ANCHOR: Fast and Precise Value-flow Analysis for Containers via Memory Orientation

CHENGPENG WANG, The Hong Kong University of Science and Technology, China
WENYANG WANG, Ant Group, China
PEISEN YAO, The Hong Kong University of Science and Technology, China
QINGKAI SHI, JINGUO ZHOU, and XIAO XIAO, Ant Group, China
CHARLES ZHANG, The Hong Kong University of Science and Technology, China

Containers are ubiquitous data structures that support a variety of manipulations on the elements, inducing the indirect value flows in the program. Tracking value flows through containers is stunningly difficult, because it depends on container memory layouts, which are expensive to be discovered.

This work presents a fast and precise value-flow analysis framework called ANCHOR for the programs using containers. We introduce the notion of anchored containers and propose the memory orientation analysis to construct a precise value-flow graph. Specifically, we establish a combined domain to identify anchored containers and apply strong updates to container memory layouts. ANCHOR finally conducts a demand-driven reachability analysis in the value-flow graph for a client. Experiments show that it removes 17.1% spurious statements from thin slices and discovers 20 null pointer exceptions with 9.1% as its false-positive ratio, while the smashing-based analysis reports 66.7% false positives. ANCHOR scales to millions of lines of code and checks the program with around 5.12 MLoC within 5 hours.

CCS Concepts: • **Theory of computation → Program analysis**; • **Software and its engineering → Software safety;**

Additional Key Words and Phrases: Abstract interpretation, value-flow analysis, data structure analysis

ACM Reference format:
Chengpeng Wang, Wenyang Wang, Peisen Yao, Qingkai Shi, Jinguo Zhou, Xiao Xiao, and Charles Zhang. 2023. ANCHOR: Fast and Precise Value-flow Analysis for Containers via Memory Orientation. *ACM Trans. Softw. Eng. Methodol.* 32, 3, Article 66 (April 2023), 39 pages.
https://doi.org/10.1145/3565800

The authors are supported by the RGC16206517, ITS/440/18FP and PRP/004/21FX grants from the Hong Kong Research Grant Council and the Innovation and Technology Commission, Ant Group, and the donations from Microsoft and Huawei. This work was finished when Qingkai Shi was with Ant Group. He is currently with Purdue University and is available via email at shi553@purdue.edu.

Authors' addresses: C. Wang, P. Yao (corresponding author), and C. Zhang, Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, New Territories, Hong Kong, China; emails: {cwangch, pyao, charlesz}@cse.ust.hk; W. Wang, Q. Shi, J. Zhou, and X. Xiao, Ant Group, No. 3239 Keyuan South Road, Shenzhen, Guangdong, China; emails: {penguin.wwy, qingkai.sqk, jinguo.zjg, xx}@antgroup.com.

1049-331X/2023/04-ART66 $15.00
https://doi.org/10.1145/3565800

1 INTRODUCTION

A container, e.g., *list*, *set*, or *map*, is an abstract data type that supports manipulating a collection of objects by its interfaces. General-purpose programming languages provide many implementations, such as the C++ STL containers [1], the Java Collections Framework (JCF) [2], and the specific classes in the Java EE framework (Java EE) [3]. Their pervasive usages require a static analyzer to reason how an object flows into and out of containers for various tasks, such as program understanding [4–6], bug detection [7–11], and debugging [12, 13].

Goal and Challenge. Our goal is to establish a fast and precise reasoning of container memory layouts for value-flow analysis. Unfortunately, the problem is always one of the "Achilles' heels" of static analysis [14]. Note that a container modification changes both which objects are stored, i.e., the ownership, and which indexes are associated with the objects, i.e., the index-value correlation. The precise reasoning of containers requires the strong updates upon container memory layouts, involving the program facts in multiple domains. First, we require a precise pointer analysis [15–17] to identify the manipulated objects, including both the containers and the elements. Second, applying strong updates to a listlike container relies on numeric analyses [18–20] to determine the relational positions of the manipulations. Third, the indexes of maplike containers are general comparable objects, and its strong updates often depend on complex relational properties of strings [21–24] and user-defined data structures [25, 26]. More importantly, the prerequisites are closely intertwined, demanding a solution to address them simultaneously. The overall quality of the results would collapse if any one of these analyses became imprecise.

Existing Effort. Reasoning container memory layouts is theoretically an undecidable problem [27]. Existing approaches mainly adopt two different strategies to achieve the overapproximation. One line of the techniques smashes a container and only reasons about the ownership without analyzing the indexes [28–32]. Although the analyses scale to large programs, the spurious value flows plague the analysis results such that 75.2% of the client analysis are false positives [33]. The other line of the techniques encodes the program values by logical formulae and applies strong updates by enforcing the container axioms [34, 35]. Despite the high precision, the exhaustive symbolic reasoning introduces a significant number of case analyses, causing the disjunctive explosion problem [36] and degrading the scalability significantly. For example, Compass only scales to 128 KLoC even when the solving procedure is optimized [35, 37].

Insight. Although analyzing generic containers is pie in the sky, there exists a particular class of containers, of which the memory layouts can be precisely tracked by deterministic indexes. As shown in Figure 1, for example, the modifications of the container objects always occur at the end or use constant keys, which can be discovered by tracking all possible modifications upon container objects. The stored objects can be identified by the deterministic indexes, which enables strong updates upon container memory layouts. Specifically, we formulate the idiom by the notion of *anchored containers*. A container is an anchored container at a program location if all the preceding modifications have deterministic indexes. Anchored containers widely exist in real-world programs, e.g., 75.6% Java EE containers in the top 10 cases searched on GitHub conform to the programming idiom.[1] They establish the "anchors" for memory objects, which can be used to identify precise value flows through containers.

Solution. We introduce the *memory orientation analysis* to identify anchored containers and compute their precise index-value correlations, which further support a fast and precise value-flow analysis. Specifically, we establish a combined abstract domain embodied with path constraints to

[1]The empirical data are listed online: https://containeranalyzer.github.io/empirical.pdf.

```
Queue<Pr> ps = new LinkedList<Pr>();
Pr p = Space.getProject(dir);
ps.offer(p);
ps.offer(new Proj(newDir, arg));
executeProject(ps.peek());
```

(a) Example code in Hibernate-ORM

```
HttpSession<String,Object> s = new HttpSession<>();
s.put(Support.FIND_BLOCK, Boolean.FALSE);
s.put(Support.FIND_WHAT, searchWord);
String word = (String) s.get(Support.FIND_WHAT);
addToHistory(new EditorFindSupport(block, word));
```

(b) Example code in NetBeans

```
Stack<State> s = new Stack<>();
s.addElement(otherStateElem);
s.push(new State(cursor));
State state = includeStack.pop();
```

(c) Example code in Struts

```
Dictionary<String, String> config = new Hashtable<>();
config.put(Factory.CLASS, Driver.getName());
config.put(Factory.NAME, "iotdb");
String name = config.get(Factory.NAME);
```

(d) Example code in IoTDB

Fig. 1. Examples of a programming idiom in Hibernate-ORM, NetBeans, Struts, and IoTDB.

```
1  void foo(String s) {
2    HttpSession hs;//o1
3    Map m = new HashMap<String,String>();//o2
4    hs.setAttribute("id", "a");
5    hs.setAttribute("age", null);
6    m.put("id", "b");
7    String i = hs.getAttribute("id");
8    if (c) {m.put(s, i);}
9    else {m.put(s, null);}
10   Stack<String> ids = new Stack<>();//o3
11   String j = m.get("id");
12   ids.add(i);
13   if (c) {ids.add(j);}
14   bar(hs, ids);
15 }
16 void bar(HttpSession hs, Stack ids) {
17   String p = hs.getAttribute("age");
18   String q = ids.peek();
19   String r = hs.getAttribute("id");
20   if (c)
21     out(p.length()+q.length()+r.length());
22 }
```

Fig. 2. A motivating program.[2]

track multi-domain properties precisely, such as points-to facts and deterministic indexes. At a high level, our approach works in two stages as follows:

- The memory orientation analysis identifies anchored containers and applies the strong updates to their memory layouts. A non-anchored container is smashed without analyzing its index-value correlation. Based on container memory layouts, the memory orientation analysis enables the construction of a precise value-flow graph. For example, the container objects o_1 and o_3 in Figure 2 are anchored containers. It index-value correlation implies that p is *null* and r is not *null* at line 21, and the analysis constructs the precise value-flow graph with the solid edges in Figure 3.
- We conduct a demand-driven reachability analysis to solve an instance of the value-flow problem. It collects the value-flow facts of interest when traversing the value-flow graph. The constraints are collected and solved to determine the reachability if necessary. For example, the null pointer exception (NPE) detector traverses the value-flow graph in Figure 3 from *null* values to dereferenced pointers, and reports an NPE with no false positive.

Note that it is non-trivial to identify and apply strong updates to anchored containers in the first stage, involving the accumulative effects of the modifications along control flow paths. For example, the container object o_2 are modified at lines 6, 8, and 9, and the last two modifications have non-deterministic indexes, so o_2 is not an anchored container after line 8. Particularly, we establish a subdomain to maintain the accumulative effects of modifications upon each container object, which explicitly indicates whether it is an anchored container. When transforming each subdomain simultaneously, we instantiate a semantic reduction operator [38] to track the interleaving among multiple subdomains and apply strong updates to anchored containers.

[2]The program is simplified from Hibernate-ORM, IoTDB, and Struts.