Big Code Mining in Boa

Robert Dyer   Hoan Nguyen   Hridesh Rajan   Tien Nguyen

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 CNS-15-12947, CNS-15-13263, CCF-13-49153.
Why mine software repositories?

Learn from the past
What is actually practiced
Keep doing what works

To find better designs
Empirical validation

Spot (anti-)patterns

Inform the future

Learn from the past  
Inform the future
Open source repositories

1,000,000+ projects

1,000,000,000+ lines of code

10,000,000+ revisions

3,000,000+ issue reports
Open source repositories

1,000,000+ projects

What is the most used PL?

1,000,000,000+ lines of code

How many methods are named "test"?

10,000,000+ revisions

How many words are in log messages?

3,000,000+ issue reports

How many issue reports have duplicates?
Consider a task to answer

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

Yes

Access repository

mine revisions

Find null checks in each source

mine source code

Output count of all null checks

foreach project

Find all Java source files

Fixed bug?

Yes
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

Too much code! Do not read!
The Boa language and data-intensive infrastructure

http://boa.cs.iastate.edu/
Design goals

- Easy to use
- Scalable and efficient
- Reproducible research results
Design goals

- Easy to use
  - Simple language
  - No need to know details of
    - Software repository mining
    - Data parallelization
Design goals

- Study *millions* of projects
- Results in minutes, not days
Among these threats, we may encounter: lack of independent validation of the presented results; changes in practices, tools or methodologies; or generalization of knowledge although a limited amount of case studies have been performed.

A simple taxonomy of replication studies provides us with two main groups: exact replications and conceptual replications. The former ones are those in “which the procedures of an experiment are followed as closely as possible to determine whether the same results can be obtained”, while the latter ones are those “in which the same research question or hypothesis is evaluated by using a different experimental procedure, i.e. many or all of the variables described above are changed” [2]. In this paper, we will target exact replications as the requirements that have to be met to perform an exact replication are more severe, and in general make a conceptual replication feasible.

We are focusing in this paper on potential replication as we have actually not replicated any of the studies presented in the papers under review. Our aim in this sense is more humble: we want to check if the necessary conditions that make a replication possible are met.

The rest of the paper is structured as follows: in the next section, the method used for this study is presented. Then some general remarks on the MSR conference are given, in view for reviewing some of the papers that are
Boa architecture

**Boa Language**
- MapReduce
- Domain-specific Types
- Visitors

**Boa's Compiler**
- MapReduce
- Quantifiers
- User Functions
- Visitors
- Cached Data input reader
- Runtime

**Boa's Data Infrastructure**
- SF.net
- Replicator
- Caching Translator
- Local Cache

Recall: 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!)
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;
    count <<= 1;
});
Design goals

Easy to use

Scalable and efficient

Reproducible research results
Let's see it in action!

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

Username: mbds
Password: summerschool (no space)
Why are we waiting for results?

Program is analyzing...

699,331 projects
494,158 repositories
15,063,073 revisions
69,863,970 files
18,651,043,238 AST nodes
Let's check the results!

<<demo>>
Domain-specific types

http://boa.cs.iastate.edu/docs/dsl-types.php

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;
});

Abstracts details of how to mine software repositories
Domain-specific types
http://boa.cs.iastate.edu/docs/dsl-types.php

Project

id : string
name : string
description : string
homepage_url : string
programming_languages : array of string
licenses : array of string
maintainers : array of Person
... 

code_repositories : array of CodeRepository
Domain-specific types

http://boa.cs.iastate.edu/docs/dsl-types.php

**CodeRepository**

- `url` : string
- `kind` : `RepositoryKind`
- `revisions` : array of `Revision`

**Revision**

- `id` : `int`
- `author` : `Person`
- `committer` : `Person`
- `commit_date` : `time`
- `log` : `string`
- `files` : array of `File`

**File**

- `name` : `string`
- `kind` : `FileKind`
- `change` : `ChangeKind`
Domain-specific functions

https://boa.cs.iastate.edu/docs/dsl-functions.php

```haskell
hasfiletype := function (rev: Revision, ext: string) : bool {
    exists (i: int; match(format(`\.%s$`, ext), rev.files[i].name))
        return true;
    return false;
}
```

Mines a revision to see if it contains any files of the type specified.
isfixingrevision := function (log: string) : bool {
  if (match(`\bfix(s|es|ing|ed)?\b`, log)) return true;
  if (match(`\b(error|bug|issue)(s)\b`, log)) return true;
  return false;
};

Mines a revision log to see if it fixed a bug.
User-defined functions

http://boa.cs.iastate.edu/docs/user-functions.php

id := function (a₁: t₁, ..., aₙ: tₙ) [:: ret] {
    ... # body
    [return ...;]
};

Return type is optional

- Allows for complex algorithms and code re-use
- Users can provide their own mining algorithms
Quantifiers
http://boa.cs.iastate.edu/docs/quantifiers.php

foreach (i: int; condition...) 
  body;

For each value of i,

  if condition holds
  then
    run body (with i bound to the value)
Quantifiers

exists (i: int; condition...)  
    body;

For some value of \(i\),

if condition holds
    then
run body once (with \(i\) bound to the value)
Quantifiers
http://boa.cs.iastate.edu/docs/quantifiers.php

```plaintext
ifall (i: int; condition...)  
   body;
```

For all values of i,

if `condition` holds
then
run `body` once (with i not bound)
Output and aggregation

- Output defined in terms of predefined data aggregators
  - sum, set, mean, maximum, minimum, etc
- Values sent to output aggregation variables
- Output can be indexed

```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;
})
```
Declarative Visitors in Boa

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

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

visit(startNode, id);

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

```java
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;
}
Putting it all together
(implementing the motivating example)

http://boa.cs.iastate.edu/
Recall the task is to answer

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

Yes

Access repository

mine revisions

Foreach project

Output count of all null checks

Find null checks in each source

mine source code

Find all Java source files

Yes

Fixes bug?

Yes
Step 1: Declare input and visitor

```text
p: Project = input;

visitor {

};
```
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    # null == expr OR expr == null
    # null != expr OR expr != null
};
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    #   null == expr OR expr == null
    #   null != expr OR expr != null
    before exp: Expression ->
};
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    #  null == expr OR expr == null
    #  null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
};
Step 2: Finding null checks

p: Project = input;

visitor {
    # look for expressions of the form:
    #   null == expr OR expr == null
    #   null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
};
Step 3: Output null checks count

p: Project = input;

NullChecks: output sum of int;

visitor {
    # look for expressions of the form:
    #   null == expr OR expr == null
    #   null != expr OR expr != null
    before exp: Expression ->
        if (exp.kind == ExpressionKind.EQ || exp.kind ==
            ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
            NullChecks << 1;
};
Step 4: Name and call the visitor

p: Project = input;
NullChecks: output sum of int;

nullCheckVisitor :=
   visitor {
      # look for expressions of the form:
      # null == expr OR expr == null
      # null != expr OR expr != null
      before exp: Expression ->
         if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
               NullChecks << 1;
   };

visit(p, nullCheckVisitor);
Let’s see it in action!

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

nullCheckVisitor :=
    visitor {
        # look for expressions of the form:
        # null == expr OR expr == null
        # null != expr OR expr != null
        before exp: Expression ->
            if (exp.kind == ExpressionKind.EQ || exp.kind == ExpressionKind.NEQ)
                exists (i: int; isliteral(exp.expressions[i], "null"))
                    NullChecks <<= 1;
    };

visit(p, nullCheckVisitor);
```

http://boa.cs.iastate.edu/boa/
Has repository?

- Yes:
  - Access repository
  - mine revisions
  - mine source code
  - Fixes bug?
    - Yes:
      - Find all Java source files
    - Fixes bug?
      - mine source code

- No:
  - mine project metadata
  - foreach project
  - Output count of all null checks
  - Find null checks in each source
Recall the visitor

nullCheckVisitor :=

```java
visitor {
  # look for expressions of the form:
  # null == expr OR expr == null
  # null != expr OR expr != null
  before exp: Expression ->
      if (exp.kind == ExpressionKind.EQ
           || exp.kind == ExpressionKind.NEQ)
          exists (i: int; isliteral(exp.expressions[i], "null"))
              NullCheck << 1;
}
```
Step 5: Make visitor more specific

```plaintext
nullCheckVisitor := visitor {
  before stmt: Statement ->
    # increase the counter if there is an IF statement
    if (stmt.kind == StatementKind.IF)
      visit(stmt.expression, visitor {
        # where the boolean condition is of the form:
        #   null == expr OR expr == null
        #   null != expr OR expr != null
        before exp: Expression ->
          if (exp.kind == ExpressionKind.EQ
              || exp.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(exp.expressions[i], "null"))
              NullCheck << 1;
      });
};
```
Step 6: Make visitor reusable

count := 0;
nullCheckVisitor := visitor {
    before stmt: Statement ->
        # increase the counter if there is an IF statement
        if (stmt.kind == StatementKind.IF)
            visit(stmt.expression, visitor {
                # where the boolean condition is of the form:
                # null == expr OR expr == null
                # null != expr OR expr != null
                before exp: Expression ->
                    if (exp.kind == ExpressionKind.EQ
                        || exp.kind == ExpressionKind.NEQ)
                        exists (i: int; isliteral(exp.expressions[i], "null"))
                        count++;
            });
};
files: map[string] of ChangedFile;

visit(p, visitor {

    before cf: ChangedFile -> {
        if (haskey(files, node.name))
            analysis(cf, files[cf.name]); # TODO

        if (cf.change == ChangeKind.DELETED)
            remove(files, cf.name);
        else
            files[cf.name] = cf;
        stop;
    }
});
Step 8: Check for bug fixes

```javascript
isfixing := false;
files: map[string] of ChangedFile;

visit(p, visitor {
  before rev: Revision -> isfixing = isfixing_revision(rev.log);
  before cf: ChangedFile -> {
    if (haskey(files, node.name) && isfixing)
      analysis(cf, files[cf.name]); # TODO

    if (cf.change == ChangeKind.DELETED)
      remove(files, cf.name);
    else
      files[cf.name] = cf;
    stop;
  }
});
```
Step 9: Define the analysis

```
analysis := function(cf: ChangedFile, prevCf: ChangedFile) {
    # count how many null checks were previously in the file
    count = 0;
    visit(prevCf, nullCheckVisitor);
    last := count;
    
    # count how many null checks are currently in the file
    count = 0;
    visit(cf, nullCheckVisitor);
    
    # if there are more null checks, output
    if (count > last)
        NullCheck << 1;
};
```
This solves the ENTIRE task!

Let’s see it in action!

http://boa.cs.iastate.edu/boa/
Design goals

Easy to use

Scalable and efficient

Reproducible research results
Efficient execution
Efficient execution
Scalability of input size
Scalability of input size
### Scales to more cores

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3</td>
<td>92</td>
</tr>
<tr>
<td>B.6</td>
<td>102</td>
</tr>
<tr>
<td>C.1</td>
<td>90</td>
</tr>
<tr>
<td>D.5</td>
<td>90</td>
</tr>
</tbody>
</table>

Legend:
- **1 map**
- **2 maps**
- **4 maps**
- **8 maps**
- **16 maps**
- **32 maps**
Design goals

Easy to use

Scalable and efficient

Reproducible research results
Reproducing MSR results

Robles, MSR'10

2/154 experimental papers "replication friendly."

48 due to lack of published data
Prior research results are difficult (or impossible) to reproduce.

Boa makes this easier!
Controlled Experiment

- Published artifacts (Boa website):
  - Boa source code
  - Dataset used (timestamp of data)
  - Results

<table>
<thead>
<tr>
<th>Expert</th>
<th>Education</th>
<th>Intro Time</th>
<th>Task 1 Task</th>
<th>Task 1 Time</th>
<th>Task 2 Task</th>
<th>Task 2 Time</th>
<th>Task 3 Task</th>
<th>Task 3 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Post-doc</td>
<td>6</td>
<td>B.1</td>
<td>1</td>
<td>B.6</td>
<td>4</td>
<td>B.9</td>
<td>3</td>
</tr>
<tr>
<td>Yes</td>
<td>PhD</td>
<td>5</td>
<td>A.1</td>
<td>3</td>
<td>B.6</td>
<td>2</td>
<td>B.7</td>
<td>6</td>
</tr>
<tr>
<td>No</td>
<td>PhD</td>
<td>4</td>
<td>B.6</td>
<td>1</td>
<td>B.10</td>
<td>4</td>
<td>B.9</td>
<td>4</td>
</tr>
<tr>
<td>No</td>
<td>PhD</td>
<td>4</td>
<td>A.2</td>
<td>2</td>
<td>B.6</td>
<td>2</td>
<td>D.5</td>
<td>4</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>4</td>
<td>A.1</td>
<td>4</td>
<td>B.6</td>
<td>1</td>
<td>D.3</td>
<td>2</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>3</td>
<td>B.6</td>
<td>2</td>
<td>C.1</td>
<td>2</td>
<td>D.4</td>
<td>10</td>
</tr>
<tr>
<td>No</td>
<td>MS</td>
<td>6</td>
<td>A.1</td>
<td>2</td>
<td>B.7</td>
<td>3</td>
<td>B.10</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>BS</td>
<td>2</td>
<td>A.2</td>
<td>2</td>
<td>D.1</td>
<td>2</td>
<td>D.3</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 16. Study results. All times given in minutes.
Let's reproduce some prior results!

http://boa.cs.iastate.edu/examples/

Username: mbds
Password: summerschool (no space)
Boa

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

- Domain-specific language and infrastructure for software repository mining that is:
  - Easy to use
  - Efficient and scalable
  - Amenable to reproducing prior results
What is Program Analysis?

- Analyzing programs
- Developing algorithms and tools for understanding certain program behavior

```c
void foo(int n) {
    int j = 0;
    while (n > 0) {
        if (j % 2 == 0)
            j++;
        else
            j--;
        n--;
    }
    assert "j is positive"
}
```

<table>
<thead>
<tr>
<th>Program Analysis Tool</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- “j” is positive</td>
</tr>
<tr>
<td></td>
<td>- “j” is negative</td>
</tr>
<tr>
<td></td>
<td>- “j” can be anything</td>
</tr>
</tbody>
</table>

Is “j” positive?
Applications of Program Analysis

- Finding bugs
  - In the syntax, functionality, etc.

- Security
  - Does the program leak private user data?

- Verification
  - Does the program always behave according to its specification?

- Optimization
  - Which variables should be kept in the register for faster memory access?
  - Is it safe to execute loop iterations in parallel?
Two Flavors of Program Analysis

● Static
  ○ Analyzes source code artifacts

● Dynamic
  ○ Analyzes program while it is running

● Trade off
  ○ Static
    ■ Reasons about all executions, but less precise
  ○ Dynamic
    ■ Limited to observed executions, but more precise
Static Program Analysis

- Is about analyzing source code artifacts
  - Intermediate representations
    - Abstract syntax trees (ASTs)
    - Control flow graphs (CFGs)
    - Program dependence graphs (PDGs)
    - Call graph (CGs)
Control and Data Flow Analysis

- Performed over Control Flow Graph (CFG) representation of the program

- Control Flow Graph (CFG)
  - A directed graph $G = (N, E)$ with nodes representing program statements and edges representing the flow of control
void foo(int n) {
    int j = 0;
    while (n > 0) {
        if (j % 2 == 0)
            j++;
        else
            j--;
        n--;
    }
    assert "j is positive"
}
Data Flow Analysis

- Computing certain information at every node of the CFG
  - For instance, Live variable analysis computes a set of live variables at each program statement
  - A variable is live at a program point if it has some use after the program point
Data Flow Analysis

Essentials for describing a data flow analysis

- What information to compute?
- What are the initial values at nodes?
- Analysis direction?
  - FORWARD or BACKWARD
- How to merge two or more values at join nodes?
  - Intersection
  - Union
Data Flow Analysis

- Live variable analysis
  - What information to compute?
    - A set of variables
  - What are the initial values at nodes?
    - Empty set
  - Analysis direction?
    - BACKWARD
  - How to merge two or more values at join nodes?
    - Set union
Control and Data Flow Analysis in Boa
Control/Data Flow Analysis in Boa

- Core concept
  - An analysis is a collection of traversals
  - A traversal visits every node in the CFG in a certain order
Control Flow Graph (CFG) in Boa

**Accessing CFG of a method**

```plaintext
visit(input, visitor {
   before method: Method -> {
      g: CFG;
      g = getcfg(method);
   }
});
```
Control Flow Graph (CFG) in Boa

Accessing data from CFG Nodes

```c
int traversal(CFGNode n) {
  int nodeid = n.id;

  if (n.stmt != null) {
    stmt = n.stmt;
  }

  if (n.expr != null) {
    expr = n.expr;
  }

  foreach (j in n.predecessors) {
    n.successors[j] = #n.successors[j]
  }
}
```
Language Extensions
For Program Analysis
Language Extensions for Program Analysis

- Traversal

```plaintext
  t1 := traversal(n: CFGNode): T {
      body
  }
```

- **t1** is the traversal variable
- **n** is the node currently being visited
- **body** may contain any valid Boa statement
- **T** is the type of the return value
Language Extensions for Program Analysis

- **Traverse**

\[ \text{traverse}(g, d, k, t, fp) \]

- **g** is the graph
- **d** is the traversal direction
  - FORWARD or BACKWARD
- **k** is the traversal kind
- **t** is the name of the traversal function
- **fp** is the fixpoint function, if any
Language Extensions for Program Analysis

- Traverse

\[ \text{traverse}(g, d, k, t, fp) \]

\( k \) is the traversal kind

<table>
<thead>
<tr>
<th>Traversal Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Depth-first</td>
</tr>
<tr>
<td>POSTORDER</td>
<td>Post-order</td>
</tr>
<tr>
<td>REVERSEPOSTORDER</td>
<td>Reverse post-order</td>
</tr>
<tr>
<td>WORKLIST_POSTORDER</td>
<td>Worklist with post-order</td>
</tr>
<tr>
<td>WORKLIST_REVERSEPOSTORDER</td>
<td>Worklist with reverse post-order</td>
</tr>
<tr>
<td>ITERATIVE</td>
<td>Any order</td>
</tr>
<tr>
<td>HYBRID</td>
<td>When you are not sure</td>
</tr>
</tbody>
</table>
Language Extensions for Program Analysis

- **Fixpoint**

  ```
  fp := fixp(curr, prev: T): bool {
    body
  }
  ```

  - **fp** is the name of the fixpoint function
  - **curr, prev** are two values that are compared
  - **bool** a fixpoint function must always return a boolean
  - **body** may contain any valid boa statement
Accessing Traversal Results

- `getvalue(node)`
  - Get result of a node in the current traversal

- `getvalue(node, myTraversal)`
  - Get result of a node in some other traversal
Boa

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