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

```
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) != 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

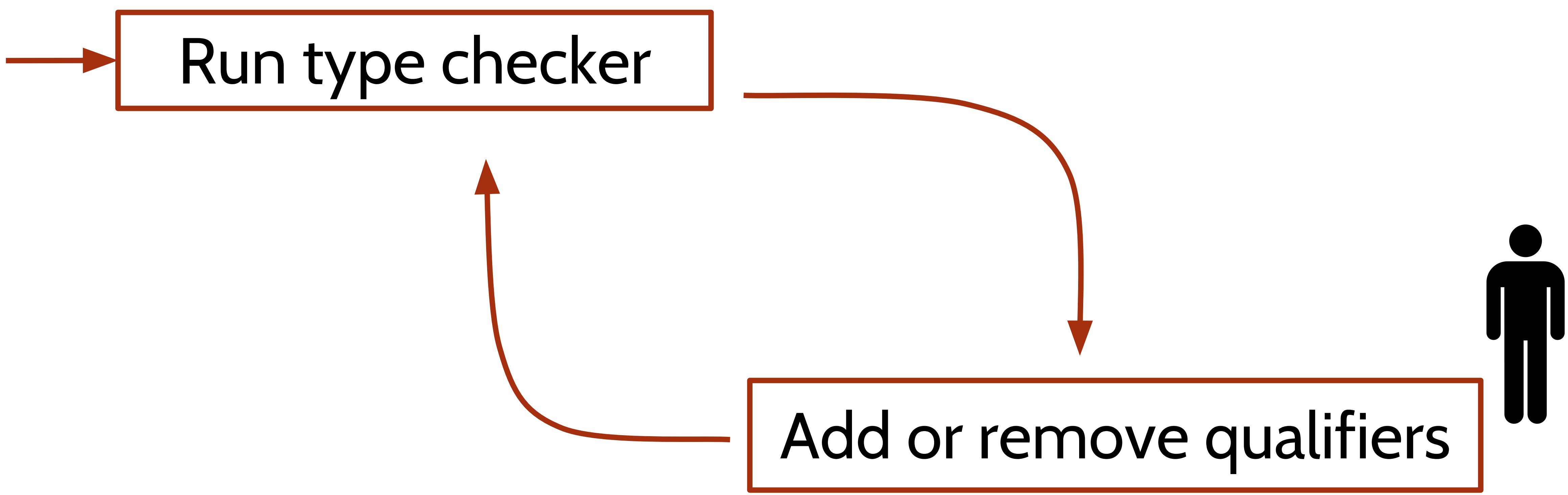
- additional code annotation

*checkerframework.org

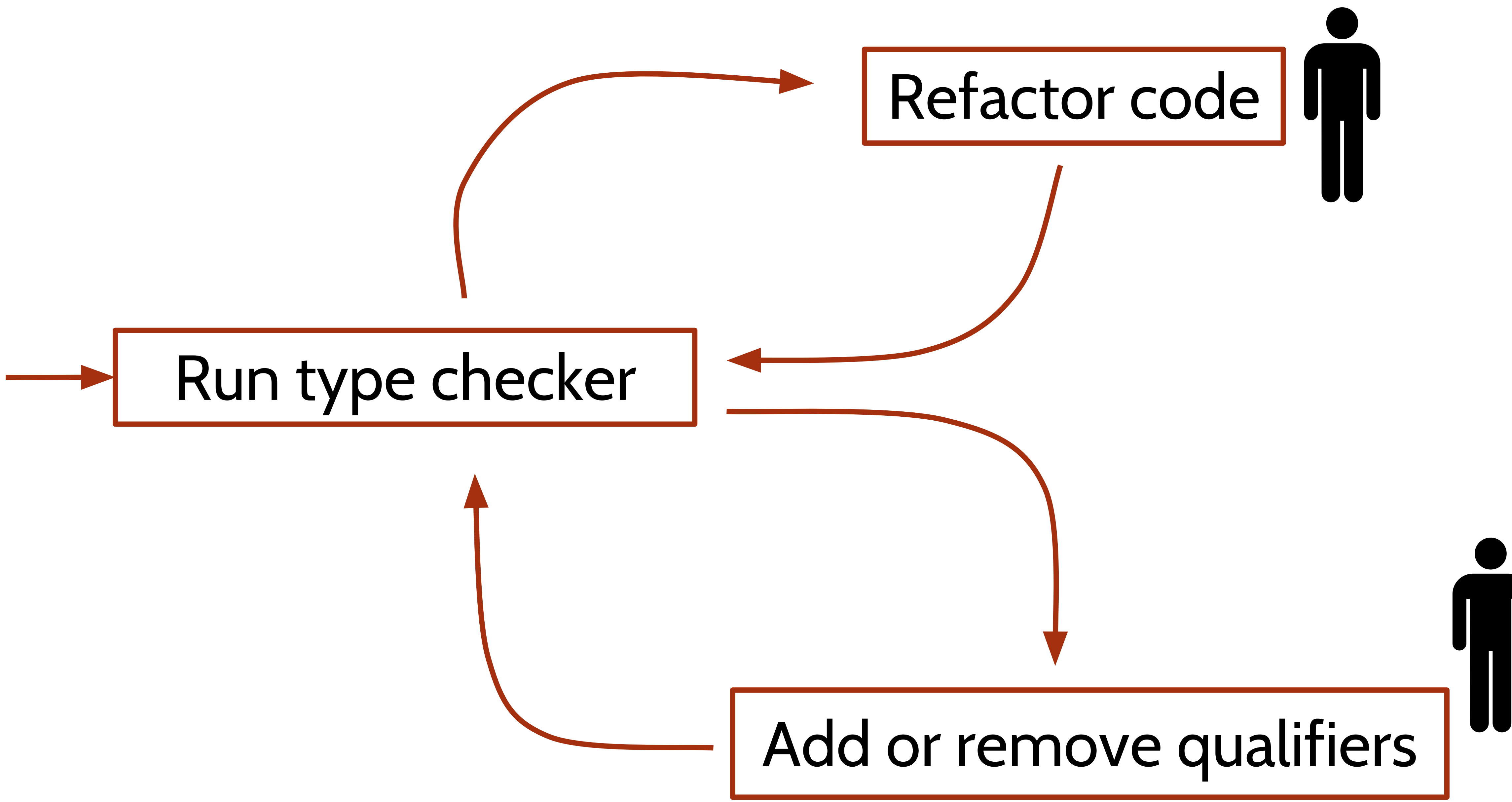
The manual annotation process is tedious

Run type checker

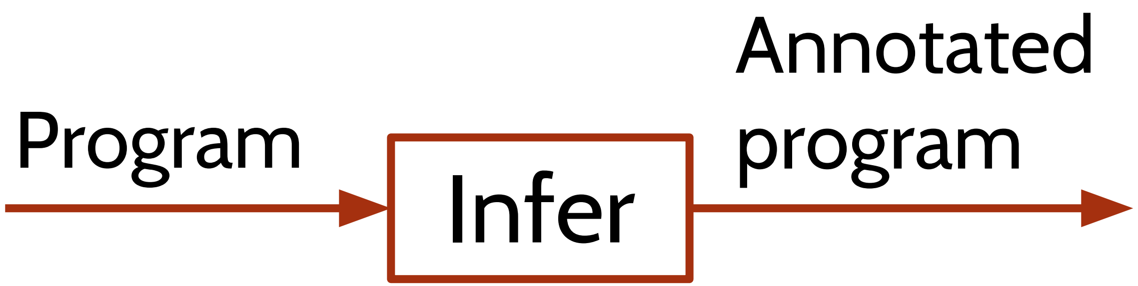
The manual annotation process is tedious



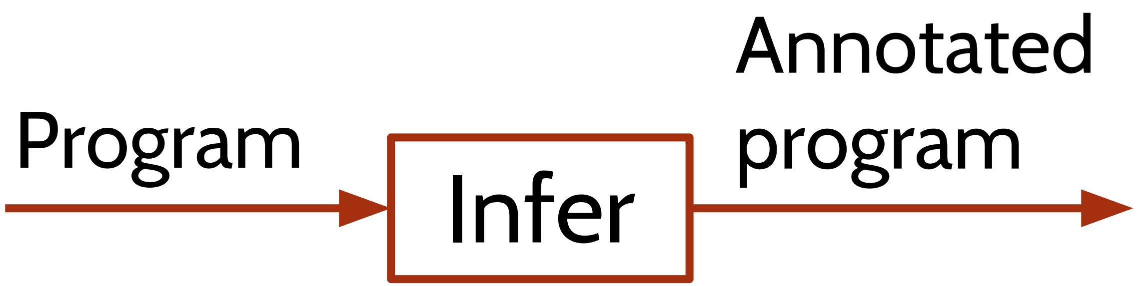
The manual annotation process is tedious



Type qualifier inference tools run in batch mode



Type qualifier inference tools run in batch mode

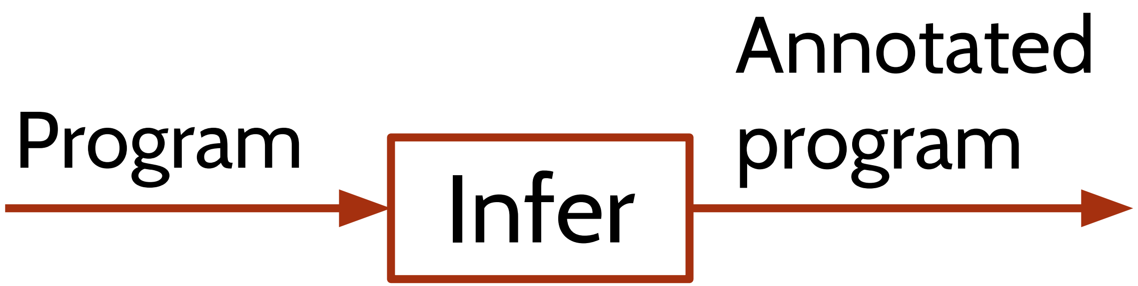


Nullness
Julia, Nit, JastAddJ Nullness, SALSA Nullness, Xylem, Daikon Nullness

Immutability
Javarifier, Pidasai, ReImInfer

Ownership
Universe and Ownership Type Inference System

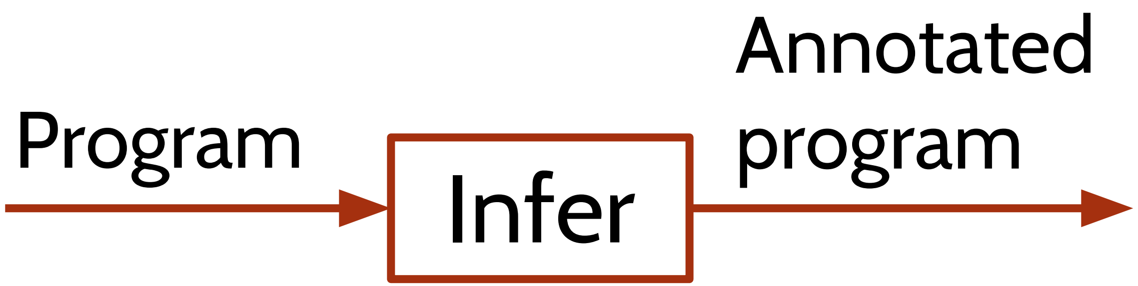
Type qualifier inference tools run in batch mode



Strengths

- Optimal under certain conditions
- Large change without user involvement

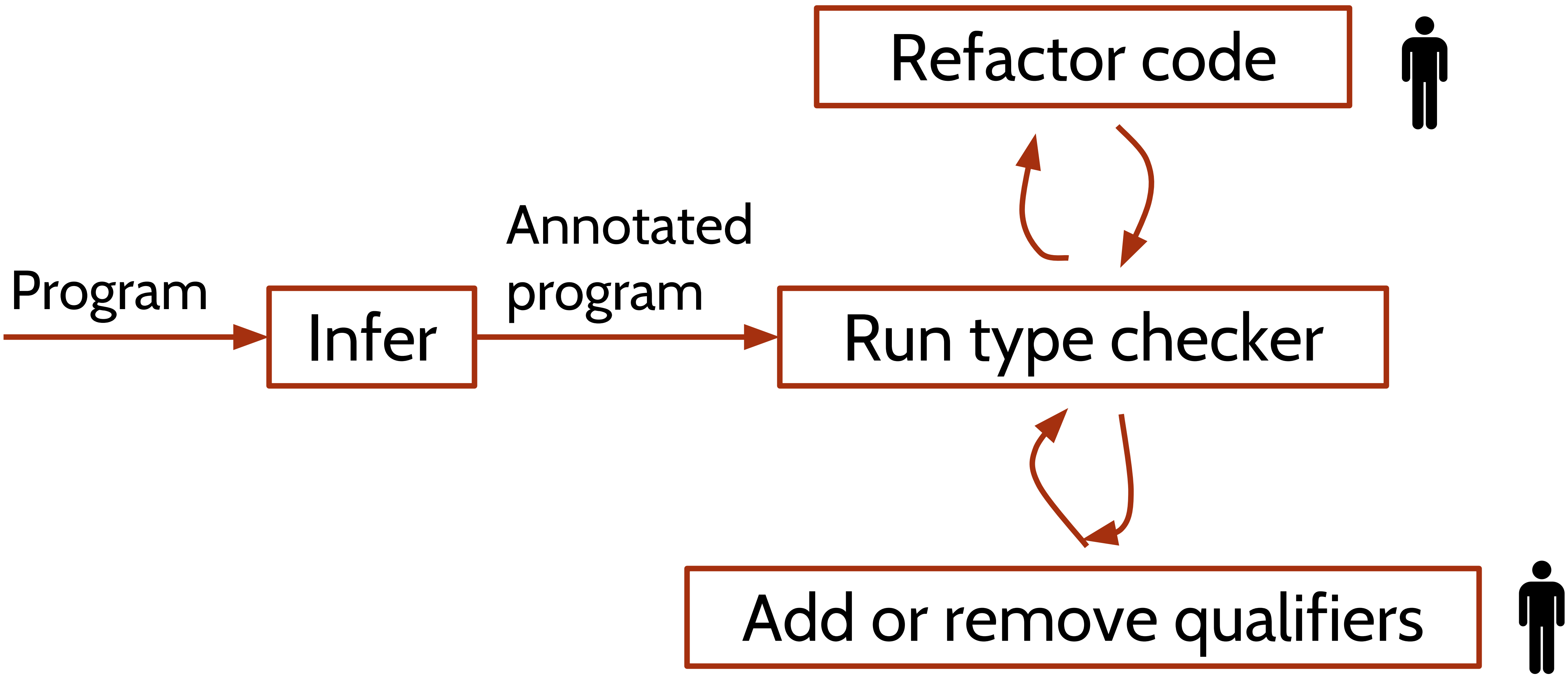
Type qualifier inference tools run in batch mode



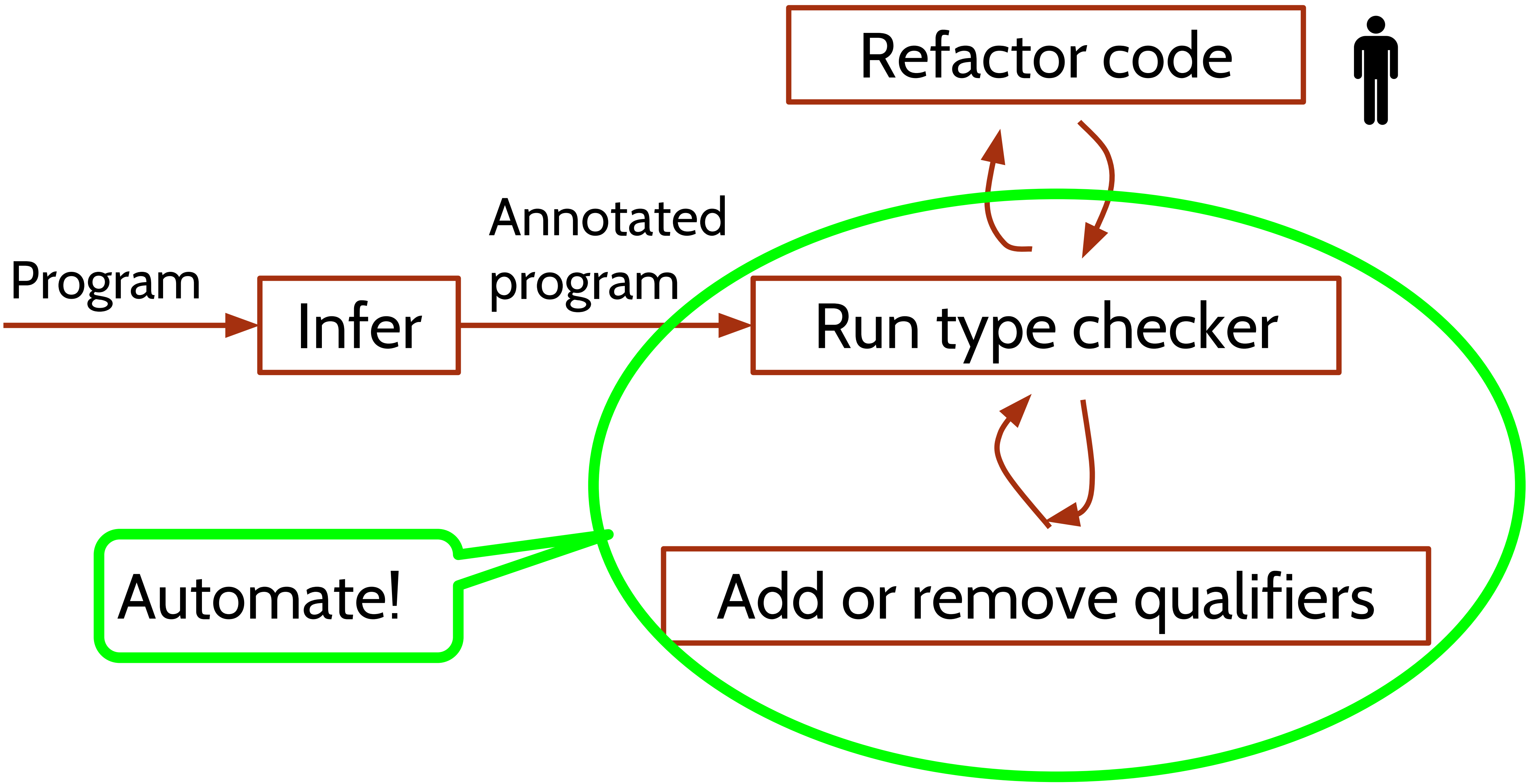
Weaknesses

- Limited to one set of qualifiers
- Unpredictable
- Rigid
- Inaccurate

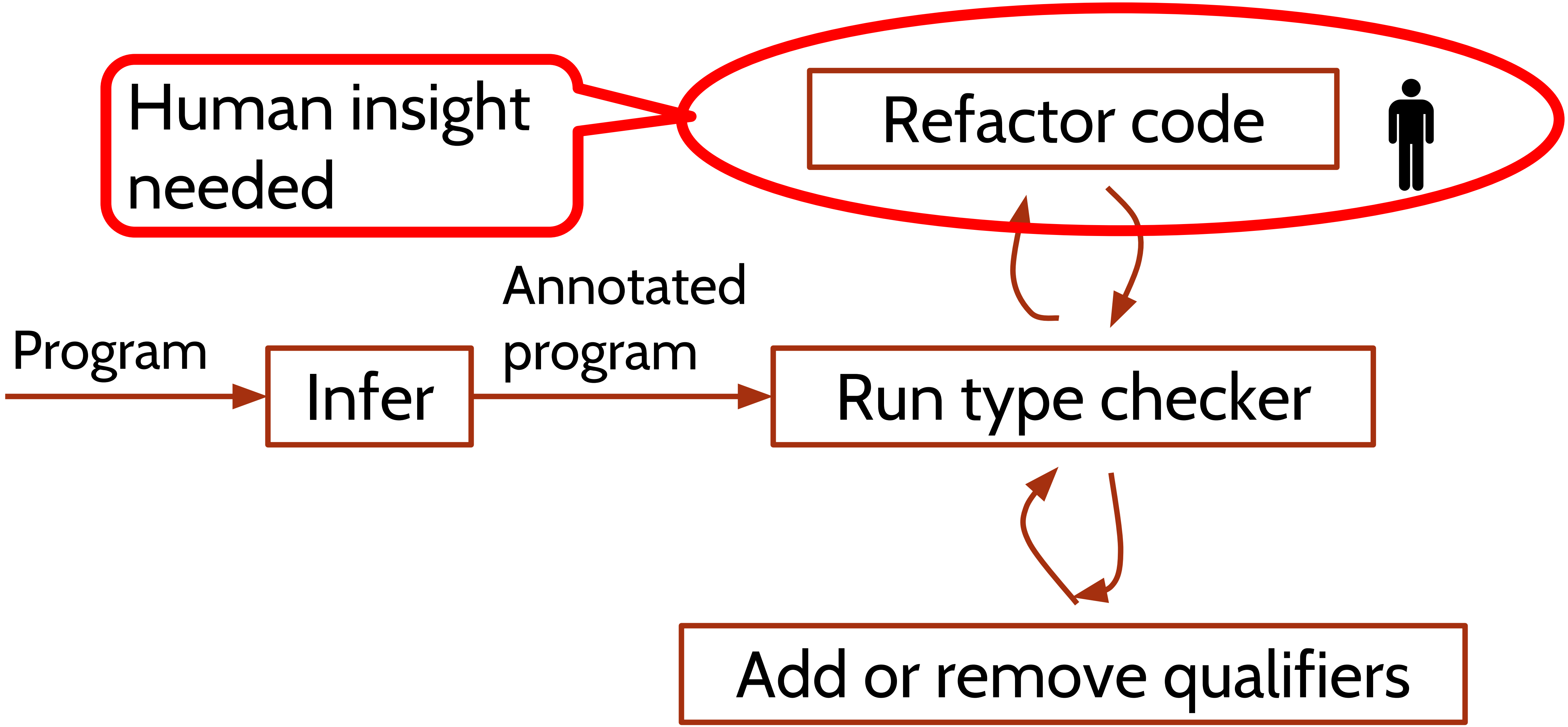
Type qualifier inference tools run in batch mode



Type qualifier inference tools run in batch mode



Type qualifier inference tools run in batch mode



Batch-mode tools make arbitrary decisions

```
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) - rmin.value(2)) / rsize;
    if (0.0 <= xsc && xsc < 1.0) {
      xp.value(2, Math.floor(Node.IMAX * xsc));
    } else {
      return null;
    }
    return xp;
  }
}
```

```
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;
  }
}
```

```
        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;
    }
```

```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) & l) != 0) i += Cell.NSUB >> (k + 1);
    }
    return i;
  }
}
```

BUG

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) - rmin.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) & l) != 0) i += Cell.NSUB >> (k + 1);
    }
    return i;
  }
}
```

BUG

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

	Type Checker Error	Fix
Nullness	see below	

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

Inferring Primitive Changes from Error Messages

Type Checker Error

Fix

Nullness

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

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

Inferring Primitive Changes from Error Messages

Type Checker Error

Fix

Nullness

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

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

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		Fix

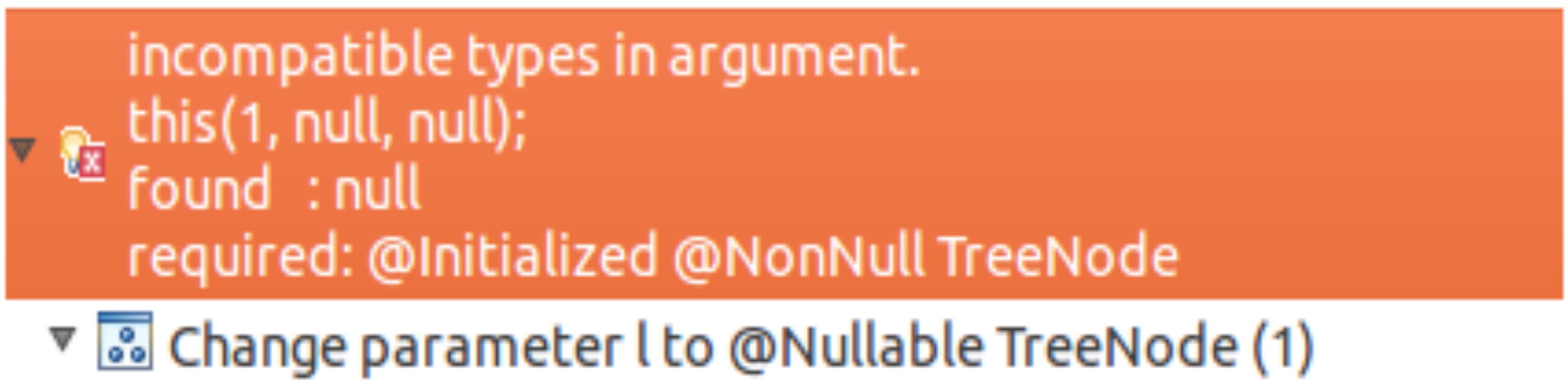
Nullness	`incompatible types in argument.`
	`  root = root.loadTree(q, xqic);`
	`                           ^`
	`found   :` **`@Nullable`** `MathVector`
	`required:` **`@NonNull`** `MathVector`
		➡	Change parameter `xpic` of `loadTree()` to `@Nullable MathVector`

Mutability	`call to value(int) not allowed`
	`on the given receiver.`
	`      ic.value(k);`
	`              ^`
	`found   :` **`@ReadOnly`** `MathVector`
	`required:` **`@Mutable`** `MathVector`
		➡	Change receiver parameter of `value()` to `@ReadOnly MathVector`

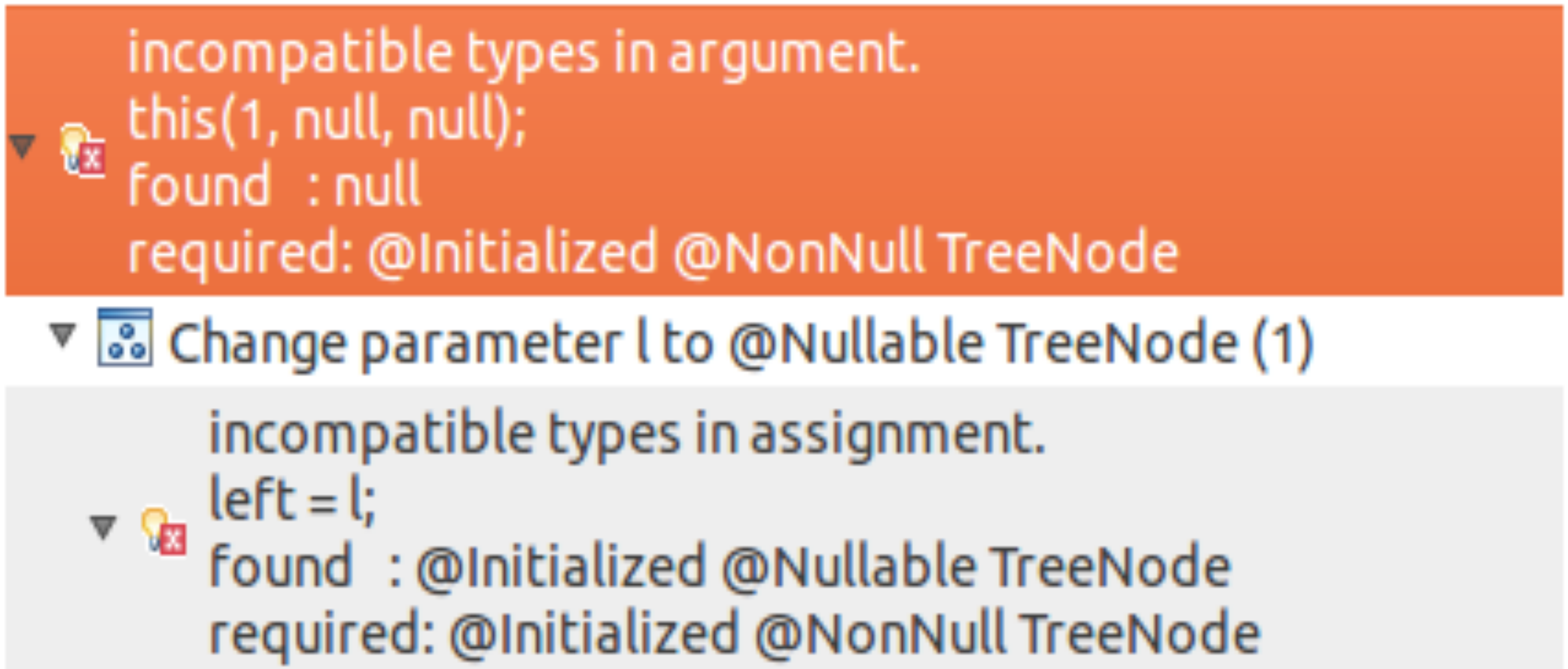
Speculative analysis suggests a change composition

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

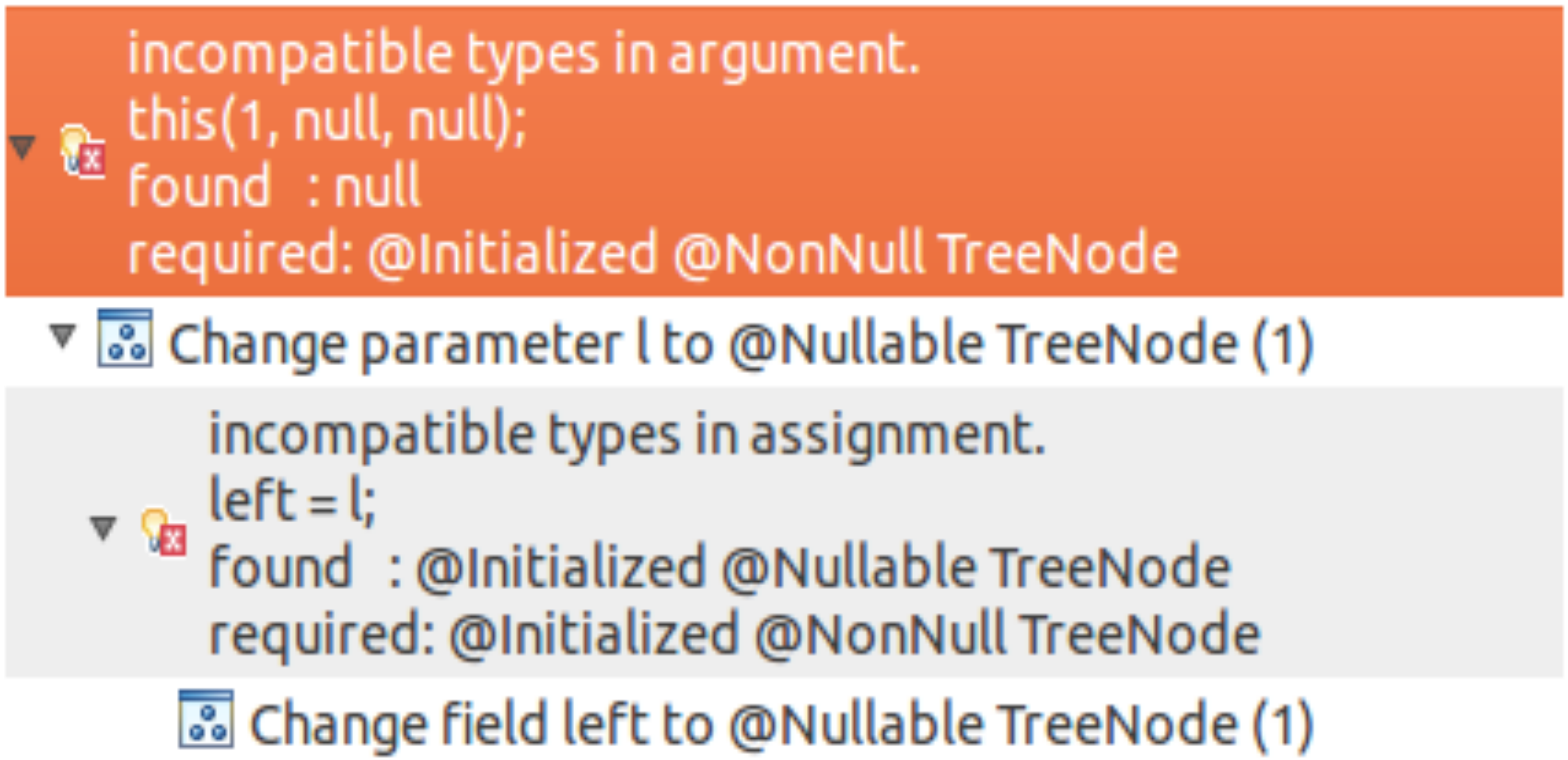
Speculative analysis suggests a change composition



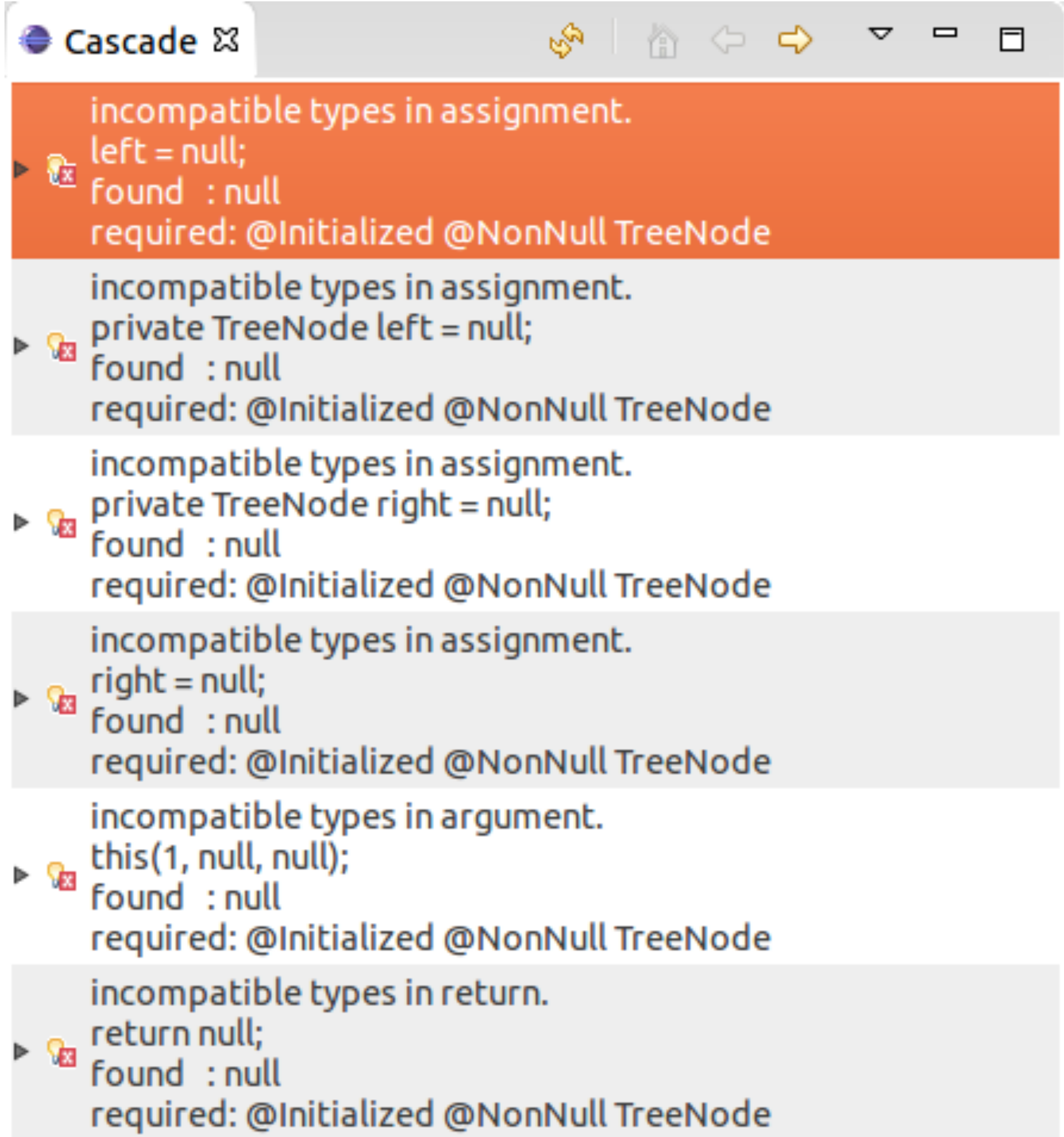
Speculative analysis suggests a change composition



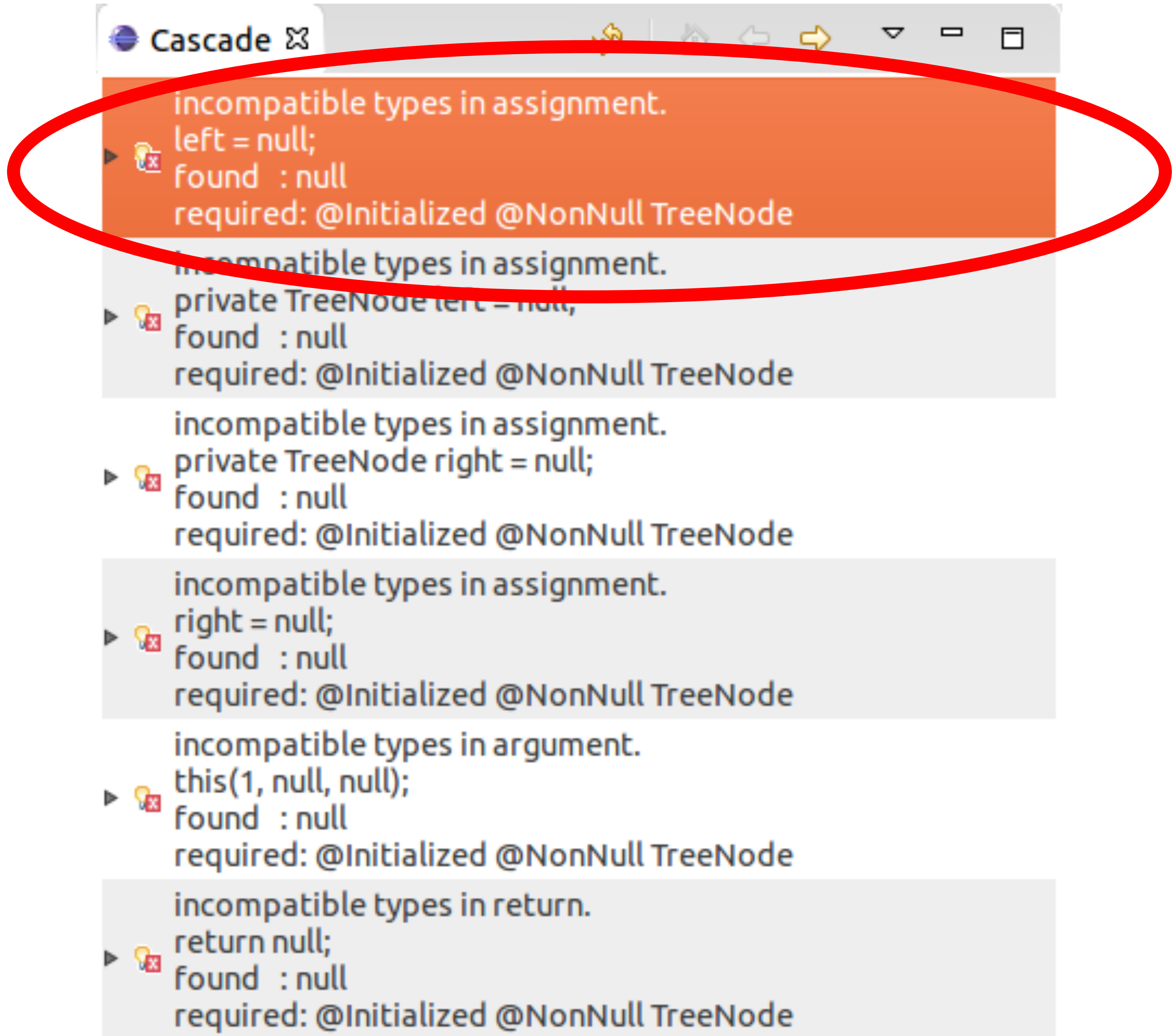
Speculative analysis suggests a change composition



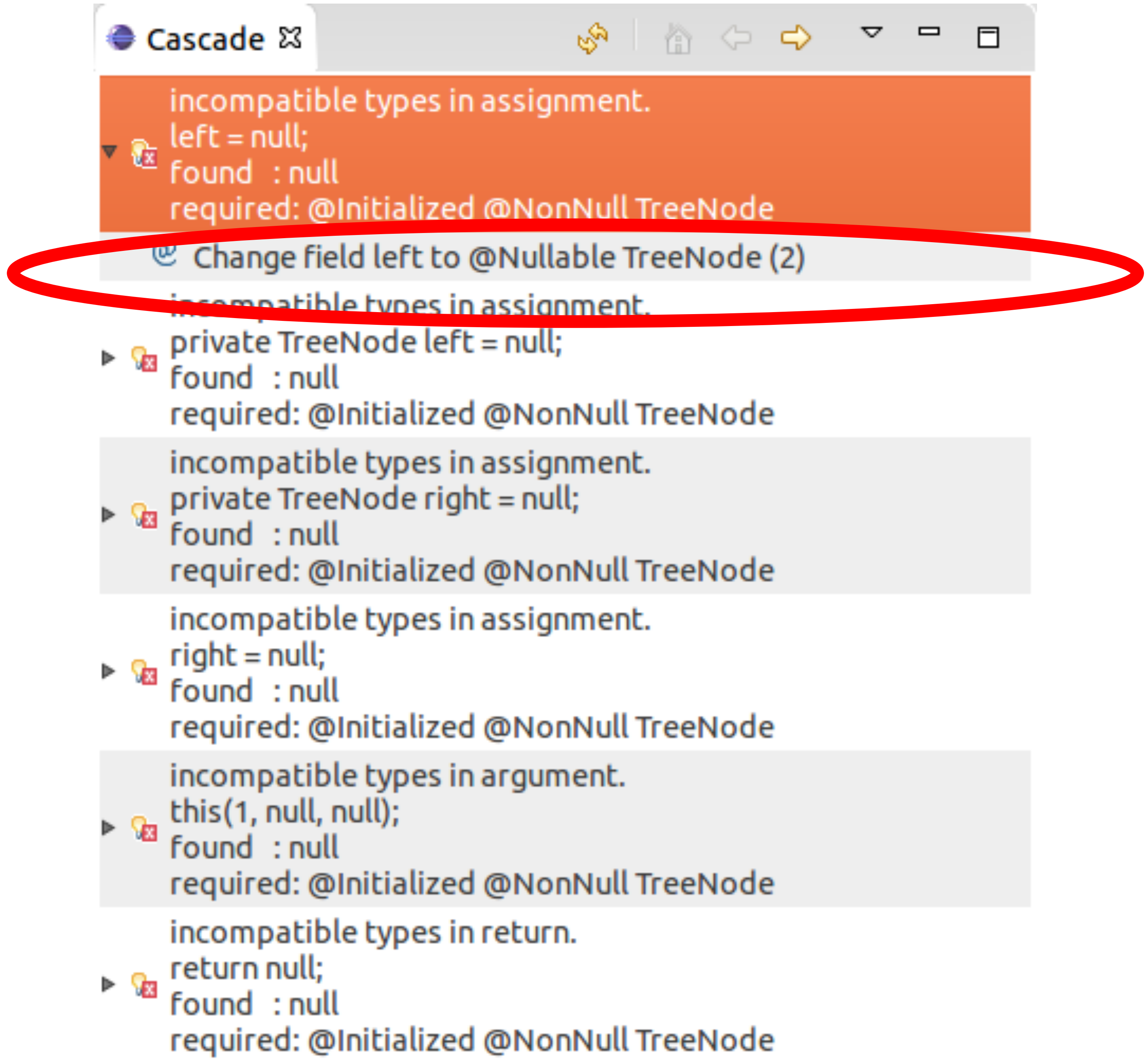
Cascade Tree



Cascade Tree



Cascade Tree



TreeNode.java

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

/**
 * Construct a subtree with the specified number of levels. We recursively call the c
 * create the tree.
 *
 * @param levels the number of levels in the subtree
 **/
public TreeNode(int levels) {
  value = 1;
  if (levels <= 1) {
    if (levels <= 0) throw new RuntimeException("Number of levels must be positive no
    left = null;
    right = null;
  } else {
    left = new TreeNode(levels - 1);
    right = new TreeNode(levels - 1);
  }
}

/**
 * Set the children of the tree
 *
 * @param l the left child
 * @param r the right child
 **/
public void setChildren(TreeNode l, TreeNode r) {
  left = l;
  right = r;
}
```

Cascade | Outline

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

TreeNode.java

```java
package treeadd;
/**
 * A Tree node 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(1, l, r);
  }

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

Cascade | Outline

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

*TreeNode.java

```java
/**
 * A Tree node data structure.
 **/
public class TreeNode {
  private int value = 0;
  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.
   **/
  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(1, l, r);
  }

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

Cascade | Outline

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

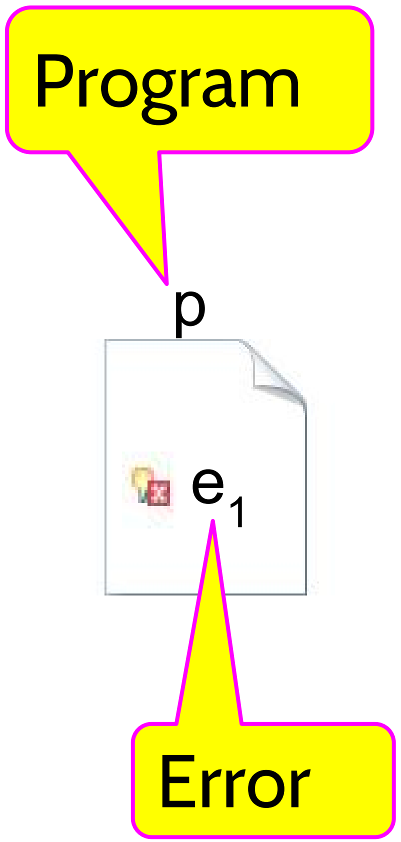
incompatible types in assignment.
private TreeNode right = null;
found : null
required: @Initialized @NonNull TreeNode

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

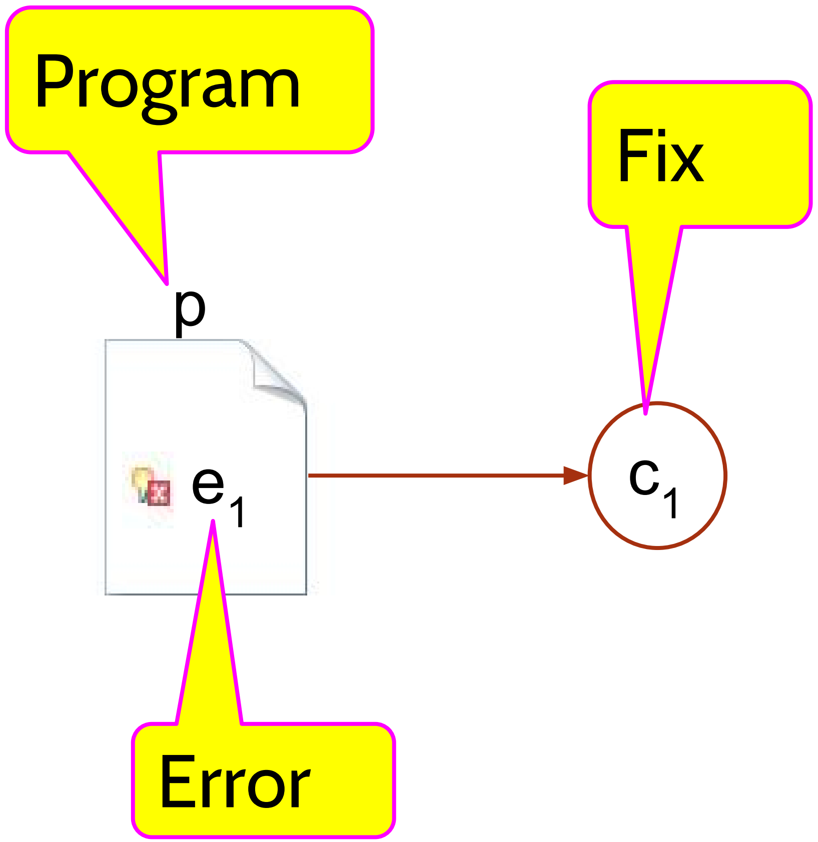
incompatible types in argument.
this(1, null, null);
found : null
required: @Initialized @NonNull TreeNode

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

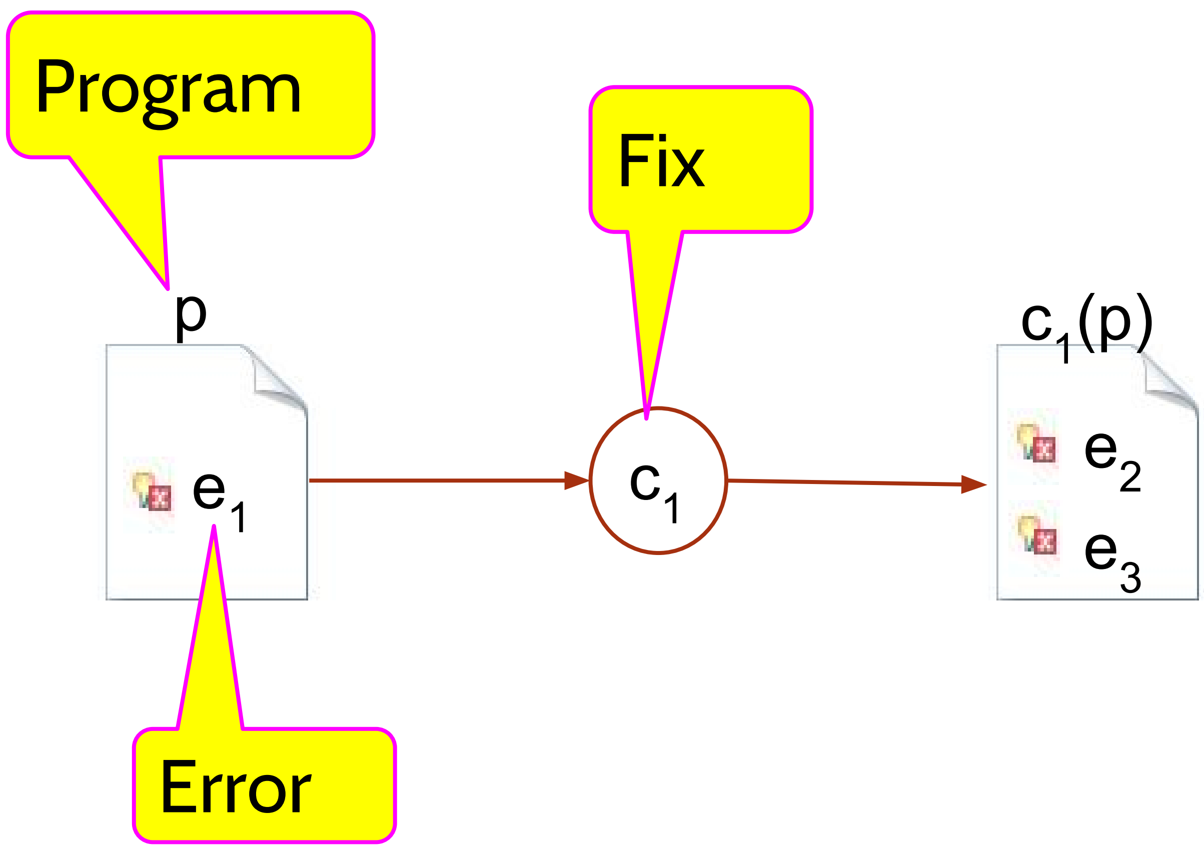
The speculative analysis computes a tree recursively



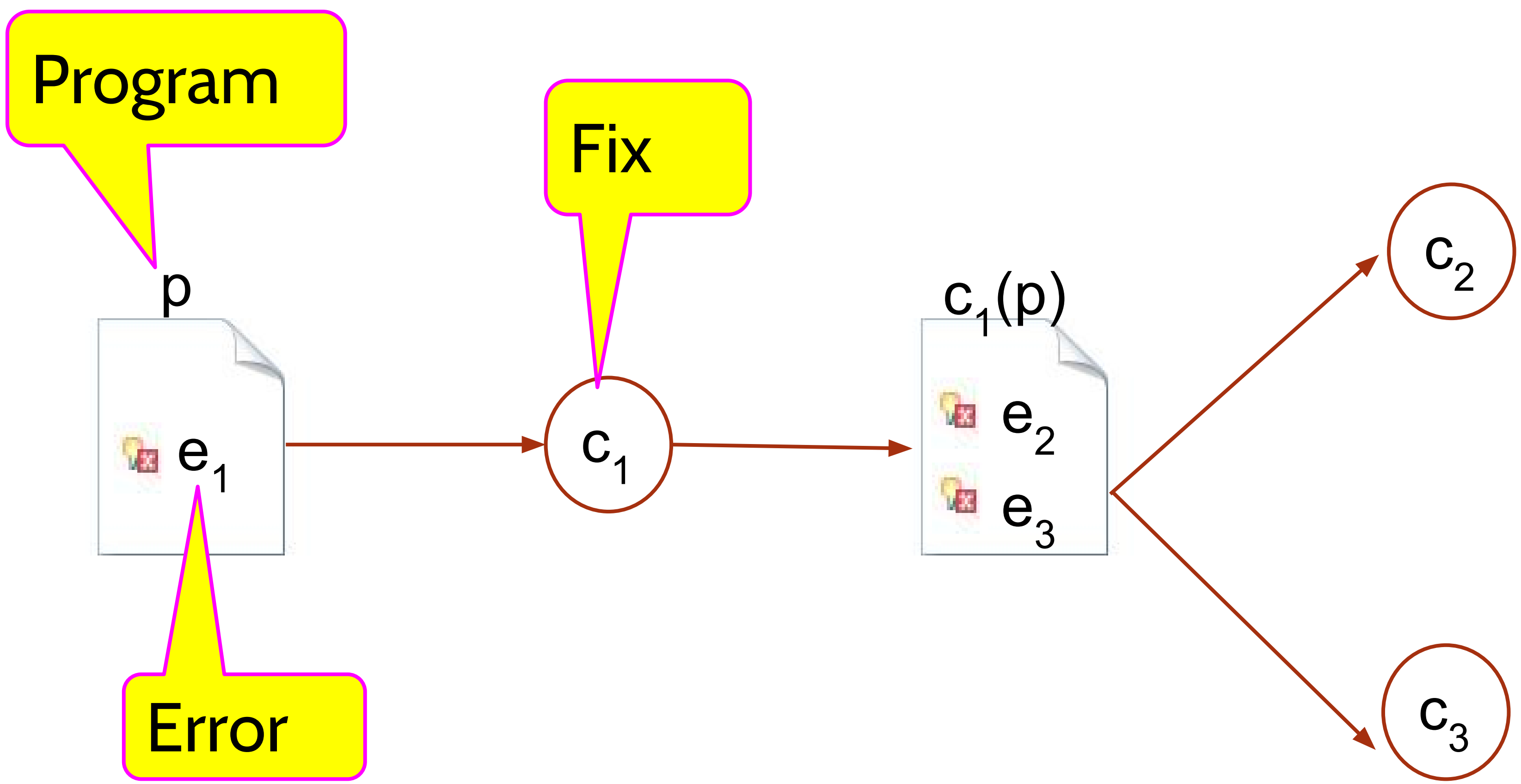
The speculative analysis computes a tree recursively



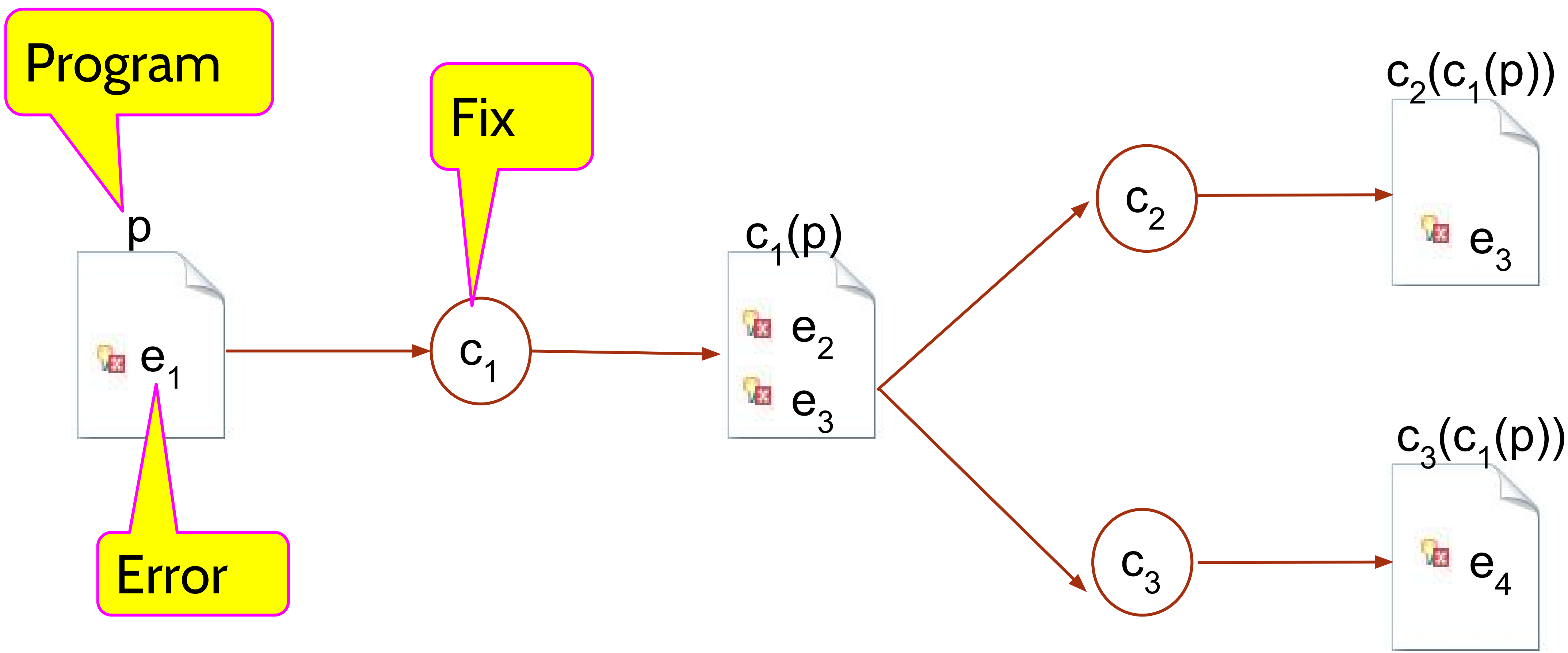
The speculative analysis computes a tree recursively



The speculative analysis computes a tree recursively



The speculative analysis computes a tree recursively



A change is represented as an AST path

Primitive Change	**Representation**
Variable Decl. Fixer	Compilation Unit + Variable Decl. + New Type
Method Return Fixer	Compilation Unit + Method Decl. + New Type
Method Receiver Fixer	Compilation Unit + Method Decl. + New Type

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

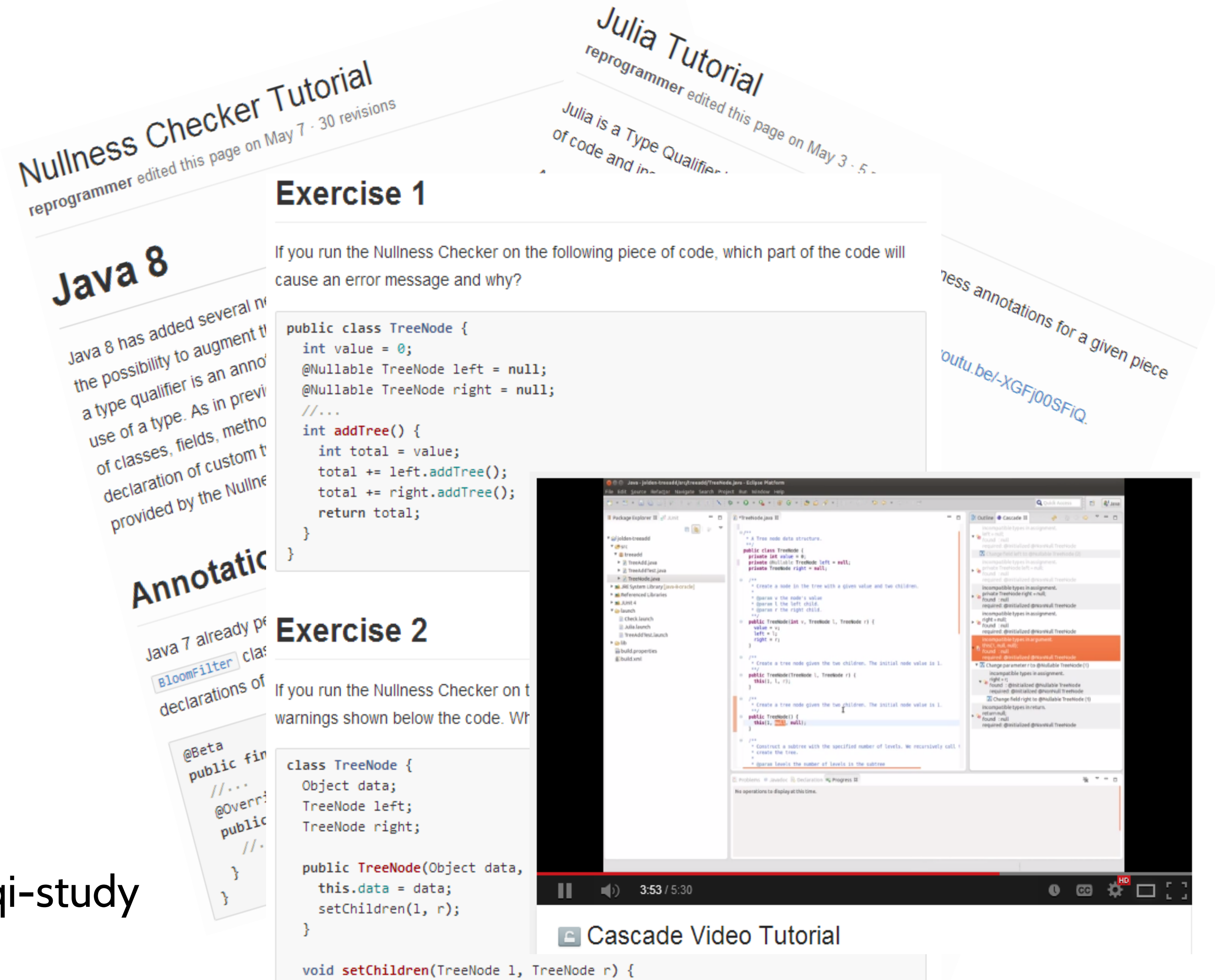
Subjects:

- **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

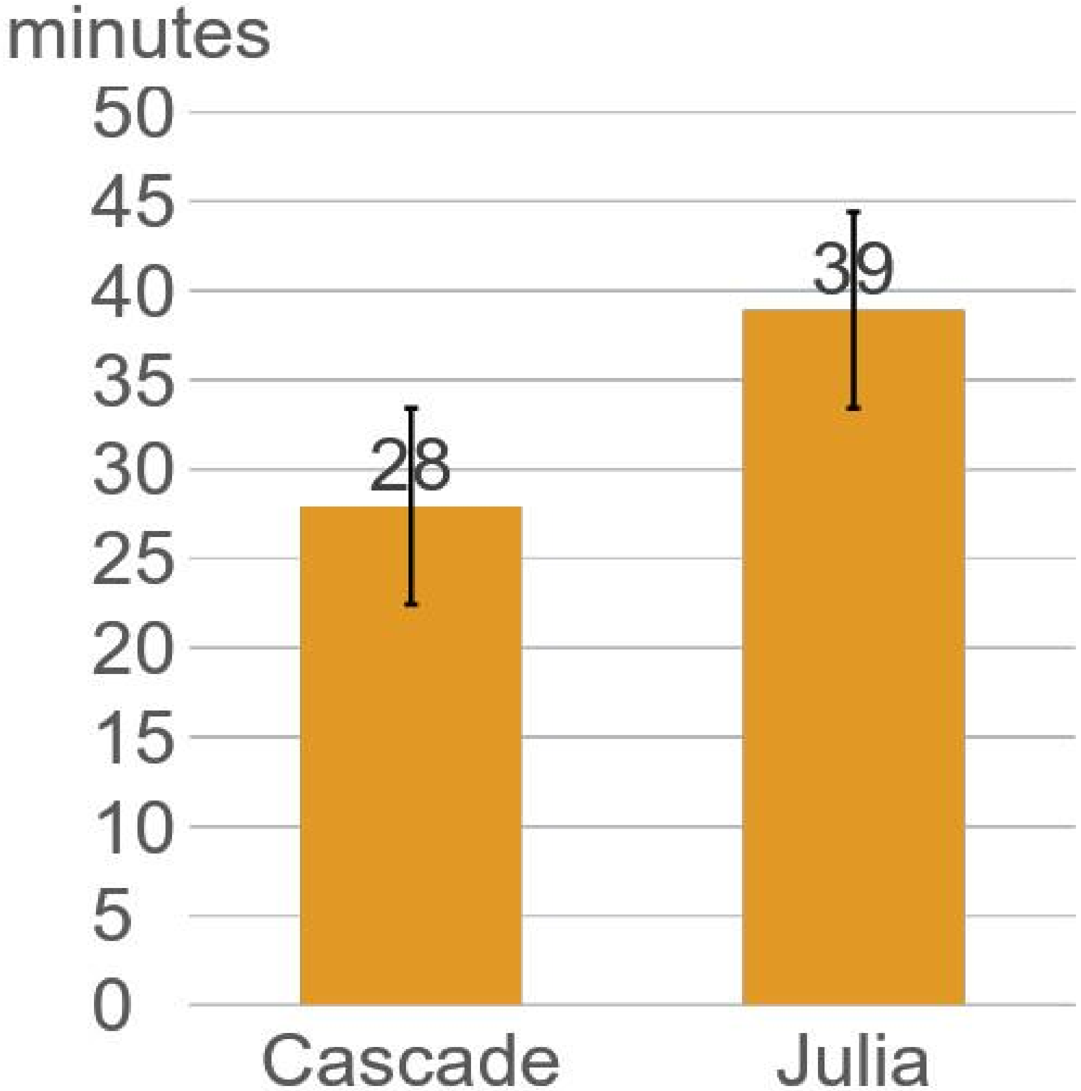


Task Design

	Julia then Cascade	Cascade then Julia
MST then BH	3 participants	3 participants
BH then MST	3 participants	3 participants

BH	Barnes-Hut, a hierarchical force-calculation algorithm
MST	Bentley's algorithm for finding the minimum spanning tree of a graph

Users complete tasks faster with Cascade



t test

- $p = 0.01$
- Cohen's $d = 1.13$

Users added less inaccurate annotations with Cascade

	BH + MST	
	Julia	**Cascade**
Correct	13.9	9.6
Incorrect	2.8	0.1
Redundant	1.7	0.4
Unnecessary warning suppressions	7.2	0

Postquestionnaire Results

Questions *(T = Julia or Cascade)*	**Cascade better**	**Equal Rating**	**Julia better**
I found T easy to **learn**.	3	6	3
I know **why** T inserted each annotation.	8	4	0
Using T, I have **control** over the process of annotating the code.	9	0	3
I'm willing to **use** T in the future	11	0	1

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: A Universal Type Qualifier Inference Tool

- 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