Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool

Mohsen Vakilian*
Amarin Phaosawasdi*
Michael D. Ernst†
Ralph E. Johnson*

*University of Illinois at Urbana-Champaign
†University of Washington
Type qualifiers allow additional static type checks

```java
static int oldSubindex(@Nullable MathVector ic, int l) {
    int i = 0;
    for (int k = 0; k < MathVector.NDIM; k++) {
        if (((int) ic.value(k) & l) != 0)
            i += Cell.NSUB >> (k + 1);
    }
    return i;
}
```
Java 8 supports custom type systems*

- Locking
- Aliasing
- Interning
- Immutability
- Tainting

*checkerframework.org
Java 8 supports custom type systems*

* Locking
* Aliasing
* Interning
* Immutability
* Tainting

+ additional static checks

*checkerframework.org
Java 8 supports custom type systems*

- Locking
- Aliasing
- Interning
- Immutability
- Tainting

+ additional static checks
- additional code annotation

*checkerframework.org
The manual annotation process is tedious

Run type checker
The manual annotation process is tedious

Run type checker

Add or remove qualifiers
The manual annotation process is tedious

- Run type checker
- Add or remove qualifiers
- Refactor code
Type qualifier inference tools run in batch mode.
Type qualifier inference tools run in batch mode

- **Nullness**: Julia, Nit, JastAddJ Nullness, SALSA Nullness, Xylem, Daikon Nullness
- **Immutability**: Javarifier, Pidasai, ReImInfer
- **Ownership**: Universe and Ownership Type Inference System
Type qualifier inference tools run in batch mode

Strengths

- Optimal under certain conditions
- Large change without user involvement
Type qualifier inference tools run in batch mode

Weaknesses

- Limited to one set of qualifiers
- Unpredictable
- Rigid
- Inaccurate
Type qualifier inference tools run in batch mode

- Infer
- Run type checker
- Add or remove qualifiers
- Refactor code

Program → Infer → Annotated program → Run type checker → Add or remove qualifiers → Refactor code
Type qualifier inference tools run in batch mode

Program → Infer → Run type checker → Annotated program → Refactor code → Add or remove qualifiers → Automate!
Type qualifier inference tools run in batch mode

Program → Infer → Run type checker → Annotated program

Human insight needed

Refactor code

Add or remove qualifiers

Human insight needed
Batch-mode tools make arbitrary decisions

```java
class Tree {
    // ...
    @Nullable Node root;

    void makeTree(int nstep) {
        for (Enumeration e = bodiesRev(); e.hasMoreElements();)
            Body q = (Body) e.nextElement();
        if (q.mass != 0.0) {
            q.expandBox(this, nstep);
            MathVector xqic = intcoord(q.pos);
            if (root == null) {
                root = q;
            } else {
                root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
            }
        }
    }
    root.hackcofm();
}

@Nullable MathVector intcoord(MathVector vp) {
    MathVector xp = new MathVector();
    // ...
    xsc = (vp.value(2) - xmin.value(2)) / rsize;
    if (0.0 <= xsc && xsc < 1.0) {
        xp.value(2, Math.floor(Node.IMAX * xsc));
    } else {
        return null;
    }
    return xp;
}
```

```java
class Node {
    // ...
    Node loadTree(Body p, @Nullable MathVector xpic, int l, Tree tree) {
        int si = oldSubindex(xpic, l);
        // ...
    }
    static int oldSubindex(@Nullable MathVector ic, int l) {
        int i = 0;
        for (int k = 0; k < MathVector.NDIM; k++) {
            if ((int) ic.value(k) & 1) i += Cell.NSUB >> (k + 1);
        }
        return i;
    }
}
```
q.expandBox(this, nstep);
MathVector xqic = intcoord(q.pos);
if (root == null) {
    root = q;
} else {
    root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
}
root hackcofm();

@Nullable MathVector intcoord(MathVector vp) {
    MathVector xp = new MathVector();
    //...
    xsc = (vp.value(2) - rmin.value(2)) / rsize;
    if (0.0 <= xsc && xsc < 1.0) {
        xp.value(2, Math.floor(Node.IMAX * xsc));
    } else {
        return null;
    }
}
class Node {
  //...
  Node loadTree(Body p, @Nullable MathVector xpic, int l, Tree tree) {
    int si = oldSubindex(xpic, l);
    //...
  }
}

static int oldSubindex(@Nullable MathVector ic, int l) {
  int i = 0;
  for (int k = 0; k < MathVector.NDIM; k++) {
    if (((int) ic.value(k) & l) != 0) i += Cell.NSUB >> (k + 1);
  }
  return i;
}
Batch-mode tools make arbitrary decisions

```java
class Tree {
    //...
    @Nullable Node root;

    void makeTree(int nstep) {
        for (Enumeration e = bodiesRev(); e.hasMoreElements(); ) {
            Body q = (Body) e.nextElement();
            if (q.mass != 0.0) {
                q.expandBox(this, nstep);
                MathVector xqic = intcoord(q.pos);
                if (root == null) {
                    root = q;
                } else {
                    root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
                }
            }
        }
        root.haxcofm();
    }

    @Nullable MathVector intcoord(MathVector vp) {
        MathVector xp = new MathVector();
        //...
        xsc = (vp.value(2) - min.value(2)) / rsize;
        if (0.0 <= xsc && xsc < 1.0) {
            xp.value(2, Math.floor(Node.IMAX * xsc));
        } else {
            return null;
        }
        return xp;
    }
}
```
Type qualifier inference is a refactoring

- Adding type qualifiers preserves the program behavior
- Adding maintainable type qualifiers that match the programmer's intention requires code refactoring
Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool
Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool

- Compositional Refactoring
- Speculative Analysis

→ Cascade
## Inferring Primitive Changes from Error Messages

<table>
<thead>
<tr>
<th>Type Checker Error</th>
<th>Fix</th>
</tr>
</thead>
</table>
| incompatible types in argument.  
root = root.loadTree(q, xqic);  
^  
found : @Nullable MathVector  
required: @NonNull MathVector |
Inferring Primitive Changes from Error Messages

Type Checker Error

```
incompatible types in argument.
    root = root.loadTree(q, xqic);
    ^
found : @Nullable MathVector
required: @NonNull MathVector
```

Fix

Change parameter `xpic` of `loadTree()` to @Nullable MathVector
Inferring Primitive Changes from Error Messages

Type Checker Error

incompatible types in argument.
root = root.loadTree(q, xqic);

^  
found : @Nullable MathVector
required: @NonNull MathVector

Fix

Change parameter xpic of loadTree() to @Nullable MathVector

Nullness

Mutability

call to value(int) not allowed on the given receiver.
ic.value(k);

^  
found : @ReadOnly MathVector
required: @Mutable MathVector
Inferring Primitive Changes from Error Messages

**Type Checker Error**

**Nullness**

Incompatible types in argument.

```java
root = root.loadTree(q, xqic);
```

- **found**: `@Nullable MathVector`
- **required**: `@NonNull MathVector`

**Fix**

Change parameter `xqic` of `loadTree()` to `@Nullable MathVector`

**Mutability**

Call to `value(int)` not allowed on the given receiver.

```java
ic.value(k);
```

- **found**: `@ReadOnly MathVector`
- **required**: `@Mutable MathVector`

**Fix**

Change receiver parameter of `value()` to `@ReadOnly MathVector`
Speculative analysis suggests a change composition

```
incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode
```
Speculative analysis suggests a change composition

```
incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode
```

```
Change parameter l to @Nullable TreeNode (1)
```
Speculative analysis suggests a change composition
Speculative analysis suggests a change composition

```java
incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode
```

```java
incompatible types in assignment.
left = l;
found : @Initialized @NonNull TreeNode
required: @Initialized @NonNull TreeNode
```

```java
Change parameter l to @Nullable TreeNode (1)
```

```java
Change field left to @Nullable TreeNode (1)
```
Cascade Tree

```java
incompatible types in assignment.
left = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in assignment.
private TreeNode left = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in assignment.
private TreeNode right = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in assignment.
right = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode

incompatible types in return.
return null;
found : null
required: @Initialized @NonNull TreeNode
```
Cascade Tree
Cascade Tree

incompatible types in assignment.
left = null;
found : null
required: @Initialized @NonNull TreeNode

- Change field left to @Nullable TreeNode (2)
incompatible types in assignment.
private TreeNode left = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in assignment.
private TreeNode right = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in assignment.
right = null;
found : null
required: @Initialized @NonNull TreeNode

incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode

incompatible types in return.
return null;
found : null
required: @Initialized @NonNull TreeNode
public TreeNode() {
    this(1, null, null);
}

public TreeNode(int levels) {
    value = 1;
    if (levels <= 0) throw new RuntimeException("Number of levels must be positive not \n    left = null;
    right = null;
} else {
    left = new TreeNode(levels - 1);
    right = new TreeNode(levels - 1);
}

public void setChildren(TreeNode l, TreeNode r) {
    left = l;
    right = r;
}
package treecdd;
/**
 * A TreeNode data structure.
 */
public class TreeNode {
    private int value = 0;

    private TreeNode left = null;
    private TreeNode right = null;

    /**
     * Create a node in the tree with a given value and two children.
     *
     * @param v the node's value
     * @param l the left child.
     * @param r the right child.
     */
    public TreeNode(int v, TreeNode l, TreeNode r) {
        value = v;
        left = l;
        right = r;
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode(TreeNode l, TreeNode r) {
        this(l, l, r);
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode() {
        this(1, null, null);
    }
}
```java
public class TreeNode {
    private @Nullable TreeNode left = null;
    private TreeNode right = null;

    /* Create a node in the tree with a given value and two children. */
    /* @param v the node's value */
    /* @param l the left child. */
    /* @param r the right child. */
    /**
     * Create a node in the tree with a given value and two children.
     */
    public TreeNode(int v, TreeNode l, TreeNode r) {
        value = v;
        left = l;
        right = r;
    }

    /* Create a tree node given the two children. The initial node value is 1. */
    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode(TreeNode l, TreeNode r) {
        this(l, l, r);
    }

    /* Create a tree node given the two children. The initial node value is 1. */
    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode() {
        this(1, null, null);
    }
}
```
The speculative analysis computes a tree recursively.
The speculative analysis computes a tree recursively
The speculative analysis computes a tree recursively
The speculative analysis computes a tree recursively

Program

$\text{p}$

Fix

$c_1$

Error

$e_1$

$c_1(p)$

$e_2$

$e_3$

$c_2$

$c_3$
The speculative analysis computes a tree recursively.
A change is represented as an AST path

<table>
<thead>
<tr>
<th>Primitive Change</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Decl. Fixer</td>
<td>Compilation Unit + Variable Decl. + New Type</td>
</tr>
<tr>
<td>Method Return Fixer</td>
<td>Compilation Unit + Method Decl. + New Type</td>
</tr>
<tr>
<td>Method Receiver Fixer</td>
<td>Compilation Unit + Method Decl. + New Type</td>
</tr>
</tbody>
</table>
Research Questions

How does **Cascade** compare with **Julia**, a batch qualifier inference tool?

- Learnability
- Quality of results
- Task completion time
- Control over process
- Willingness to use
User study

Subjects:
- 12 computer science graduate students from 9 different research labs
- Familiar with Java and Eclipse
- Average of 10 years of programming experience
Training

- Nullness Checker
- Julia
- Cascade

github.com/reprogrammer/tqi-study
## Task Design

<table>
<thead>
<tr>
<th></th>
<th>Julia then Cascade</th>
<th>Cascade then Julia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MST then BH</strong></td>
<td>3 participants</td>
<td>3 participants</td>
</tr>
<tr>
<td><strong>BH then MST</strong></td>
<td>3 participants</td>
<td>3 participants</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BH</strong></th>
<th>Barnes-Hut, a hierarchical force-calculation algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MST</strong></td>
<td>Bentley’s algorithm for finding the minimum spanning tree of a graph</td>
</tr>
</tbody>
</table>
Users complete tasks faster with Cascade

- **t test**
  - $p = 0.01$
  - Cohen’s $d = 1.13$

![Bar chart showing comparison between Cascade and Julia in terms of time (in minutes). The chart indicates that users complete tasks faster with Cascade.]
Users added less inaccurate annotations with Cascade

<table>
<thead>
<tr>
<th></th>
<th>BH + MST</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Julia</td>
<td>Cascade</td>
</tr>
<tr>
<td>Correct</td>
<td>13.9</td>
<td>9.6</td>
</tr>
<tr>
<td>Incorrect</td>
<td>2.8</td>
<td>0.1</td>
</tr>
<tr>
<td>Redundant</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td>Unnecessary warning</td>
<td>7.2</td>
<td>0</td>
</tr>
<tr>
<td>suppressions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Postquestionnaire Results

<table>
<thead>
<tr>
<th>Questions</th>
<th>Cascade better</th>
<th>Equal Rating</th>
<th>Julia better</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found T easy to <strong>learn</strong>.</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>I know <strong>why</strong> T inserted each annotation.</td>
<td>8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Using T, I have <strong>control</strong> over the process of annotating the code.</td>
<td>9</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>I'm willing to <strong>use</strong> T in the future.</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Qualitative Interview Results

The participants believe that:

- Cascade's *speculative analysis* is useful (N = 8).
- Cascade is more *predictable* (N = 7).
- Cascade's tree computation is *slow* (N = 5).
- The *overhead* of fixing Julia's annotations is high (N = 7).
Future Work

- Improve the performance of Cascade.
- Evaluate compositional refactoring and Cascade in the field.
- Make Cascade support bidirectional speculative analysis.
Cascade is easy to use and helps users complete tasks fast.

Compositional refactoring and speculative analysis.

Less is sometimes more in the automation of software evolution tasks.
  - More automation is not always better
  - Some tasks need problem-solving and creativity
  - Applicable to other fields