Static Automated Program Repair for Heap Properties

by Rijnard van Tonder and Claire le Goues

presented by Bruno Martins (bm2787) - COMSW6156 Spring 2019

Problem statement

Problem statement

- Programs have bugs!

Problem statement

- Programs have bugs!
- Static analysis tools can **find** and **report** certain kinds of bugs

Problem statement

- Programs have bugs!
- Static analysis tools can **find** and **report** certain kinds of bugs
- What if a tool could also *automatically* **fix** the bugs it finds?

Problem statement

- Programs have bugs!
- Static analysis tools can **find** and **report** certain kinds of bugs
- What if a tool could also *automatically* **fix** the bugs it finds?
    - Automated Program Repair

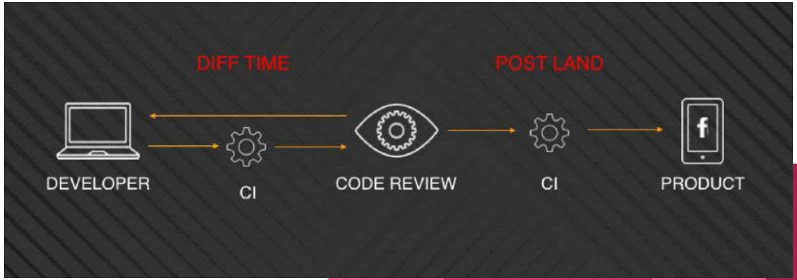
Background: Infer

- Static analysis tool by Facebook
- Extensible
- Works by converting source code to an Intermediate Language:

  Smallfoot Intermediate Language (SIL)

Background: Infer

- Static analysis tool by Facebook
- Extensible
- Works by converting source code to an Intermediate Language



One of the tools mentioned in *"From Start-ups to Scale-ups: Opportunities and Open Problems for Static and Dynamic Program Analysis"*

Background: Infer

Infer in action

Background: Separation Logic

- Infer works by using **Separation Logic** to extract **specifications** from each **command** in the source code, which can then be composed into **procedure summaries**.
  - Separation Logic: formal way to reason about a program's manipulation of its state (memory)
  - Specification: a triplet in the form:
    - `{P}C{Q}` where P is the current heap state, C is the **command** being considered and Q is the heap state after C executes
  - Procedure summary: the summary of the combined effects of all of the procedure's commands.

Background:

$$\frac{\{P\}\ C\ \{Q\}}{\{P * F\}\ C\ \{Q * F\}}\ \textsc{Frame Rule}$$

- The **Frame Rule** states that, given a heap H, the analysis of the command C with specification `{P}C{Q}` can proceed without considering the *unaffected* parts of H, namely the **frame** F.
- This enables **local reasoning**: the ability to reason only over the affected heap parts and ignore the rest.
- The process by which a frame F is found is called **frame inference**.
- The *affected* part is called **footprint**.

Background: Example

$$\frac{\{P\}\ C\ \{Q\}}{\{P * F\}\ C\ \{Q * F\}}\ \textsc{Frame Rule}$$

```
{hmap⇒undefined, root⇒undefined}
    hmap=sw_malloc(...)
{hmap⇒alloced, root⇒undefined}
```

```c
swHashMap *hmap =
  sw_malloc(sizeof(swHashMap));
if (!hmap) {
  swWarn("malloc[1] failed.");
  return NULL;
}
swHashMap_node *root =
  sw_malloc(sizeof(swHashMap_node));
if (!root) {
  swWarn("malloc[2] failed.");
  return NULL; // returns, hmap not freed
}
```

Background: Example

$$\frac{\{P\}\ C\ \{Q\}}{\{P * F\}\ C\ \{Q * F\}}\ \textsc{Frame Rule}$$

```
{hmap⇒undefined, root⇒undefined}
    hmap=sw_malloc(...)
{hmap⇒alloced, root⇒undefined}
```

Frame (F): { root }
Footprint: { hmap }

```c
swHashMap *hmap =
  sw_malloc(sizeof(swHashMap));
if (!hmap) {
  swWarn("malloc[1] failed.");
  return NULL;
}
swHashMap_node *root =
  sw_malloc(sizeof(swHashMap_node));
if (!root) {
  swWarn("malloc[2] failed.");
  return NULL; // returns, hmap not freed
}
```

Background: Separation Logic

- Armed with Separation Logic, Infer finds bugs by looking in its symbolic representation of the program for certain problematic specifications
- For example, it can find the memory leak in the previous code snippet

Let's use the notation $C_\ell, \sigma \rightsquigarrow \texttt{fault}$ to mean that the interpretation step $\rightsquigarrow$ for instruction C at location ℓ in symbolic state σ results in a $\texttt{fault}$

Background: Example

$$C_{\ell}, \sigma \rightsquigarrow \mathtt{fault}$$

- In this example, at the location **`l=11`**:

$$\mathtt{return}_{\mathtt{l}}, \{\mathtt{hmap} \Rightarrow \mathtt{alloced}\} \rightarrow \mathtt{fault}$$

```
1   swHashMap *hmap =
2     sw_malloc(sizeof(swHashMap));
3   if (!hmap) {
4     swWarn("malloc[1] failed.");
5     return NULL;
6   }
7   swHashMap_node *root =
8     sw_malloc(sizeof(swHashMap_node));
9   if (!root) {
10    swWarn("malloc[2] failed.");
11    return NULL; // returns, hmap not freed
12  }
```

Background: Example

$$C_\ell, \sigma \rightsquigarrow \texttt{fault}$$

- In this example, at the location **`l=11`**:

$$\texttt{return}_{\texttt{l}}, \{\texttt{hmap} \Rightarrow \texttt{alloced}\} \rightarrow \texttt{fault}$$

At this point, Infer would report the bug.This is where FootPatch comes in.

```
1   swHashMap *hmap =
2     sw_malloc(sizeof(swHashMap));
3   if (!hmap) {
4     swWarn("malloc[1] failed.");
5     return NULL;
6   }
7   swHashMap_node *root =
8     sw_malloc(sizeof(swHashMap_node));
9   if (!root) {
10    swWarn("malloc[2] failed.");
11    return NULL; // returns, hmap not freed
12  }
```

FootPatch

- FootPatch is an extension to the Infer tool, created by the authors of the paper
- It deals with three heap-based properties violations:
  - Null dereference
  - Resource leak
  - Memory leak
- **Once Infer finds a violation, FootPatch tries to repair it based on other similar snippets from elsewhere in the code.**

What is a Repair

- A repair satisfies:

$$C_\ell, \mathbb{H}_{Bad} \rightsquigarrow \texttt{fault} \implies T_{\ell'}, \mathbb{H} \overset{*}{\rightsquigarrow} C_\ell, \mathbb{H}_{Good} \not\rightsquigarrow \texttt{fault}$$

Where T is an additive transformation in the form of a program fragment that avoids the fault state.

Repair specification

$\{F\}C_R\{F'\}$ where:

- F: error heap configuration
- C_R: repair fragment
- F': fixed heap configuration

Repair search

$$?C_{\ell'}, \{\texttt{map} \Mapsto \mathit{alloced}\} \rightsquigarrow \texttt{return}_{\ell}, \{\texttt{map} \Mapsto \mathit{freed}\} \not\rightsquigarrow \texttt{fault} \quad (1)$$

$$\{\mathit{pvar} \Mapsto \mathit{alloced}\}\ ?C\ \ \{\mathit{pvar} \Mapsto \mathit{freed}\} \quad (2)$$

Figure 4: Modeling repair search.

Repair queries

$$\{\ pvar \mapsto Null\ \}\ ?C\ \{\ \_\ \mapsto Exn\ e\ \} \tag{5}$$

$$\{\ pvar \mapsto \langle File, Acquired \rangle\ \}\ ?C\ \{\ pvar\ \mapsto \langle File, Release \rangle\ \} \tag{6}$$

$$\{\ pvar \mapsto \langle Memory, Acquired \rangle\ \}\ ?C\ \{\ pvar\ \mapsto \langle Memory, Release \rangle\ \} \tag{7}$$

Figure 5: Repair Specifications.

Repair queries

- A key insight from the paper is that the queries can be relaxed to capture extra semantic effects by applying **frame inference** on the pre- and post-conditions to candidate repair fragments.

Repair location

- Few locations possible

```c
swHashMap *hmap =
  sw_malloc(sizeof(swHashMap));
if (!hmap) {
  swWarn("malloc[1] failed.");
  return NULL;
}
swHashMap_node *root =
  sw_malloc(sizeof(swHashMap_node));
if (!root) {
  swWarn("malloc[2] failed.");
  return NULL; // returns, hmap not freed
}
```

Repair location

- Few locations possible
- In this example, the fix could be applied either before line 10 or before line 11
- FootPatch chooses the line immediately before the fault-causing statement
- In this case, between lines 10 and 11.

```
1   swHashMap *hmap =
2     sw_malloc(sizeof(swHashMap));
3   if (!hmap) {
4     swWarn("malloc[1] failed.");
5     return NULL;
6   }
7   swHashMap_node *root =
8     sw_malloc(sizeof(swHashMap_node));
9   if (!root) {
10    swWarn("malloc[2] failed.");
11    return NULL; // returns, hmap not freed
12  }
```

Generating the patch

- Infer's Smallfoot Intermediate Language (SIL) carries type information in its symbolic statements
- Therefore, FootPatch can use that type information to decide which repair to use
- In this case, `sw_free` is used instead of `free`, since hmap is of type `swHashMap`

```c
swHashMap *hmap =
  sw_malloc(sizeof(swHashMap));
if (!hmap) {
  swWarn("malloc[1] failed.");
  return NULL;
}
swHashMap_node *root =
  sw_malloc(sizeof(swHashMap_node));
if (!root) {
  swWarn("malloc[2] failed.");
  return NULL; // returns, hmap not freed
}
```

Finding more bugs

```
     fp = fopen(rdbfilename,"r");
     ...
     if (memcmp(buf,"REDIS",5) != 0) {
         rdbCheckError("Wrong signature trying to load DB from file");
+        fclose(fp);
         return 1;
     }
     rdbver = atoi(buf+5);
     if (rdbver < 1 || rdbver > RDB_VERSION) {
         rdbCheckError("Can't handle RDB format version %d",rdbver);
+        fclose(fp);
         return 1;
     }
    ...
```

Correctness

How to verify that a patch was successful:

- Apply it and run Infer again: should result in the bug not being found
- Run the project's test suite

Results

Project	Lang	kLOC	Time (s)	ΔGL	Bug Type	Bugs	Max Cands	ΔGL	Fixes	ΔGL	FP	ΔGL
Swoole	C	44.5	20	+83	Res. Leak†	7	1	+6	1	+2	0	+0
					Mem. Leak†	20	3	+0	6	+0	3	+0
lxc	C	63.0	51	-	Res. Leak	3	5	-	1	-	0	-
					Mem. Leak	8	13	-	0	-	1	-
Apktool	Java	15.0	584	+92	Res. Leak†	19	3	+2	1	+0	0	+0
dablooms	C	1.2	9	+0	Res. Leak†	7	2	+0	7	+0	0	+0
php-cp	C	9.0	20	+5	Res. Leak†	4	3	+1	1	+0	0	+0
armake	C	16.0	10	+13	Res. Leak†	5	7	+4	4	+0	0	+0
sysstat	C	24.9	28	+10	Res. Leak	1	10	+0	1	+0	0	+0
redis	C	115.0	79	+121	Res. Leak†	8	8	+10	6	+0	0	+0
rappel	C	2.1	7	+3	Mem. Leak†	1	6	+0	1	+0	0	+0
error-prone	Java	149.0	262	+602	Null Deref	11	66	+0	2	+0	0	+0
jfreechart	Java	282.7	1,268	-	Null Deref	53	221	-	22	-	0	-

Table 1: Bugs repaired with FootPatch. "Bugs" is the number of bugs detected by Infer's static analysis. "Max Cands" is the maximum number of IL repair candidates for the bug (pre-compatibility check). "Fixes" are the number of unique patches fixing unique bugs (post check). † indicates one or more fixes for previously undiscovered bugs. "ΔGL" is the change in associated column when using the global search space.

Thank you!