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.

What is actually practiced

Keep doing what works

To find better designs

Empirical validation

Spot (anti-)patterns

Why mine software repositories?

Learn from the past → **Inform the future**

Google code

CodePlex
Open Source Project Community

Apache

github
SOCIAL CODING

Java.net

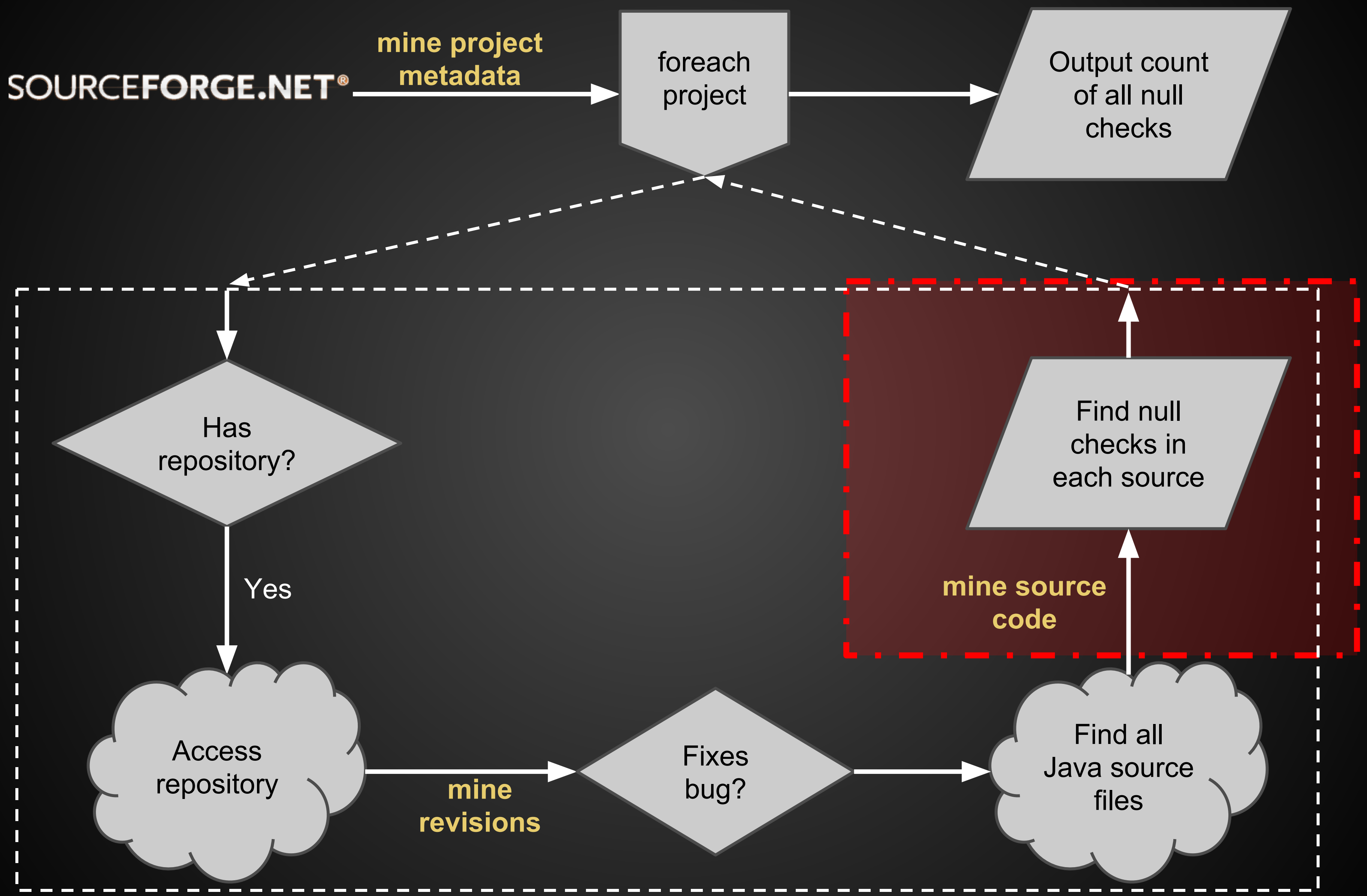
SOURCEFORGE.NET®

Atlassian
bitbucket

launchpad

Consider a task to answer

"How many bug fixes add checks for null?"


SOURCEFORGE.NET®
mine project metadata
foreach project
Output count of all null checks
Has repository?
Yes
Access repository
mine revisions
Fixes bug?
Find all Java source files
mine source code
Find null checks in each source

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));
        }
    }
```

Too much code!
Do not read!

Full program
over 140 lines of code

Uses ***JSON, SVN, and Eclipse JDT libraries***

Uses ***Hadoop framework***

Explicit/manual
parallelization

A better solution...

```
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 ***micro***seconds each!)

A better solution...

```
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 ⇐

[Dyer-etal-13] Robert Dyer, Hoan Nguyen, Hridesh Rajan, and Tien Nguyen. *Boa: A language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories*. ICSE'13

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 := visitor {
   before id:T -> statement
   after  id:T -> statement
   ...
};
```

Execute **`statement`** either **`before`** or **`after`** visiting the children of a node of type **`T`**

Basic Syntax

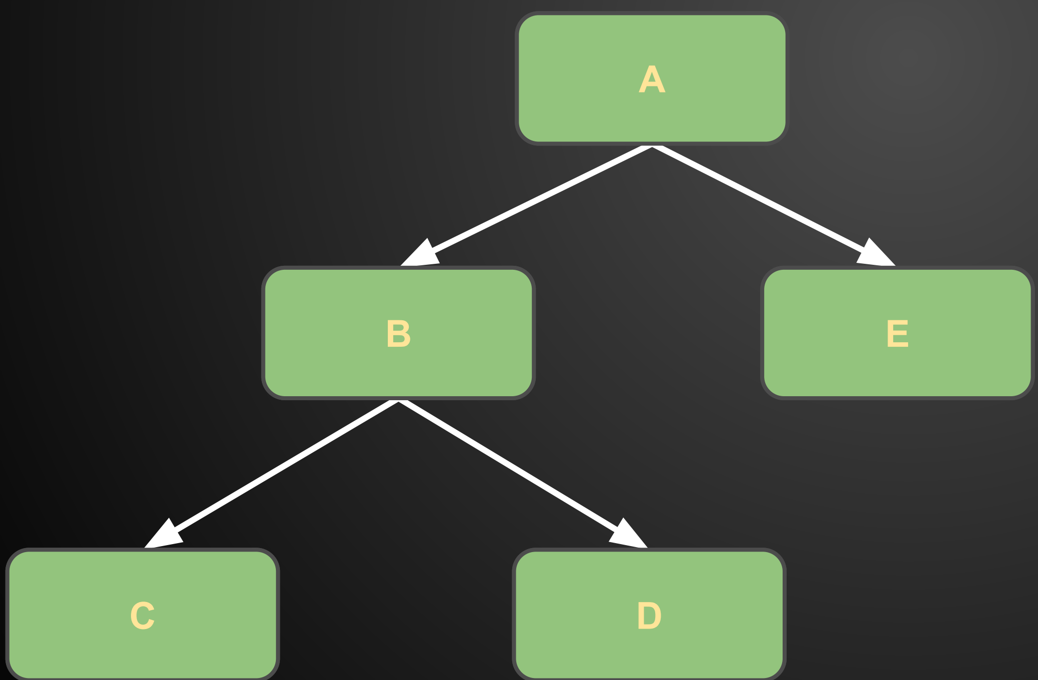
```
visit(startNode, id);
```

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




```
before A -> statement
before B -> statement
before C -> statement
after  C -> statement
before D -> statement
after  D -> statement
after  B -> statement
before E -> statement
after  E -> statement
after  A -> statement
```

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

```
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

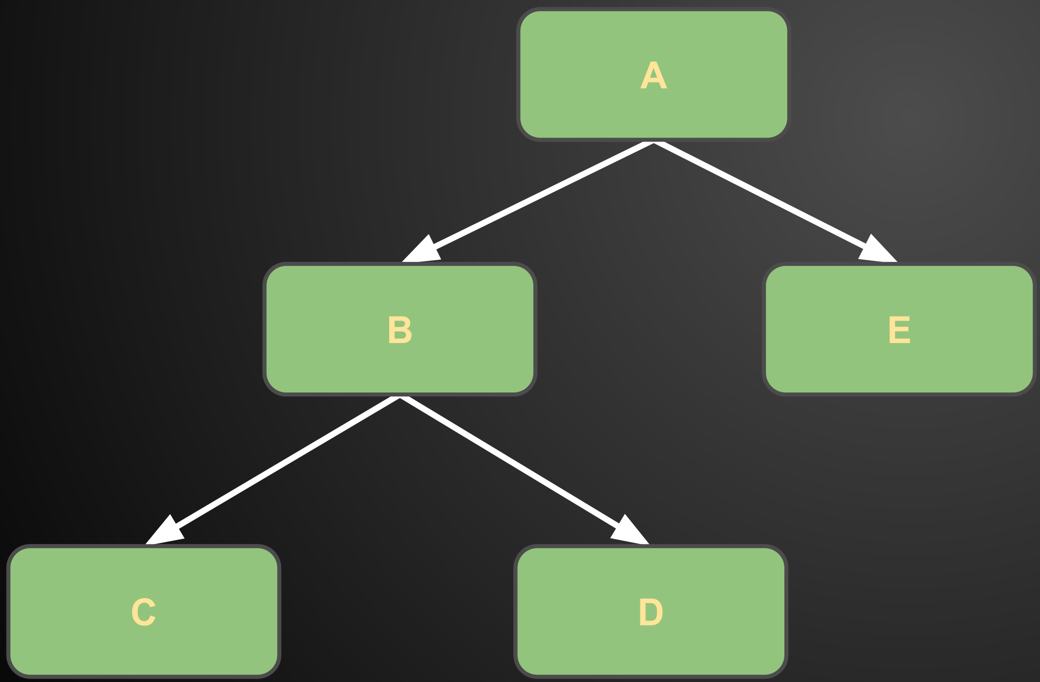
Type Lists and Wildcards

```
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:

```
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]

Boa Programs

Q1	Q2	Q3	Q4	Q5
N	Y	Y	Y	Y
-Y	Y	Y	Y	Y
?	Y	Y	Y	Y
-Y	Y	Y	Y	Y
?	+N	Y	Y	N
N	Y	Y	Y	-Y
N	-Y	Y	Y	Y
N	+N	-Y	-Y	Y

Hadoop Programs

Q1	Q2	Q3	Q4	Q5
-Y	-Y	N	-Y	-Y
?	-Y	-Y	-Y	N
-Y	Y	+N	Y	-Y
N	Y	N	-Y	N
N	-Y	N	N	N
-Y	Y	Y	Y	Y
N	N	Y	-Y	-Y
-Y	+N	Y	N	Y

Source Code Comprehension [3/3]

Grading: Use Multiple Choice

Boa Programs

Q1	Q2	Q3	Q4	Q5	Total
N	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	100%
?	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	100%
?	+N	Y	Y	N	40%
N	Y	Y	Y	-Y	80%
N	-Y	Y	Y	Y	80%
N	+N	-Y	-Y	Y	60%

77.5%

Hadoop Programs

Q1	Q2	Q3	Q4	Q5	Total
-Y	-Y	N	-Y	-Y	80%
?	-Y	-Y	-Y	N	60%
-Y	Y	+N	Y	-Y	80%
N	Y	N	-Y	N	40%
N	-Y	N	N	N	20%
-Y	Y	Y	Y	Y	100%
N	N	Y	-Y	-Y	60%
-Y	+N	Y	N	Y	60%

62.5%

Source Code Comprehension [3/3]

Grading: Use Free-form

Boa Programs

Q1	Q2	Q3	Q4	Q5	Total
N	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	80%
?	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	80%
?	+N	Y	Y	N	60%
N	Y	Y	Y	-Y	60%
N	-Y	Y	Y	Y	60%
N	+N	-Y	-Y	Y	40%

67.5%

Hadoop Programs

Q1	Q2	Q3	Q4	Q5	Total
-Y	-Y	N	-Y	-Y	0%
?	-Y	-Y	-Y	N	0%
-Y	Y	+N	Y	-Y	60%
N	Y	N	-Y	N	20%
N	-Y	N	N	N	0%
-Y	Y	Y	Y	Y	80%
N	N	Y	-Y	-Y	20%
-Y	+N	Y	N	Y	60%

30%

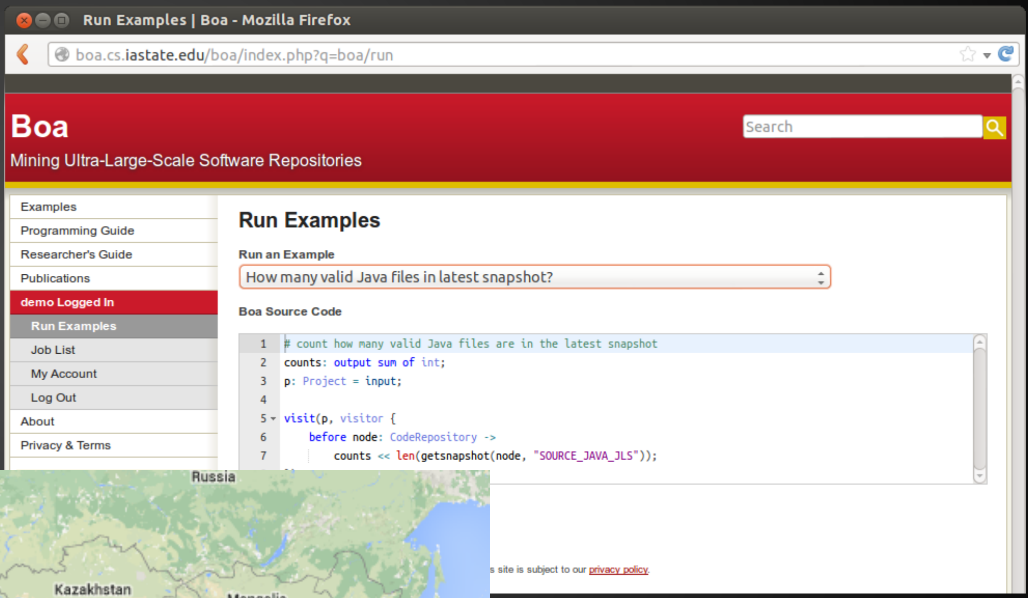
Boa with Domain-specific features for mining code

- **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