Detected non-deterministic behaviors when using different algorithms to generate call graphs

Hello OPAL team,

I have been using OPAL for an empirical study aimed at detecting non-deterministic behaviors in static analyzers. During the experiments, we observed some non-deterministic behaviors across multiple runs when using different algorithms to generate call graphs.

Experimental Setup:

- Benchmark: Dacapo-2006
- Call Graph Generation Algorithms: CHA, RTA, XTA, and Points-to
- Timeout: 30 minutes per program
- Runs: 5 times per program-algorithm combination
- Environment: Docker containers on a server with 376GB of RAM and 2 Intel Xeon Gold 5218 16-core CPUs @ 2.30GHz running Ubuntu 18.04

Observations:

We detected non-deterministic results in one program: eclipse.jar. This non-determinism occurred with all four algorithms. Specifically, we found that certain edges in the call graphs were inconsistent across runs.

Example:

Algorithm: RTA
Call Graphs from Two Runs:
Call Graph 1
Call Graph 2

One observation is that when the below edge appears, several other edges will be missing in the same run:

```
"method": {
    "name": "initializeContextFinder",
    "declaringClass": "Lorg/eclipse/core/runtime/adaptor/EclipseStarter;"
    "returnType": "V",
    "parameterTypes": []
```

```
"targets": [
    {
        "name": "getContextClassLoader",
        "declaringClass": "Ljava/lang/Thread;",
        "returnType": "Ljava/lang/ClassLoader;",
        "parameterTypes": []
    }
```

Additional Data:

The attached xlsx file provides more details on these observations. It shows that when the red edge appears in one run, the black edges are missing in that run:
OPAL_diff.xlsx.

Could you please provide any insights into the possible causes of this non-deterministic behavior? Any feedback would be greatly appreciated!

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
No branches or pull reque

Notifications
Un
You're receiving notificatio thread.

Collaborator

thank your for reporting this. We're investigating the issue, but were yet unable to reproduce it. Can you give us more information on your setup, including which version of OPAL you are using (master or develop, commit hash if necessary), the JDK version and whether/which parts of the JDK you analyze alongside eclipse.jar, how you compute the call graphs (which tool are you using, what are your command line arguments) and how you compared them?

I just fixed an incorrectly implemented double-checked locking pattern (#200) that could lead to duplicate VirtualDeclaredMethods being created, but that doesn't seem to be the root cause of your reported non-determinism. Nonetheless, you could check whether you can still reproduce it using the up-to-date develop branch.

Author

Hi,

Thank you **@errt** so much for the prompt response! I'm happy to provide more details about my setup:

OPAL Version: de.opal-project version 5.0.0 from the Maven Central Repository
JDK Version: 11.0.19
I've developed an interface to invoke the OPAL API and print out the call graph using `CallGraphSerializer.writeCG()`. You can find the interface code at this GitHub repository.

For comparison, we converted each call graph into a set of edges (caller->callee) and compared the sets. The OPAL_diff.xlsx file shows the edge differences detected in the call graphs generated using different algorithms.

Regarding the OPAL version, I used OPAL as a Maven dependency, and it seems that version 5.0.0 is the most up-to-date version available in the Maven repository.

Collaborator

Thanks, with this I was able to reproduce the issue also on the current develop branch. I'm currently in the process of localizing and fixing it. It seems to stem from the reflection analysis.

added the bug label last week

self-assigned this last week

removed the bug label last week

Collaborator

It turns out this is actually expected behavior, and there is even a warning issued about this:

```
[warn][project configuration] java.lang.Thread is defined by multiple class
[warn][project configuration]   jar:file:[...]/eclipse.jar!/jar:data/eclips
[warn][project configuration]   jar:file:[...]/eclipse.jar!/jar:data/eclipse.zip!
[warn][project configuration]   keeping the first one.
```

(and several more like these)
The size of the call graph then depends on which class file is selected to be kept, i.e., which jar file is parsed first.
I don't think we will do anything about this. Having multiple conflicting class files on your class path is generally not a good idea.