`
Type Lists and Wildcards

visitor {
    before id:T -> statement
    after T2,T3,T4 -> statement
    after _ -> statement
}

Wildcard (no identifier)

Executes statement for any node not already listed in another similar clause (e.g., T but not T2/T3/T4)

Provides default behavior
Type Lists and Wildcards

```plaintext
visitor {
    before id:T   -> statement
    after T2,T3,T4 -> statement
    after _       -> statement
}
```

Types can be matched by at most 1 before clause and at most 1 after clause.
Custom Traversals

A -> E -> B -> C -> D

before n: A -> {
  visit(n.E);
  visit(n.B);
  stop;
}
That’s the language...

what can we do with it?
Mining Revision Pairs

files: map[string] of ChangedFile;

v := visitor {
    before cf: ChangedFile -> {
        if (haskey(files, cf.name)) {
            prevCf = files[cf.name];
            ... # task comparing cf and prevCf
        }
        files[cf.name] = cf;
    }
};

Useful for tasks comparing versions of same file
Mining Snapshots in Time

snapshot: map[string] of ChangedFile;

visit(node, visitor {
    before n: Revision -> if (n.commit_date > TIME) stop;

    before n: ChangedFile ->
    if (n.change == ChangeKind.DELETED)
        remove(snapshot, n.name);
    else
        snapshot[n.name] = n;
});

Computes the snapshot for a given TIME
Mining Snapshots in Time

Previous code provided as domain-specific function

Using that code to visit each file in the snapshot:

```java
visitor { 
  before n: CodeRepository -> { 
    snapshot := getsnapshot(n); 
    foreach (i: int; def(snapshot[i])) 
      visit(snapshot[i]); 
    stop; 
  } 
... 
}
```
Expressiveness

Treasure study reproduction [Grechanik10]
⇒ 22 tasks

Feature study reproduction [Dyer-etal-13b]
⇒ 18 tasks

3 additional tasks (on Boa website)

⇒ See paper for details ⇥
Source Code Comprehension [1/3]

- Controlled Experiment
  - Subjects shown 5 source code mining tasks in Boa
  - Asked to describe (in own words) each task
  - Same tasks shown again (random order)
    - Multiple choice this time
  - Experiment repeated 6 months later in Hadoop
    - Same tasks
    - Same wording for multiple choice answers
Source Code Comprehension [2/3]

Q1  Count AST nodes
Q2  Assert use over time
Q3  Annotation use, by name
Q4  Type name collector, by project and file
Q5  Null check
Source Code Comprehension [3/3]

<table>
<thead>
<tr>
<th>Boa Programs</th>
<th>Hadoop Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
</tr>
<tr>
<td>?</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
</tr>
<tr>
<td>?</td>
<td>+N</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>-Y</td>
</tr>
<tr>
<td>N</td>
<td>+N</td>
</tr>
<tr>
<td>-Y</td>
<td>+N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Y</td>
<td>-Y</td>
<td>N</td>
<td>-Y</td>
<td>-Y</td>
</tr>
<tr>
<td>?</td>
<td>-Y</td>
<td>-Y</td>
<td>-Y</td>
<td>N</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
<td>+N</td>
<td>Y</td>
<td>-Y</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>-Y</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>-Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>-Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>-Y</td>
<td>-Y</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>-Y</td>
<td>-Y</td>
</tr>
<tr>
<td>-Y</td>
<td>+N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

This table shows the comprehension of Boa and Hadoop programs for different questions (Q1 to Q5) with 'N' for no, 'Y' for yes, and '?' for unknown.
### Source Code Comprehension [3/3]

Grading: Use Multiple Choice

<table>
<thead>
<tr>
<th></th>
<th>Boa Programs</th>
<th></th>
<th>Hadoop Programs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
<td>Q4</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>+N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>+N</td>
<td>-Y</td>
<td>-Y</td>
</tr>
</tbody>
</table>

**77.5%**  
**62.5%**
### Source Code Comprehension [3/3]

Grading: Use Free-form

<table>
<thead>
<tr>
<th>Boa Programs</th>
<th></th>
<th>Hadoop Programs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
<td>Q4</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>?</td>
<td>+N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>-Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>+N</td>
<td>-Y</td>
<td>-Y</td>
</tr>
</tbody>
</table>

**Total Score for Boa Programs:** 67.5%

**Total Score for Hadoop Programs:** 30%
Boa with Domain-specific features for mining code

○ **Easy to use** - familiar syntax despite lack of objects
○ Can query **full history** of source files
○ **Fine-grained access** to code down to expressions

Detailed tutorial

**Wed**
10:30 - 12

Demo

**Thurs**
11:15 - 12