Dataflow-Related Execution Traces

Introduction

Many specification mining model inference techniques do not consider dataflow relationships between method calls. Unlike execution traces generated by unit tests, traces collected from a running application may have several instances of library classes running, and there may be interactions or exchange of data between these instances. To create higher quality execution traces, we propose to make use of information about the arguments and return values of methods calls to infer data-flow relationships between method calls. Using the data-flow relationship, we can filter out unrelated events that are not related to the other events in the trace.

Dataflow-Related Traces

Due to the heavier cost of static analysis, we use a cheap and simple heuristic to determine if 2 method calls, A and B, are related by data-flow. If the intersection of the parameters of A and the return value of B is non-empty, then we consider that A and B are related by data-flow.

Based on this heuristic, we determine if method calls belong to a trace based on whether or not it has any data-flow relationship with another method call in the same trace.

We instrumented method calls to 5 common Java libraries (LinkedList, HashMap, HashSet, LinkedHashMap, LinkedHashSet) called from NewPipe, an Android application. NewPipe is an alternative to the official YouTube Android client. The app was manually operated for a short period of time (<3 minutes). As the Java libraries instrumented were from the Java standard library, they are fairly low-level and are used very commonly, this short period of time was enough to produce a sufficient number of traces (over 15000 events). As such, automated testing tools were not used to produce traces for these libraries.

Class	Number of events
HashMap	13743
HashSet	836
LinkedList	418
LinkedHashMap	1238
LinkedHashSet	310

Table 1: Number of events collected

Class	Number of traces
HashMap	2113
HashSet	135
LinkedList	19
LinkedHashMap	47
LinkedHashSet	12

Table 2: Number of traces

Collected Data

We instrumented calls to the constructor and public methods on 5 Java library classes (LinkedList, HashMap, HashSet, LinkedHashMap, LinkedHashSet).

Table 1 shows the distribution of events collected for each library class. HashMap and HashSet are the most common data structures used among the classes we investigated. Due to its extremely frequent use, ArrayList was not instrumented as the cost of instrumentation caused the Android application to become too slow.

After grouping calls together based on the data-flow information, the number of traces can be seen in Table 2. We note that, in comparison to traces collected when running unit tests, these traces are collected real-world usage of these libraries. These includes long-running uses that can span the entire lifespan of the application. As such, each trace can contain a large number of events. In some cases, after instantiating, the data structure were not yet used during the duration of operating the app.

Inferred Automata

As a baseline, we also used a naive approach to group traces by considering subsequent calls following a constructor to be the part of the same trace. Using traces of HashSet to illustrate this, the automata constructed from this baseline approach is seen in Figure 1, and the automata constructed after taking into account data-flow information is seen in Figure 2.

Based on Figure 1 and Figure 2, the use of data-flow allows more successful

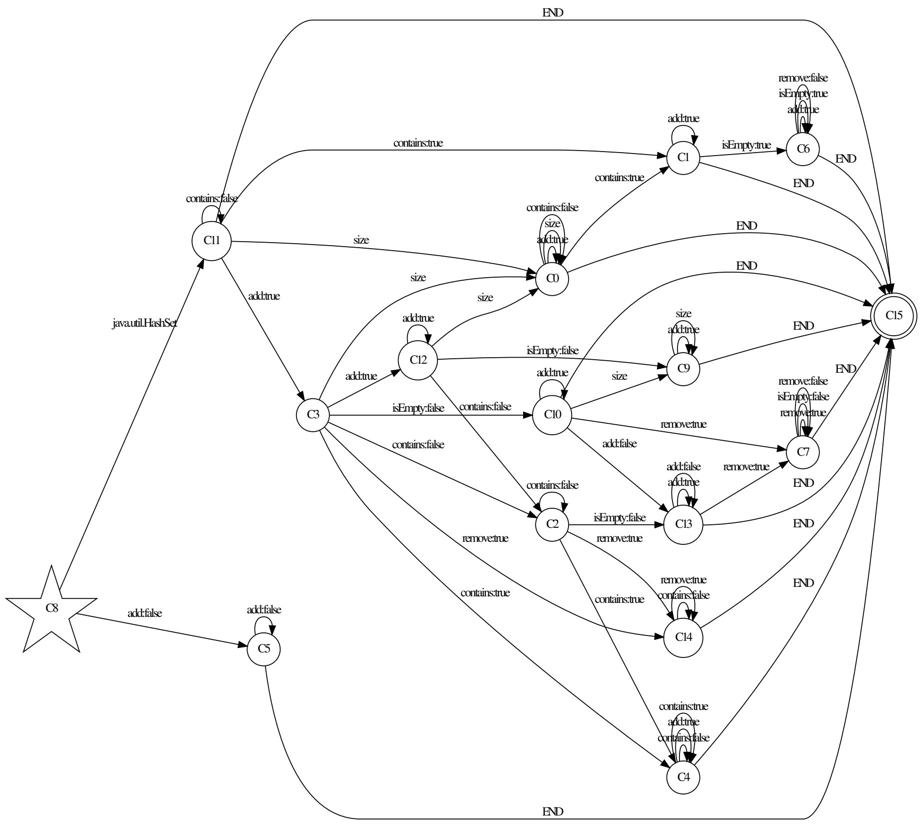
Figure 1: Automata inferred from traces without using data-flow information

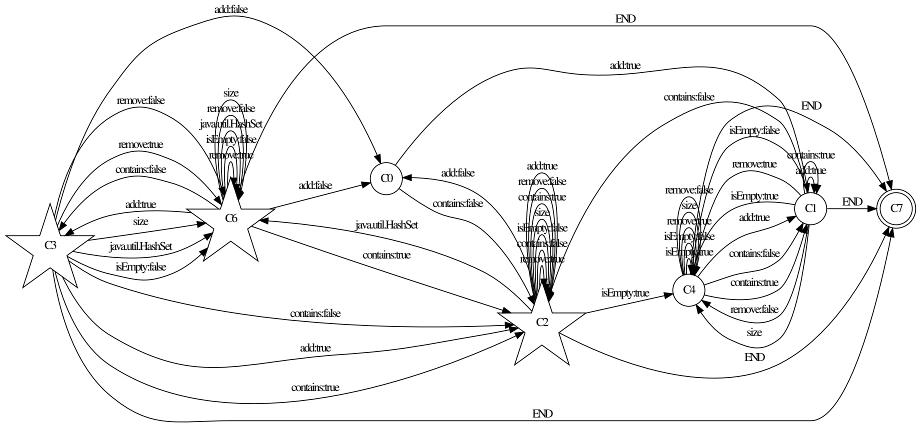
Figure 2: Automata inferred from traces grouped by data-flow

inference of the states that a HashSet can be in. The large number of nodes in Figure 1 suggests that without using data-flow information, the inference model did not succeed in merging identical states.