

Nondeterministic call graph - same method is associated with different identifiers in different runs.

Open

...

Hi, I have recently been using Soot for an empirical study to detect non-deterministic behaviors in static analyzers. The experiments resulted in discovering some nondeterministic analysis results across multiple runs under various configurations of Soot.

Describe the bug

There is a unique identifier associated with some methods, and this identifier sometimes differs across each iteration. For example:

on the target program [CATS-Microbenchmark/ClassLoading1](#).

The same method is associated with different identifiers in different runs:

```
<java.io.ObjectInputStream$checkArray__211: void <init>()>
<java.io.ObjectInputStream$checkArray__251: void <init>()>
```

In the first instance, the identifier associated with java.io.ObjectInputStream\$checkArray is 211, while in the second instance, it is 251.

We observe that this inconsistent identifier accounts for the majority of the non-determinisms we detected. However, we also observe that this inconsistency only appears in the large results. When the generated call graph is very small (several kbs), this inconsistency disappears.

Input file

one of the target programs that exhibit nondeterminism [CATS-Microbenchmark/ClassLoading1](#)

To reproduce

one of the configurations that exhibit nondeterminism:

```
-pp -w -p cg.spark on-fly-cg:false,enabled:true -p cg.spark enabled:true --
optimize -p cg.spark vta:true -p cg.spark rta:false -p cg.spark field-based:false
-p cg.spark merge-stringbuffer:false -p cg.spark string-constants:false -p
cg.spark simulate-natives:false -p cg.spark empties-as-allocs:false -p cg.spark
simple-edges-bidirectional:true -p cg.spark simplify-offline:true -p cg.spark
simplify-sccs:false -p cg.spark ignore-types-for-sccs:false -p cg.spark cs-
demand:true -p cg.spark lazy-pts:true -p cg safe-forname:true -p cg safe-
newinstance:false -p cg implicit-entry:true -p cg trim-clinit:false -p cg.spark
types-for-sites:true -p cg.spark set-impl:double -p cg.spark double-set-
old:sharedlist -p cg.spark double-set-new:sharedlist --jasmin-backend -p cg.cha
enabled:true -p cg types-for-invoke:true -p cg resolve-all-abstract-invokes:false
-p cg.spark pre-jimplify:false -p cg.spark propagator:merge
```

Could you kindly provide some insights on why this happens? Is this inconsistency expected? And why does it only occur when the generated call graph is really large?

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone


Development

No branches or pull requests

Notifications

Unsubscribe

You're receiving notifications because you're subscribed to this thread.

StevenArzt

Contributor

...

Are the identifiers really revelant? If the callgraph edges and points-to-sets are identical, I don't see that we must keep the IDs consistent. They only need to be consistent within a single run imho.

Author

...

Thank you for your swift response! We are currently conducting an empirical study to detect non-deterministic behaviors within static analyzers. We would greatly appreciate your insights regarding the generation of identifiers, such as at which phase these identifiers are generated? Additionally, we are curious as to why these identifiers occasionally exhibit differences. We observed that in large call graphs, these identifiers consistently vary, while in small results, they remain consistent. Your assistance in shedding light on this matter would be greatly valuable to our study. [@StevenArzt](#)