

Static Automated Program Repair for Heap Properties

Rijnard van Tonder and Claire Le Goues

Carnegie
Mellon
University



Motivating Example

A bug in error-prone

Motivating Example

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

“method” can be null

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
  
        + checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
  
        + checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed +
Test
Added

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed +
Test
Added

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed +
Test
Added

Same
issue!

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed +
Test
Added

Fixed 18
months later!

What We Do

What We Do

- Local reasoning

What We Do

- Local reasoning of program fragments

What We Do

- Local reasoning of program fragments

for Program Repair!

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed +
Test
Added

Fixed 18
months later!

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
  
        + checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        if (method == null)  
            throw new Error("Method may be null");  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Local semantic effects

Fixed 18 months later!

method may be null

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, id);  
  
        + checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        - method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Local semantic effects

Pre & Postconditions

Fixed 18 months later!

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, id);  
  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Local semantic effects

Pre & Postconditions



Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, id);  
  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Local semantic effects

Pre & Postconditions

Motivating Example

```
public class GuardedByBinder {  
    case IDENTIFIER: {
```

```
        Symbol.MethodSymbol method
```

```
        context.resolver.resolveMethod(node, id)
```

Local semantic effects

Pre & Postconditions

```
    checkGuardedBy(method != null, identifier.toString());  
    return bindSelect(computeBase(context, method), method);  
} case MEMBER_SELECT: {
```

Right now.

```
    Symbol method =
```

```
    context.resolver.resolveMethod(node, identifier.getName());
```

```
    checkGuardedBy(method != null, identifier.toString());
```

```
    return bindSelect(computeBase(context, method), method);
```

```
}
```

```
}
```

What We Do

- Local reasoning of program fragments

for Program Repair!

What We Do

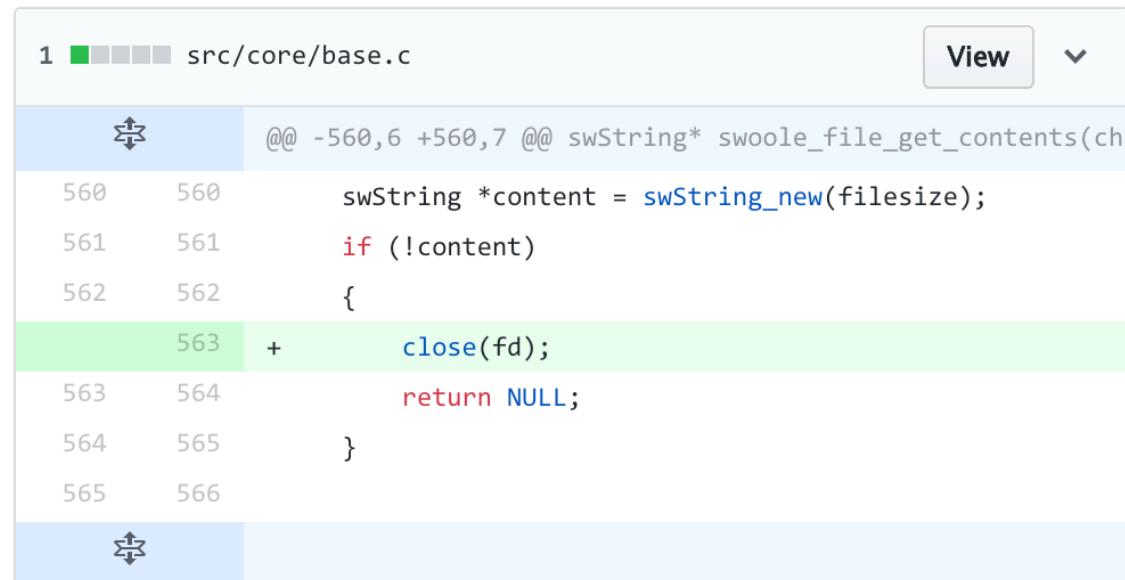
- Local reasoning of program fragments^{existing}
 ^

for Program Repair!

What We Do

- Local reasoning of program fragments

for Program Repair!



```
1  src/core/base.c
View ▾
diff --git a/src/core/base.c b/src/core/base.c
@@ -560,6 +560,7 @@ swString* swoole_file_get_contents(char *path, int fd, int offset, int size)
560      560          swString *content = swString_new(filesize);
561      561          if (!content)
562      562          {
+         563      +              close(fd);
563      564          return NULL;
564      565      }
565      566
```

What We Do

- Local reasoning of program fragments

for Program Repair!

No tests

```
1  src/core/base.c View ▾
@@ -560,6 +560,7 @@ swString* swoole_file_get_contents(char *path, int fd, int filesize)
    swString *content = swString_new(filesize);
    if (!content)
        {
            close(fd);
            return NULL;
        }
    }
```

⊕

What We Do

- Local reasoning of program fragments

for Program Repair!

The screenshot shows a code editor interface with a green header bar containing the text "No tests". The main area displays a snippet of C code from "src/core/base.c". The code is annotated with a repair patch:

```
@@ -560,6 +560,7 @@ swString* swoole_file_get_contents(char *path, int offset, int length)
    swString *content = swString_new(filesize);
    if (!content)
        {
            close(fd);
            return NULL;
        }
    }
```

A green highlight covers the line "close(fd);". A green button labeled "Real programs" is positioned at the bottom right of the code area.

What We Do

- Local reasoning of program fragments

for Program Repair!

Merged

e/base.c

@ -560,6 +560,7 @@ swString* swoole_file

```
560 560     swString *content = swString_new(filesize);
561 561     if (!content)
562 562     {
563 563 +     close(fd);
564 564     return NULL;
565 565 }
```

Watch ▾ 827 Star 8,803 Fork 2,124

The image shows a GitHub pull request interface. A purple box labeled "Merged" is overlaid on the top left. On the right, the SWOLE logo is displayed. The main area shows a diff of a file named "e/base.c". The diff shows a single addition of a line of code. The commit message above the diff is "@ -560,6 +560,7 @@ swString* swoole_file". The code changes are: line 560: "swString *content = swString_new(filesize);", line 561: "if (!content)", line 562: "{", line 563: "+ close(fd);", line 564: "return NULL;", line 565: "}". At the bottom, there are statistics: 827 watches, 8,803 stars, 2,124 forks, and a "Watch" button.

What We Do

- Local reasoning of program fragments

for Program Repair!

- Heap defects
 - Resource Leaks
 - Memory Leaks
 - Null dereferences

What We Do

- Local reasoning of program fragments

for Program Repair!

- Heap defects
 - Resource Leaks
 - Memory Leaks
 - Null dereferences

Pre/Post fix
specification

What We Do

- Local reasoning of program fragments

for Program Repair!

- Heap defects
 - Resource Leaks
 - Memory Leaks
 - Null dereferences
- 
- Separation Logic

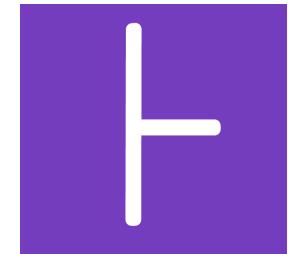
Static Analysis with Separation Logic

Static Analysis with Separation Logic

- Separation Logic reasons about heap state

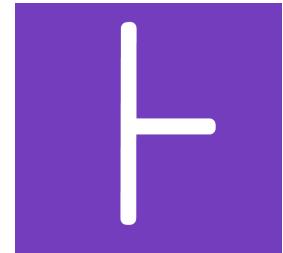
Static Analysis with Separation Logic

- Separation Logic reasons about heap state



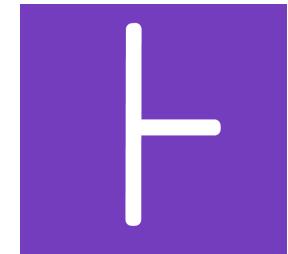
Static Analysis with Separation Logic

- Separation Logic reasons about heap state
- Bug types
 - Resource Leaks
 - Memory Leaks
 - Null Derefs



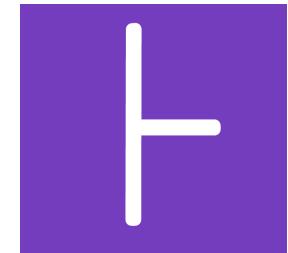
Static Analysis with Separation Logic

- Separation Logic reasons about heap state
- Bug types
 - Resource Leaks
 - Memory Leaks
 - Null Derefs
- C/C++, Java, Objective C



Static Analysis with Separation Logic

- Separation Logic reasons about heap state
- Bug types
 - Resource Leaks
 - Memory Leaks
 - Null Derefs
- C/C++, Java, Objective C



Smallfoot IR

Matching Semantic Effects in Intermediate Analysis Result

Match Fixing Properties

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```



This procedure does
good things

Match Fixing Properties

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

Inferring a Procedure Specification

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

Inferring a Procedure Specification

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

2 Specifications inferred

PRE:

n != null ;

n → Node

POST:

n != null ;

n → Node;

return = n

Inferring a Procedure Specification

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

2 Specifications inferred

PRE:

n != null ;

n → Node

Object “Node” allocated
on the heap

POST:

n != null ;

n → Node;

return = n

Match Fixing Properties

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

2 Specifications inferred

PRE:

n != null ;

n → Node

POST:

n != null ;

n → Node;

return = n

PRE:

n → null

POST:

n = null;

return = java.lang.IllegalArgumentException

Match Fixing Properties

```
// [n] may be null
public static Node checkNotNull(Node n) {
    if (n == null) {
        throw new IllegalArgumentException("Bad Arg");
    }
    return n;
}
```

Desirable property:
Throw Exn when Node is null

2 Specifications inferred

PRE:

n != null ;

n → Node

POST:

n != null ;

n → Node;

return = n

PRE:

n → null

POST:

n = null;

return = java.lang.IllegalArgumentException

Match!

Compositionality Matters

Google error-prone

Compositionality Matters

Compositionality Matters

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        checkGuardedBy(method != null, identifier.toString());  
  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Compositionality Matters



```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        checkGuardedBy(method != null, identifier.toString());  
  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Compositionality Matters

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        // method may be null  
        checkGuardedBy(method != null, identifier.toString());  
  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Cannot be inferred here!

Compositionality Matters

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
PRE: {method == null}  
checkGuardedBy(method != null, identifier.toString());  
POST: {throw new IllegalArgumentException()}  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Inferred here!

```
public static void checkGuardedBy(boolean condition, String message) {  
    if (!condition) {  
        throw new IllegalGuardedBy(message);  
    }  
}
```

Match Fixing Fragments

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
PRE: {method == null}  
checkGuardedBy(method != null, identifier.toString()); Match ✓  
POST: {throw new IllegalArgumentException()}  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Apply Fixing Fragments

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
PRE: {method == null}  
checkGuardedBy(method != null, identifier.toString()); Match ✓  
POST: {throw new IllegalArgumentException()}  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
// NULL DEREference ERROR  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Apply Fixing Fragments

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
PRE: {method == null}  
checkGuardedBy(method != null, identifier.toString()); Match ✓  
POST: {throw new IllegalArgumentException()}  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Validate Fixing Fragments

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
PRE: {method == null}  
checkGuardedBy(method != null, identifier.toString()); Match ✓  
POST: {throw new IllegalArgumentException()}  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
checkGuardedBy(method != null, identifier.toString()); Validate ✓  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

How We Formulate Repair

Repair in the Abstract

Repair in the Abstract

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

IR Command C

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

IR Command C

```
swHashMap *map = malloc(sizeof(swHashMap));
...
if (error) {
    // map allocoed but not freed
    return;
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

Fault-inducing Heap state

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

{ map \Rightarrow alloced }

```
swHashMap *map = malloc(sizeof(swHashMap));
...
if (error) {
    // map alloced but not freed
    return;
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

{ map \Rightarrow alloced }

```
swHashMap *map = malloc(size) // Heap predicates
...
if (error) {
    // map alloced but not freed
    return;
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$

Interpretation step

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault}$



Bug report

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \rightsquigarrow^* C_\ell, H_{Good}$$

Fixing transformation T

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \implies T_{\ell'}, H \rightsquigarrow^* C_\ell, H_{Good}$$

Produce a “Good” heap state

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

Fault-avoiding interpretation

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

How to find T?

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map allocoed but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

Additive transformation

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
...
if (error) {
+ sw_free(map);
    return;
}
```

ℓ

Repair in the Abstract

$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'} H \rightsquigarrow^* C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$

At the reported error location

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
...
if (error) {
    sw_free(map);
    return;
}
```

ℓ'
 ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

{ map \Rightarrow freed }

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map alloced but not freed  
    return;  
}
```

ℓ

Repair in the Abstract

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

{ map \Rightarrow allocated }

{ map \Rightarrow freed }

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
```

...

```
if (error) {  
    // map allocated but not freed  
    return;  
}
```

ℓ

Semantic Repair Search

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

{ map \Rightarrow alloced } ?C { map \Rightarrow freed }

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    // map alloced but not freed  
    return;  
}
```

ℓ

Semantic Repair Search

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

{ *pvar* \Rightarrow allocated } ?C { *pvar* \Rightarrow freed }

sw_free(*pvar*);

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
```

...

```
if (error) {
    // map alloced but not freed
    return;
}
```

ℓ

Semantic Repair Search

$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$

{ $pvar \Leftrightarrow \text{alloced}$ } ?C { $pvar \Leftrightarrow \text{freed}$ }

sw_free($pvar$);

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
```

...

```
if (error) {
```

```
    sw_free(map);
```

```
    return;
```

```
}
```



ℓ

Semantic Repair Search

$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$

{ *pvar* \Leftrightarrow allocated } ?C { *pvar* \Leftrightarrow freed }

sw_free(*pvar*);

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
...
if (error) {
    sw_free(map);
    return;
}
```

```
#define sw_free(ptr)
if (ptr) {
    free(ptr);
    ptr = NULL;
    swWarn("free");
}
```

Semantic Repair Search

$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H' \rightsquigarrow \text{no fault}$

Type “swHashMap *”

{ $pvar \Leftrightarrow \text{alloced}$ } ?C { $pvar \Leftrightarrow \text{freed}$ }

sw_free($pvar$);

```
swHashMap *map = sw_malloc(sizeof(swHashMap));
...
if (error) {
    sw_free(map);
    return;
}
```

Type “void *”

```
#define sw_free(ptr)
if (ptr) {
    free(ptr);
    ptr = NULL;
    swWarn("free");
}
```

Semantic Repair Search

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \implies T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$
$$\{ pvar \Rightarrow \text{alloced} \} \quad ?C \quad \{ pvar \Rightarrow \text{freed} \}$$

Semantic Repair Search

$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \implies T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$
$$\{ pvar \Leftrightarrow \text{open} \} \quad ?C \quad \{ pvar \Leftrightarrow \text{close} \}$$

Semantic Repair Search

$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \stackrel{*}{\rightsquigarrow} C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$

{ F }

?C

{ F' }

Local Reasoning & Separation Logic

{ F }

?C

{ F' }

Local Reasoning & Separation Logic

{ F * P } ?C {F' * Q}

Local Reasoning & Separation Logic

$$\{ F * P \} \quad ?C \quad \{ F' * Q \}$$

C can affect other things on the heap

Local Reasoning & Separation Logic

 $\{ F * P \}$ $?C$ $\{ F' * Q \}$

C's footprint

Local Reasoning & Separation Logic

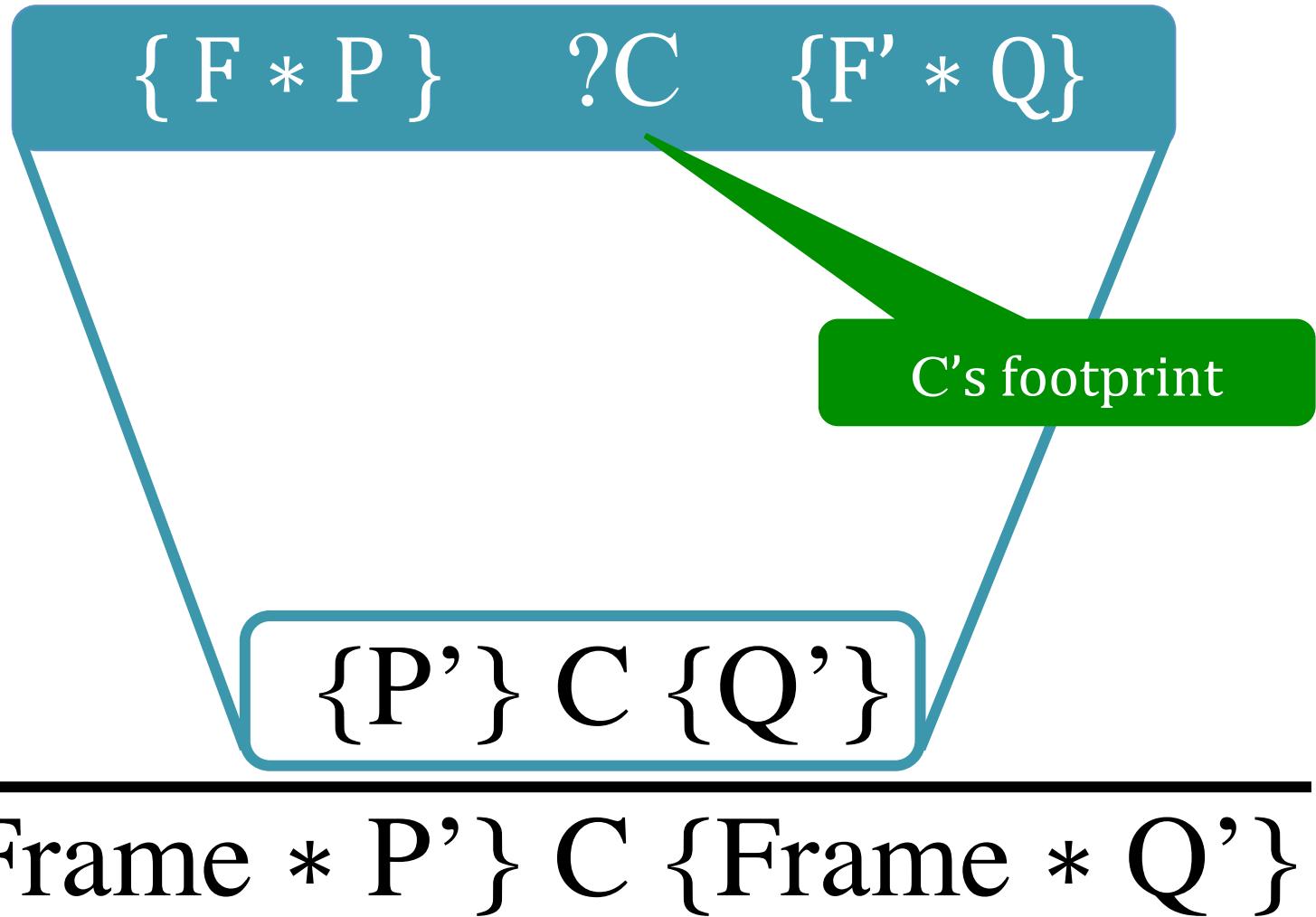
$$\{ F * P \} \quad ?C \quad \{ F' * Q \}$$

C's footprint

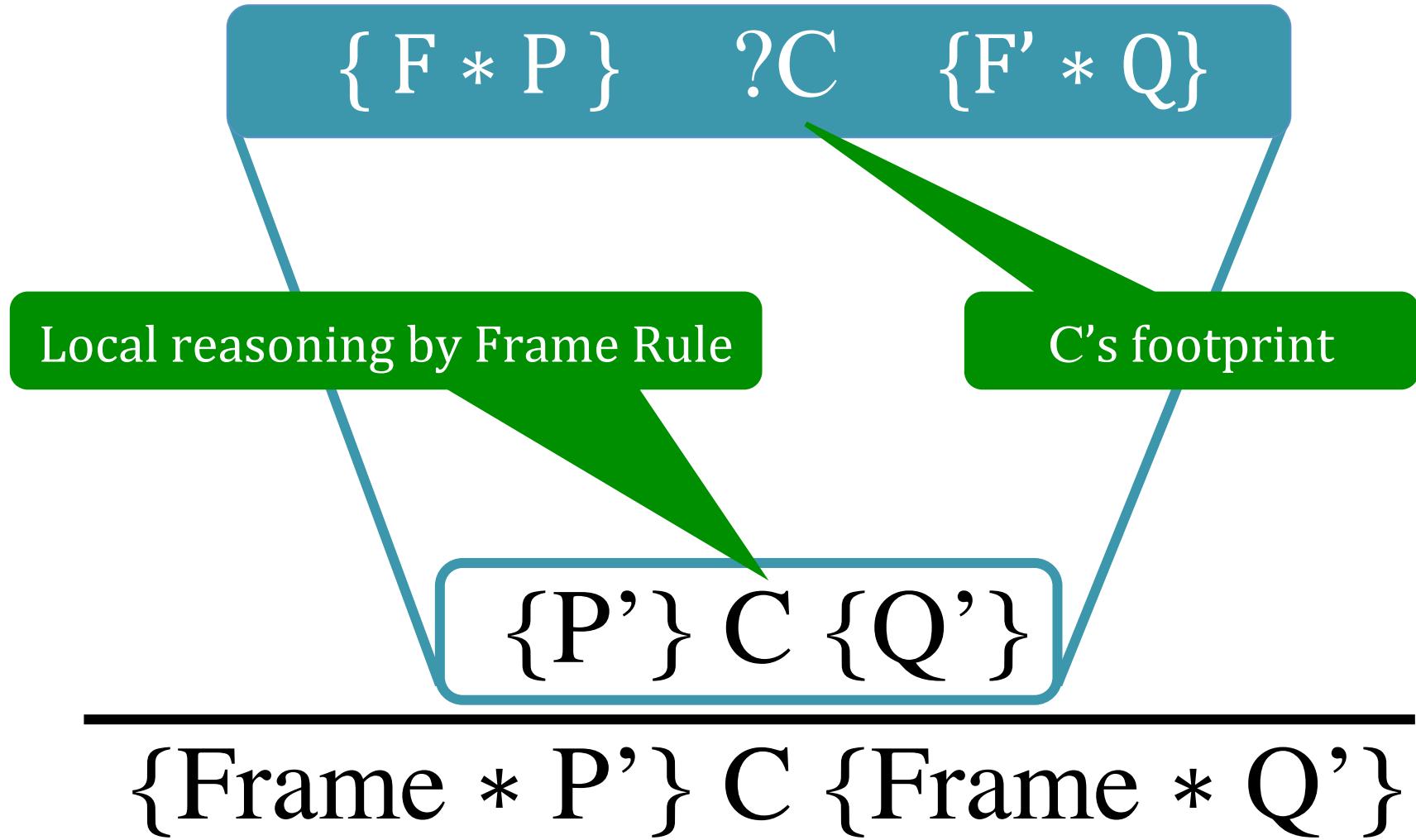
$$\{P'\} C \{Q'\}$$

$$\{ \text{Frame} * P' \} C \{ \text{Frame} * Q' \}$$

Local Reasoning & Separation Logic



Local Reasoning & Separation Logic



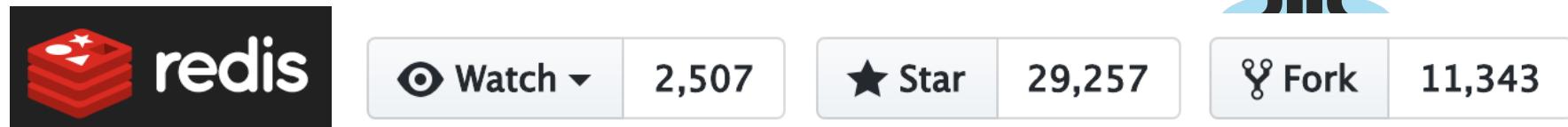
Results Summary

Results Summary

- 8 C projects and 3 Java projects

Results Summary

- 8 C projects and 3 Java projects



Results Summary

- 8 C projects and 3 Java projects



A screenshot of a GitHub repository page for the project "redis". The page shows the following metrics: Watch (2,507), Star (29,257), Fork (11,343). A large green callout box highlights the text "115KLOC".

redis

Watch 2,507

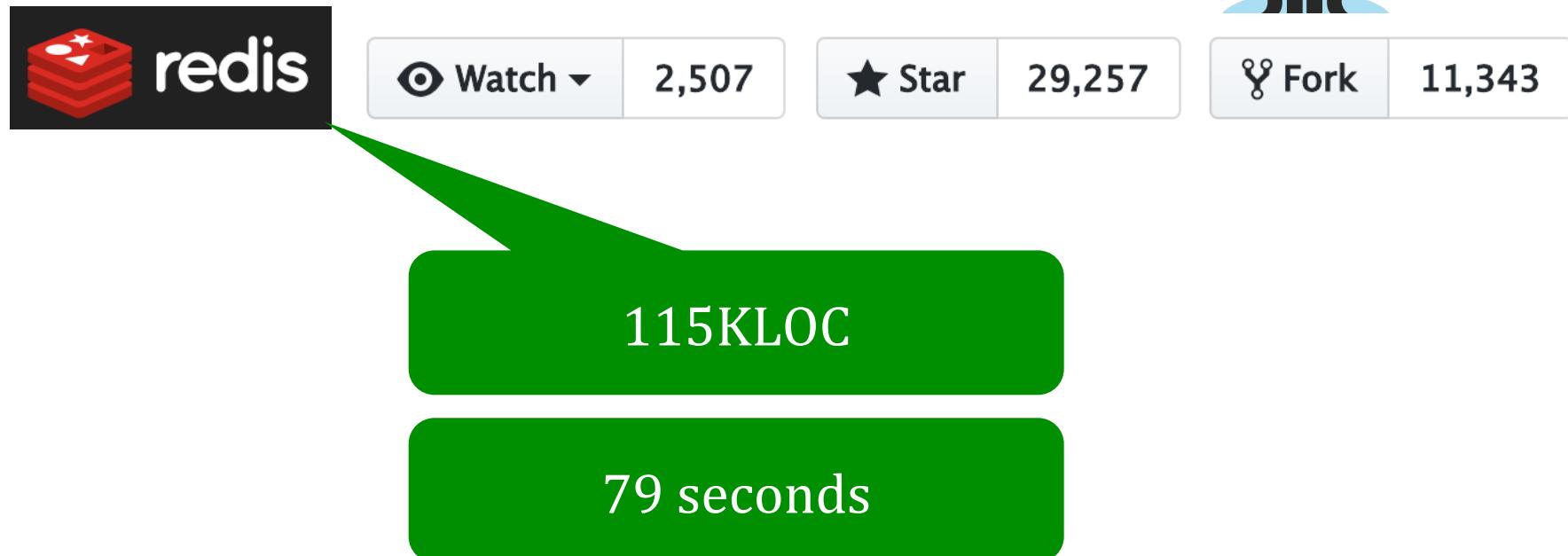
Star 29,257

Fork 11,343

115KLOC

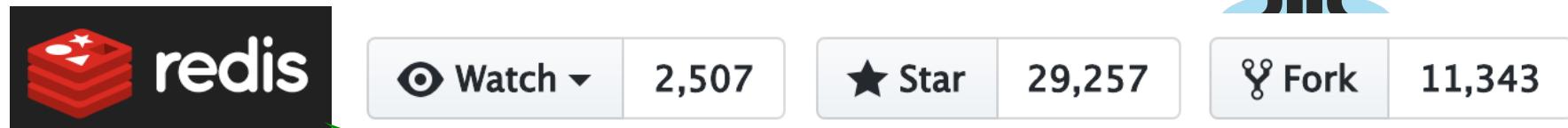
Results Summary

- 8 C projects and 3 Java projects



Results Summary

- 8 C projects and 3 Java projects



115KLOC

79 seconds

6 resource leak fixes

Results Summary

- 8 C projects and 3 Java projects



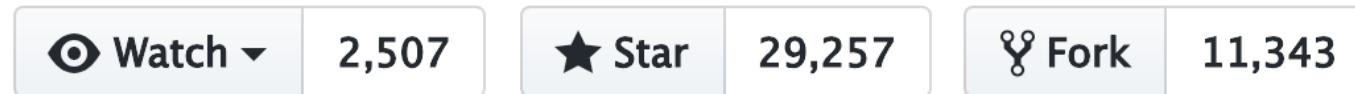
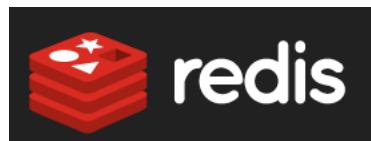
Watch ▾ 2,507 Star 29,257 Fork 11,343



Watch ▾ 836 Star 8,973 Fork 2,137

Results Summary

- 8 C projects and 3 Java projects



Results Summary

- 8 C projects and 3 Java projects



Watch ▾ 2,507 Star 29,257 Fork 11,343



Watch ▾ 836 Star 8,973 Fork 2,137



Watch ▾ 501 Star 6,288 Fork 1,790



Watch ▾ 201 Star 2,162 Fork 700

Results Summary

- 8 C projects and 3 Java projects



Fix memory leak in _attach_new_segment #528

Merged rhc54 merged 1 commit into pmix:master from footpatch:patch-1

Fix file descriptor leak #45

Merged KoffeinFlummi merged 1 commit into KoffeinFlummi:master from footpatch:patch-1

Close input stream in SmaliMod.java #1599

Merged iBotPeaches merged 1 commit into iBotPeaches:master from footpatch:patch-1

Results Summary

Project	Lang	KLOC	Time (s)	Bug	Fixes	False Pos
Swoole	C	44	20	Res. Leak	1	0
				Mem. Leak	6	3
lxc	C	63	51	Res. Leak	1	0
				Mem. Leak	0	1
Apktool	Java	15	584	Res. Leak	1	0
dabloods	C	1	9	Res. Leak	7	0
php-cp	C	9	20	Res. Leak	1	0
armake	C	16	10	Res. Leak	4	0
sysstat	C	24	28	Res. Leak	1	0
redis	C	115	79	Res. Leak	6	0
rappel	C	2	7	Mem. Leak	1	0
error-prone	Java	149	262	Null Deref	2	0
jfreechart	Java	282	1,268	Null Deref	22	0

Results Summary

Project	Lang	KLOC	Time (s)	Bug	Fixes	False Pos
Swoole	C	44	20	Res. Leak	1	0
				Mem. Leak	6	3
lxc	C	63	51	Res. Leak	1	0
				Mem. Leak	0	1
Apktool				Res. Leak	1	0
dabloods	C	1	9	Res. Leak	7	0
php-cp	C	9	20	Res. Leak	1	0
armake	C	16	10	Res. Leak	4	0
sysstat	C	24	28	Res. Leak	1	0
redis	C	115	79	Res. Leak	6	0
rappel	C	2	7	Mem. Leak	1	0
error-prone	Java	149	262	Null Deref	2	0
jfreechart	Java	282	1,268	Null Deref	22	0

4 false positives

Results Summary

Project	Lang	KLOC	Time (s)	Bug	Fixes	False Pos
Swoole	C	44	20	Res. Leak	1	0
				Mem. Leak	6	3
lxc		63	51	Res. Leak	1	0
				Mem. Leak	0	1
Apktool				Mem. Leak	1	0
dabloods				Mem. Leak	7	0
php-cp				Mem. Leak	1	0
armake				Mem. Leak	4	0
sysstat				Mem. Leak	1	0
redis				Mem. Leak	6	0
rappel	C	2	7	Mem. Leak	1	0
error-prone	Java	149	262	Null Deref	2	0
jfreechart	Java	282	1,268	Null Deref	22	0

55 fixes

- 24 resource leaks
- 7 memory leaks
- 24 null dereferences

Summary

Static Analysis for APR

A bug in error-prone

```
public class GuardedByBinder {  
    case IDENTIFIER: {  
        Symbol.MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        checkGuardedBy(method != null, identifier.toString());  
        return bindSelect(computeBase(context, method), method);  
    } case MEMBER_SELECT: {  
        ...  
        MethodSymbol method =  
            context.resolver.resolveMethod(node, identifier.getName());  
        if (method == null) {  
            throw new IllegalArgumentException("Method may be null");  
        }  
        return bindSelect(computeBase(context, method), method);  
    }  
}
```

Fixed + Test Added

Fixed 18 months later!

Repair in the Abstract

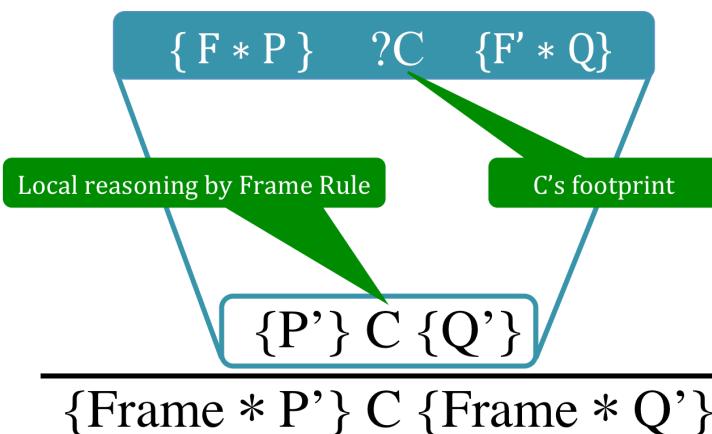
$$C_\ell, H_{Bad} \rightsquigarrow \text{fault} \Rightarrow T_{\ell'}, H \rightsquigarrow^* C_\ell, H_{Good} \not\rightsquigarrow \text{fault}$$

{ pvar \Rightarrow alloced } ?C { pvar \Rightarrow freed }

sw_free(pvar);

```
swHashMap *map = sw_malloc(sizeof(swHashMap));  
...  
if (error) {  
    sw_free(map);  
    return;  
}
```

Local Reasoning & Separation Logic



Fix Real Bugs

- 8 C projects and 3 Java projects



Watch 2,500 Star 29,049 Fork 11,245



Watch 826 Star 8,788 Fork 2,122



Watch 492 Star 6,214 Fork 1,782



Watch 202 Star 2,147 Fork 697