Static Automated Program Repair for Heap Properties

Rijnard van Tonder and Claire Le Goues

Carnegie Mellon University

squaresLab
Software Quality in Real Evolving Systems

isr institute for SOFTWARE RESEARCH

Motivating Example

A bug in error-prone

Motivating Example

A bug in error-prone

Motivating Example

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


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

A bug in error-prone

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

A bug in error-prone

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);
   }
}
```

A bug in error-prone

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);
   }
}
```

A bug in error-prone

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);
   }
}
```

A bug in error-prone

Motivating Example

Fixed + Test Added

```
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);
   }
}
```

A bug in error-prone

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!

A bug in error-prone

Motivating Example

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

      eckGuardedBy(method != null, identifier.toString());
      return bindSelect(computeBase(context, method), method);
   } case MEMBER_SELECT: {
      ...
      MethodSymbol method =
          t.resolver.resolveMethod(node, identifier.getName());
      ethod 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!

A bug in error-prone

Motivating Example

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

      eckGuardedBy(method != null, identifier.toString());
      return bindSelect(computeBase(context, method), method);
   } case MEMBER_SELECT: {
      ...
      MethodSymbol method =
          t.resolver.resolveMethod(node, identifier.getName());
      ethod may be null
      return bindSelect(computeBase(context, method), method);
   }
}
```

Fixed + Test Added

Fixed 18 months later!

A bug in error-prone

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);
   }
}
```

Local semantic effects

Fixed 18 months later!

A bug in error-prone

Motivating Example

```
public class GuardedByBinder {
   case IDENTIFIER: {
      Symbol.MethodSymbol method
        context.resolver.resolveM    d(node, id

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

Local semantic effects

Pre & Postconditions

Fixed 18 months later!

A bug in error-prone

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

A bug in error-prone

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

A bug in error-prone

Motivating Example

```
public class GuardedByBinder {
   case IDENTIFIER: {
      Symbol.MethodSymbol method
        context.resolver.resolveMe    d(node, id

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

        ymbol method =
        context.resolver.resolveMethod(node, identifier.getName());
        checkGuardedBy(method != null, identifier.toString());
        return bindSelect(computeBase(context, method), method);
   }
}
```

Local semantic effects

Pre & Postconditions

Right now.

What We Do

- Local reasoning of program fragments

for Program Repair!

What We Do

- Local reasoning of existing program fragments

for Program Repair!

What We Do

- Local reasoning of program fragments

for Program Repair!

What We Do

- Local reasoning of program fragments

for Program Repair!

No tests

What We Do

- Local reasoning of program fragments

for Program Repair!

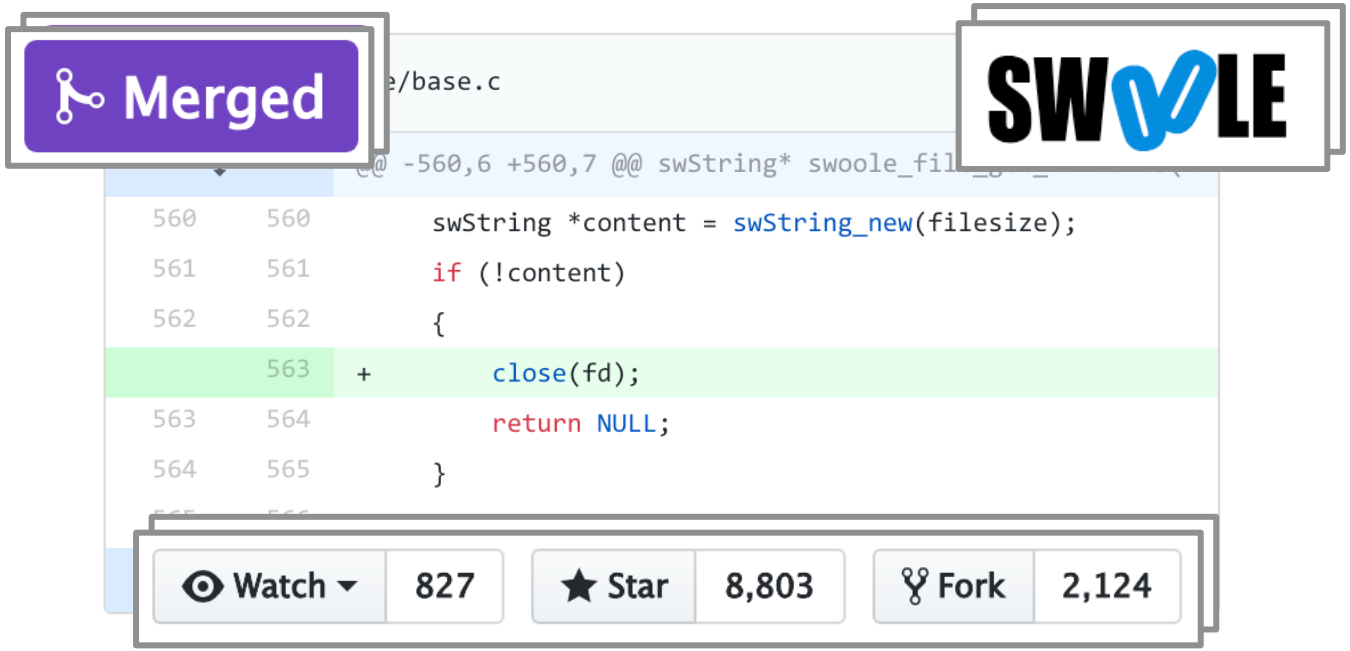
No tests

Real programs

What We Do

- Local reasoning of program fragments

for Program Repair!



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

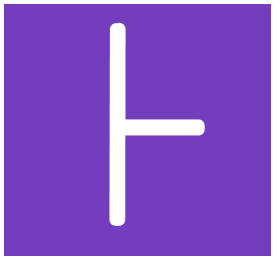
Static Analysis with Separation Logic

- Separation Logic reasons about heap state

Static Analysis with Separation Logic

- Separation Logic reasons about heap state

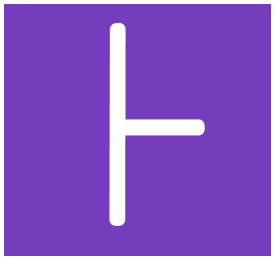
- Bug types
  - Resource Leaks
  - Memory Leaks
  - Null Derefs



Static Analysis with Separation Logic

- Separation Logic reasons about heap state

- Bug types
  - Resource Leaks
  - Memory Leaks
  - Null Derefs

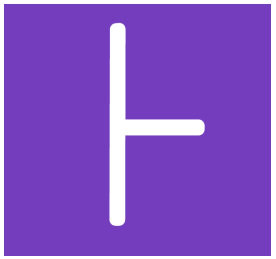


- C/C++, Java, Objective C

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.IllegalArgException

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

Match!

PRE:
n → null

POST:
n = null;
return = java.lang.IllegalArgException

Compositionality Matters

Google error-prone

Compositionality Matters

Google error-prone

Compositionality Matters

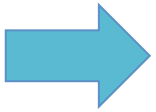
```
public class GuardedByBinder {
   case IDENTIFIER: {
      Symbol.MethodSymbol method =
        context.resolver.resolveMethod(node, identifier.getName());
      // method may be null
      checkGuardedBy(method != null, identifier.toString());

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

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

Google error-prone

Compositionality Matters



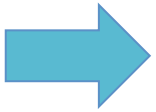
```
public class GuardedByBinder {
   case IDENTIFIER: {
      Symbol.MethodSymbol method =
        context.resolver.resolveMethod(node, identifier.getName());
      // method may be null
      checkGuardedBy(method != null, identifier.toString());

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

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

Google error-prone

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!

Google error-prone

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);
   }
}
```

Google error-prone

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());
      POST: {throw new IllegalArgumentException}
      return bindSelect(computeBase(context, method), method);
   }
}
```

Match ✓

Google error-prone

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());
      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);
   }
}
```

Match ✓

Google error-prone

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());
      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);
   }
}
```

Match ✓

Google error-prone

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());
      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);
   }
}
```

Match ✓

Validate ✓

How We Formulate Repair

Repair in the Abstract

Repair in the Abstract

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

Repair in the Abstract

$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow$ fault

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

Repair in the Abstract

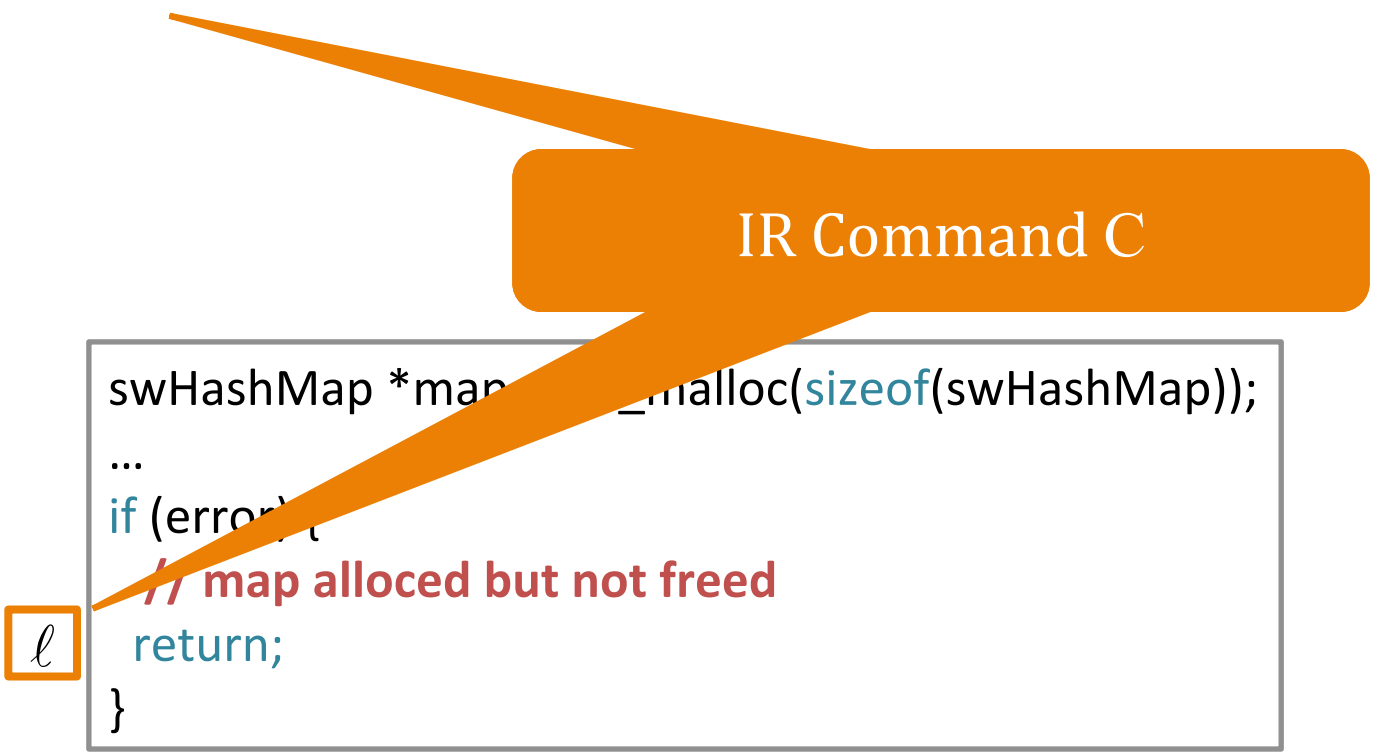
$\mathbf{C}_{\ell}, \mathbf{H}_{Bad} \rightsquigarrow$ fault

IR Command C

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

Repair in the Abstract

$\mathbf{C}_{\ell}, \mathbf{H}_{Bad} \rightsquigarrow$ fault



Repair in the Abstract

$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow$ fault

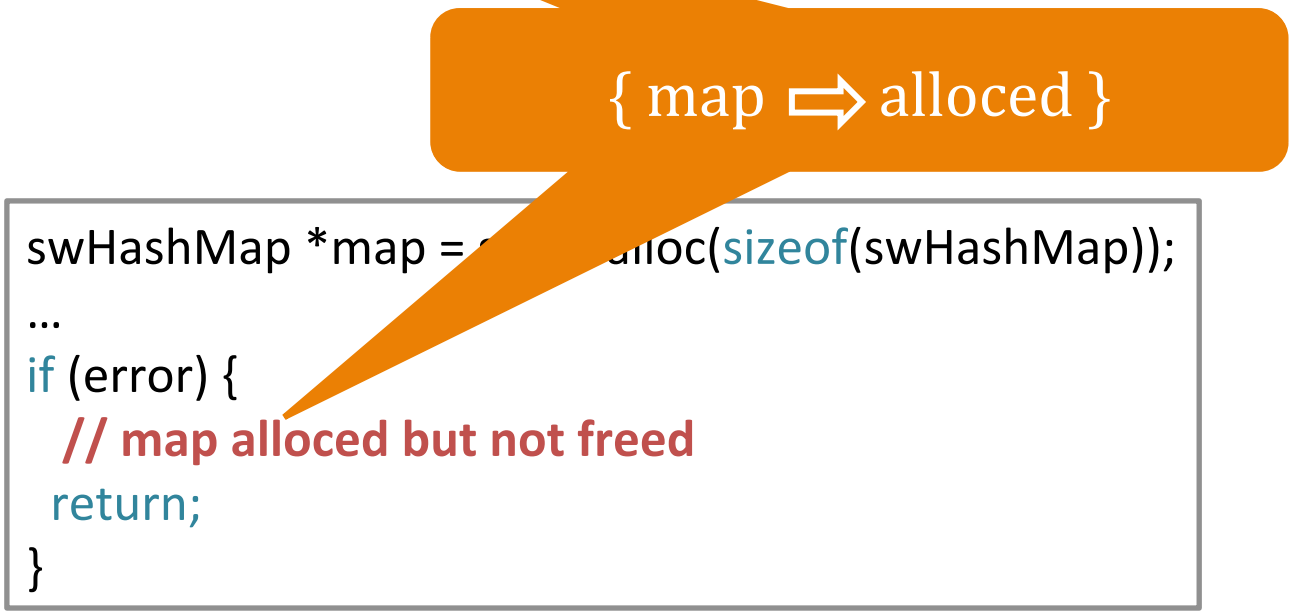
Fault-inducing Heap state

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

ℓ

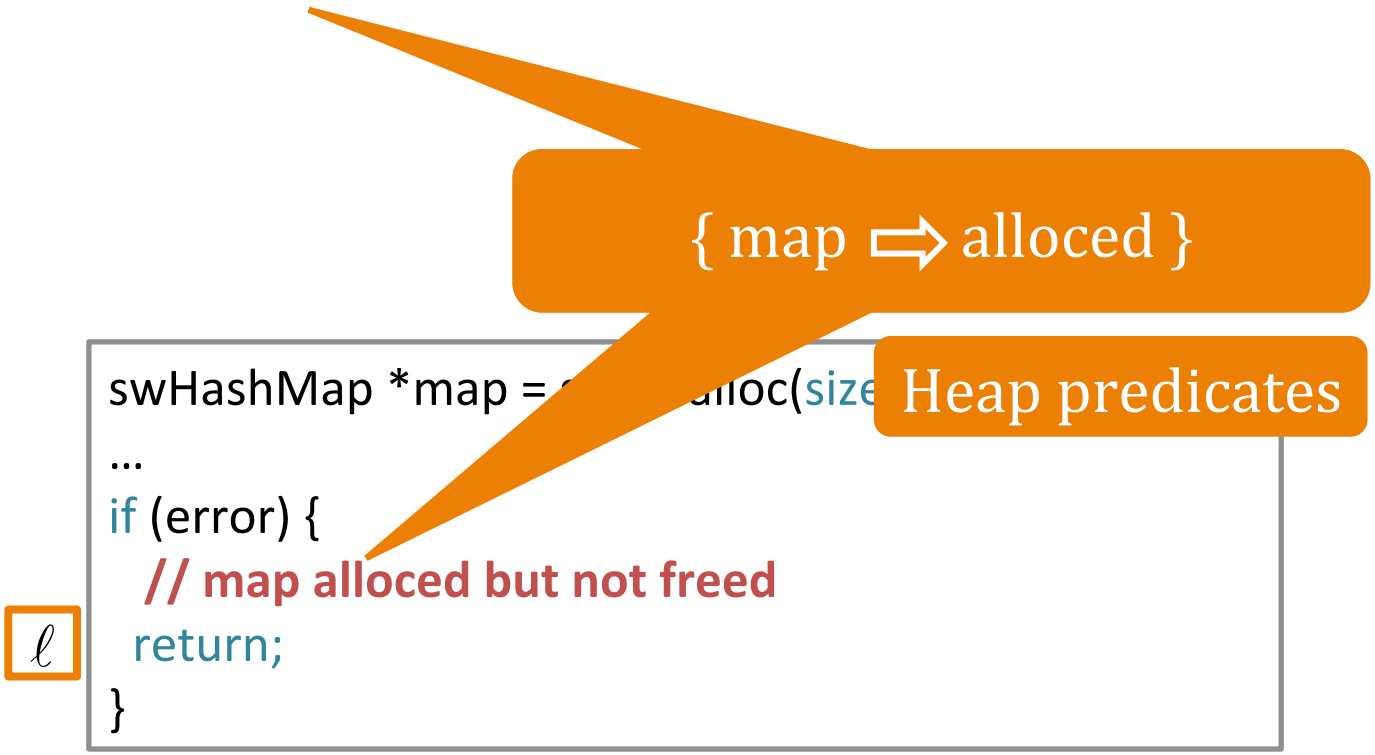
Repair in the Abstract

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



Repair in the Abstract

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



Repair in the Abstract

$\mathbf{C}_{\ell}, \mathbf{H}_{Bad} \rightsquigarrow$ fault

Interpretation step

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

ℓ

Repair in the Abstract

$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow$ fault

Bug report

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

ℓ

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good}$$

Fixing transformation T

ℓ

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

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good}$$

Produce a "Good" heap state

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

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

Fault-avoiding interpretation

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

ℓ

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

How to find T?

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

ℓ

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

Additive transformation

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

ℓ

Repair in the Abstract

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

At the reported error location

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

ℓ'

ℓ

Repair in the Abstract

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

{ map ⇨ freed }

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

ℓ

Repair in the Abstract

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

{ map ⇨ alloced }

{ map ⇨ freed }

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

ℓ

Semantic Repair Search

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

{ map ⇨ alloced } ?C { map ⇨ freed }

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

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

$\{ pvar \Rightarrow \text{alloced} \}$ $?\mathrm{C}$ $\{ pvar \Rightarrow \text{freed} \}$

sw_free(*pvar*);

ℓ

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

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

$\{ pvar \Rightarrow \text{alloced} \}$ $?\mathrm{C}$ $\{ pvar \Rightarrow \text{freed} \}$

sw_free(*pvar*);

ℓ

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

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

{ *pvar* ⇨ alloced } ?C { *pvar* ⇨ freed }

sw_free(*pvar*);

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

ℓ

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

Semantic Repair Search

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

{ *pvar* ⇨ alloced } ?C { *pvar* ⇨ freed }

Type "swHashMap *"

sw_free(***pvar***);

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

ℓ

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

Type "void *"

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \;\Rightarrow\; \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

$$\{\, pvar \Rightarrow \text{alloced} \,\} \;\; ?\mathrm{C} \;\; \{\, pvar \Rightarrow \text{freed} \,\}$$

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

$$\{\, pvar \Rightarrow \text{open} \,\} \quad ?\mathrm{C} \quad \{\, pvar \Rightarrow \text{close} \,\}$$

Semantic Repair Search

$$\mathrm{C}_{\ell}, \mathrm{H}_{Bad} \rightsquigarrow \text{fault} \Rightarrow \mathrm{T}_{\ell'}, \mathrm{H} \overset{*}{\rightsquigarrow} \mathrm{C}_{\ell}, \mathrm{H}_{Good} \not\rightsquigarrow \text{fault}$$

$$\{ \mathrm{F} \} \quad ?\mathrm{C} \quad \{ \mathrm{F}' \}$$

Local Reasoning & Separation Logic

$$\{ F \} \quad ?C \quad \{ F' \}$$

Local Reasoning & Separation Logic

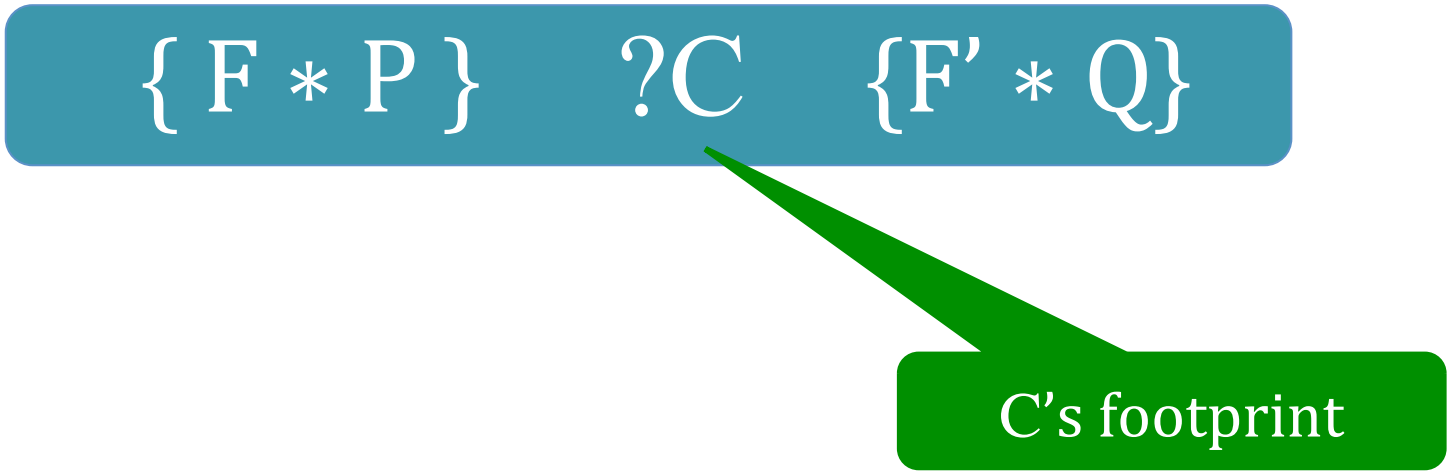
$$\{ F * P \} \quad ?C \quad \{F' * Q\}$$

Local Reasoning & Separation Logic

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

C can affect other things on the heap

Local Reasoning & Separation Logic



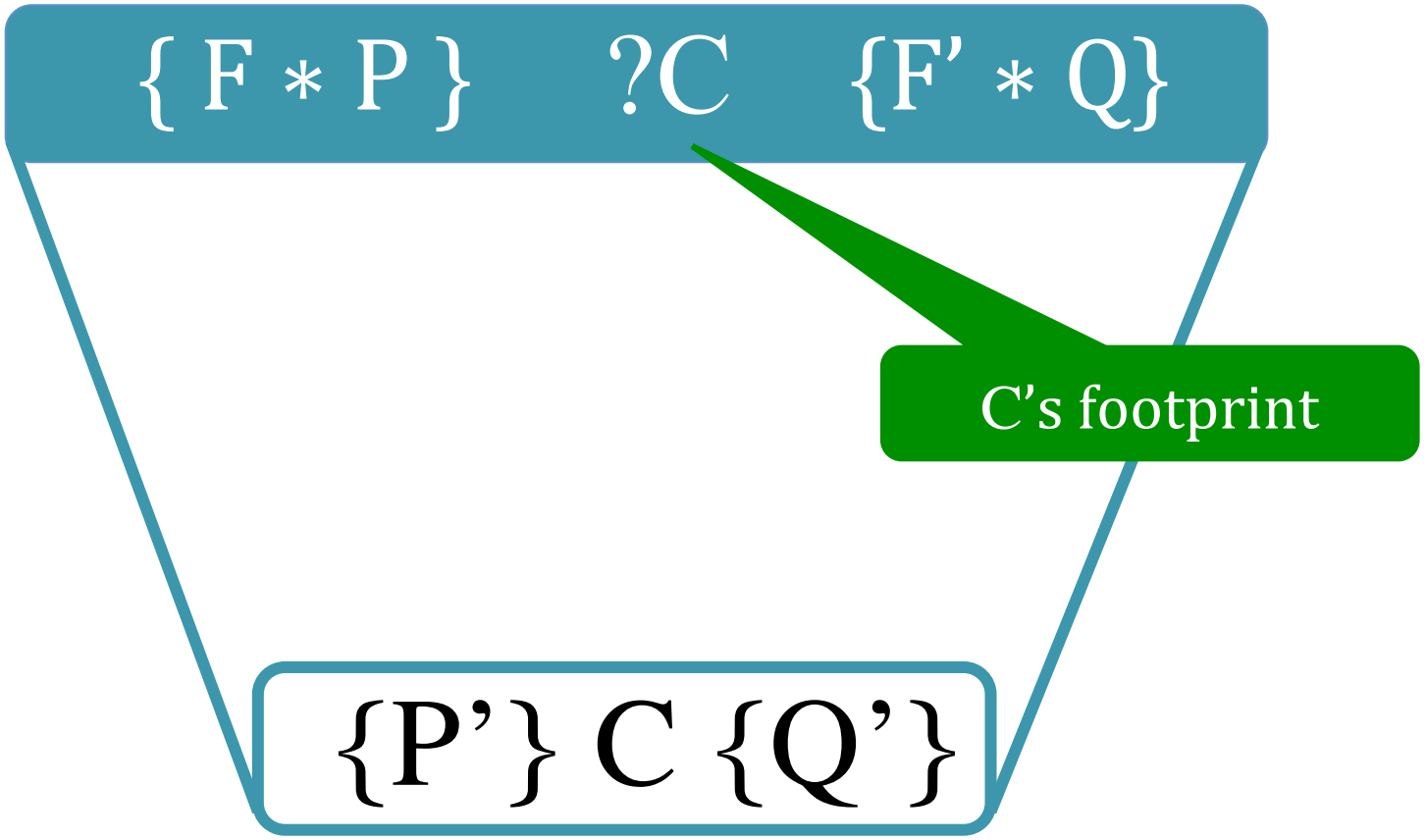
Local Reasoning & Separation Logic

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

C's footprint

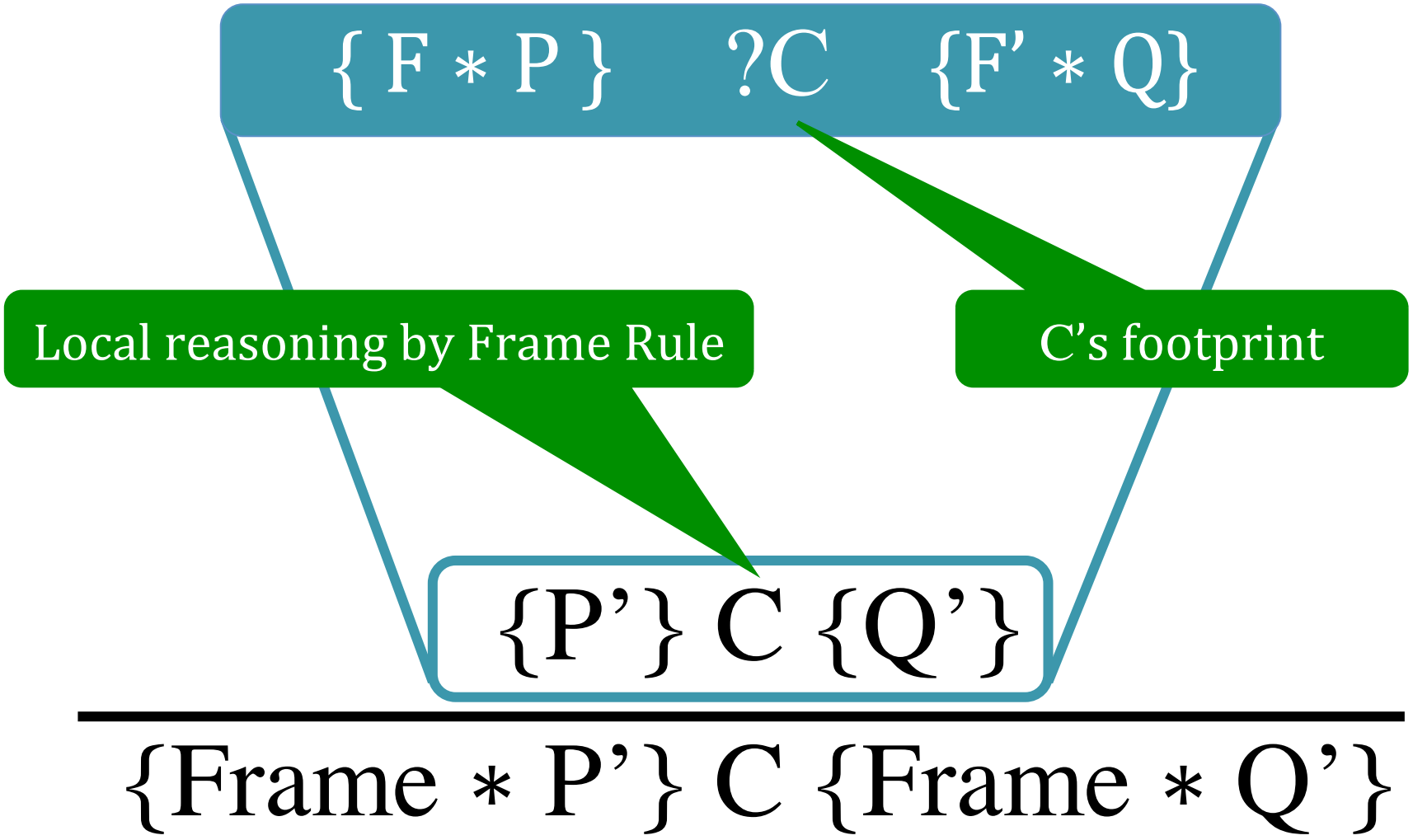
$$\frac{\{P'\}\ C\ \{Q'\}}{\{Frame * P'\}\ C\ \{Frame * Q'\}}$$

Local Reasoning & Separation Logic



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

Local Reasoning & Separation Logic



Results Summary

Results Summary

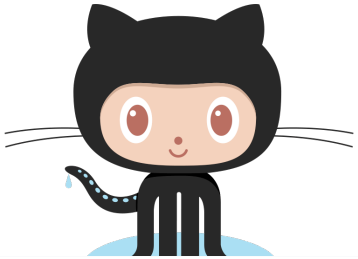
- 8 C projects and 3 Java projects

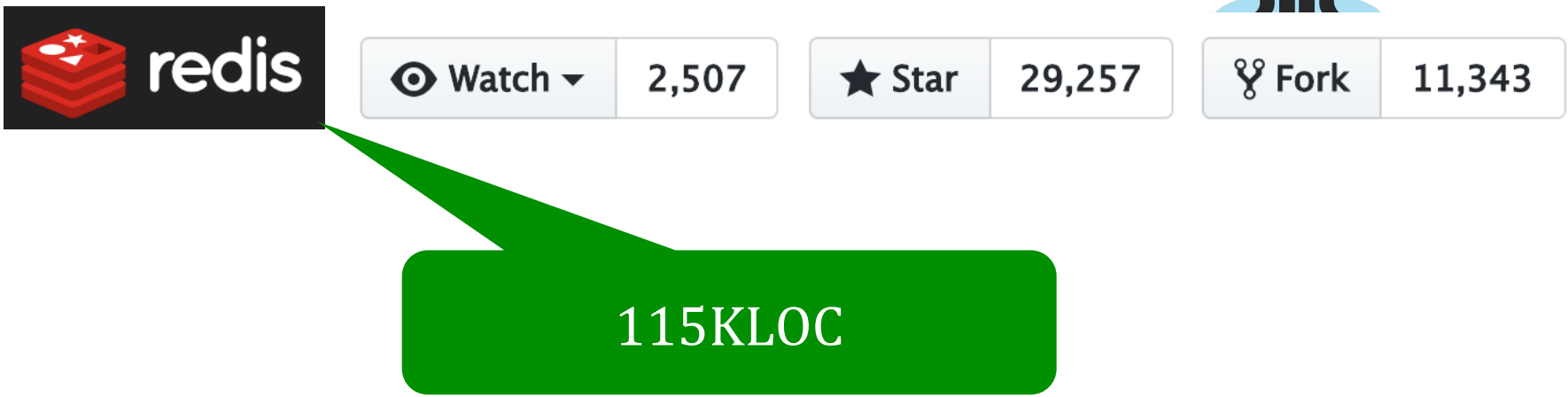
Results Summary

- 8 C projects and 3 Java projects

Results Summary

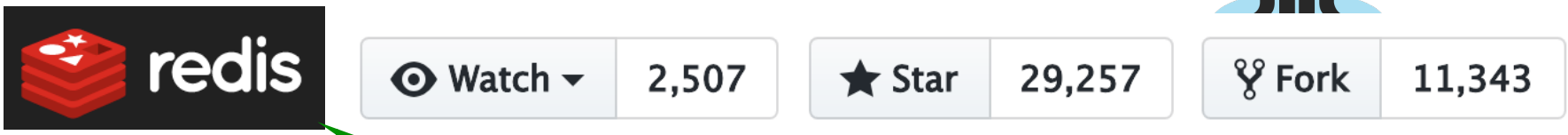
- 8 C projects and 3 Java projects





Results Summary

- 8 C projects and 3 Java projects

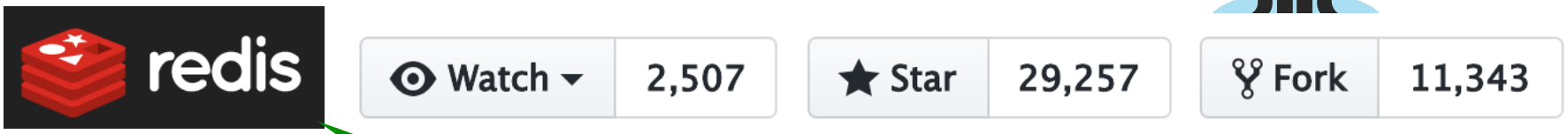


115KLOC

79 seconds

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

redis

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

SWOLE

Watch 836 Star 8,973 Fork 2,137

Results Summary

- 8 C projects and 3 Java projects

redis

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

SWOLE

Watch 836 Star 8,973 Fork 2,137

APKTOOL

Watch 501 Star 6,288 Fork 1,790

Results Summary

- 8 C projects and 3 Java projects

redis — Watch 2,507 | Star 29,257 | Fork 11,343

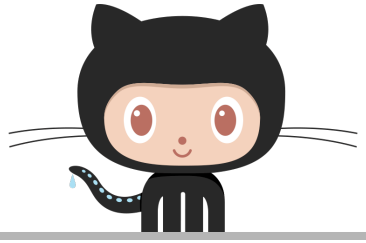
SWOLE — Watch 836 | Star 8,973 | Fork 2,137

APKTOOL — Watch 501 | Star 6,288 | Fork 1,790

LXC — 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
dablooms	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		63	51	Res. Leak	1	0
				eak	0	1
Apktool				ak	1	0
dablooms	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

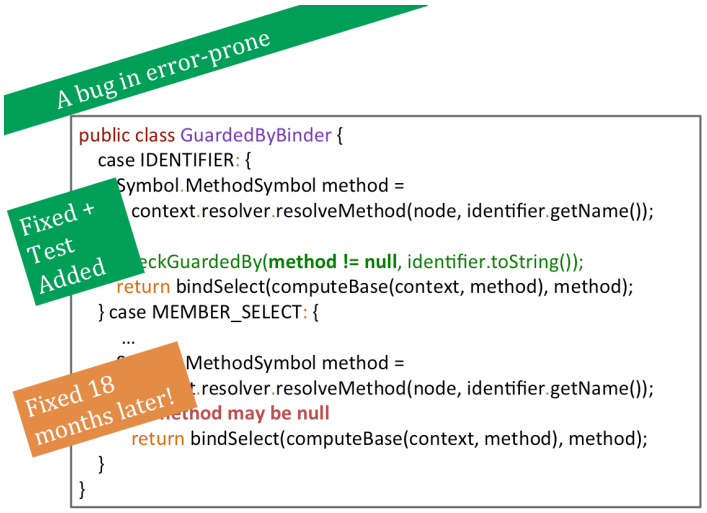
Results Summary

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
				eak	0	1
Apktool				ak	1	0
dablooms				ak	7	0
php-cp				ak	1	0
armake				ak	4	0
sysstat				ak	1	0
redis				eak	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

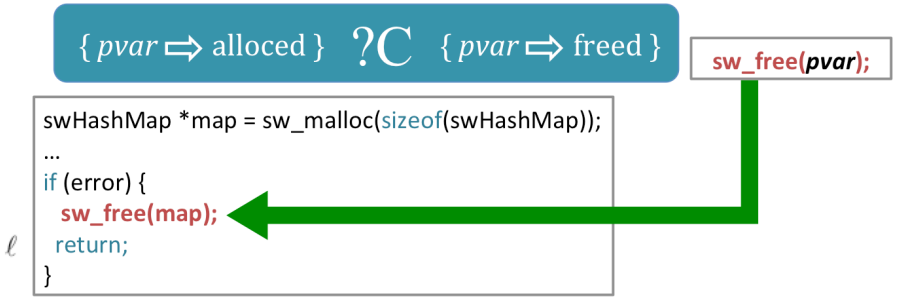
Summary

Static Analysis for APR

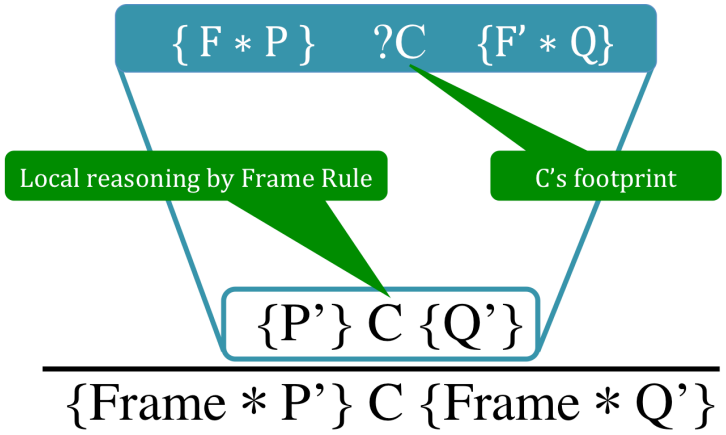


Repair in the Abstract

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

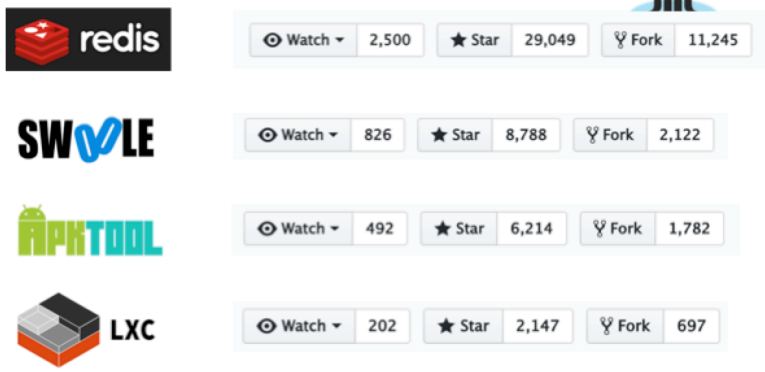


Local Reasoning & Separation Logic



Fix Real Bugs

- 8 C projects and 3 Java projects



https://github.com/squaresLab/footpatch