Non-deterministic results - certain groups of nodes always missing [redacted]

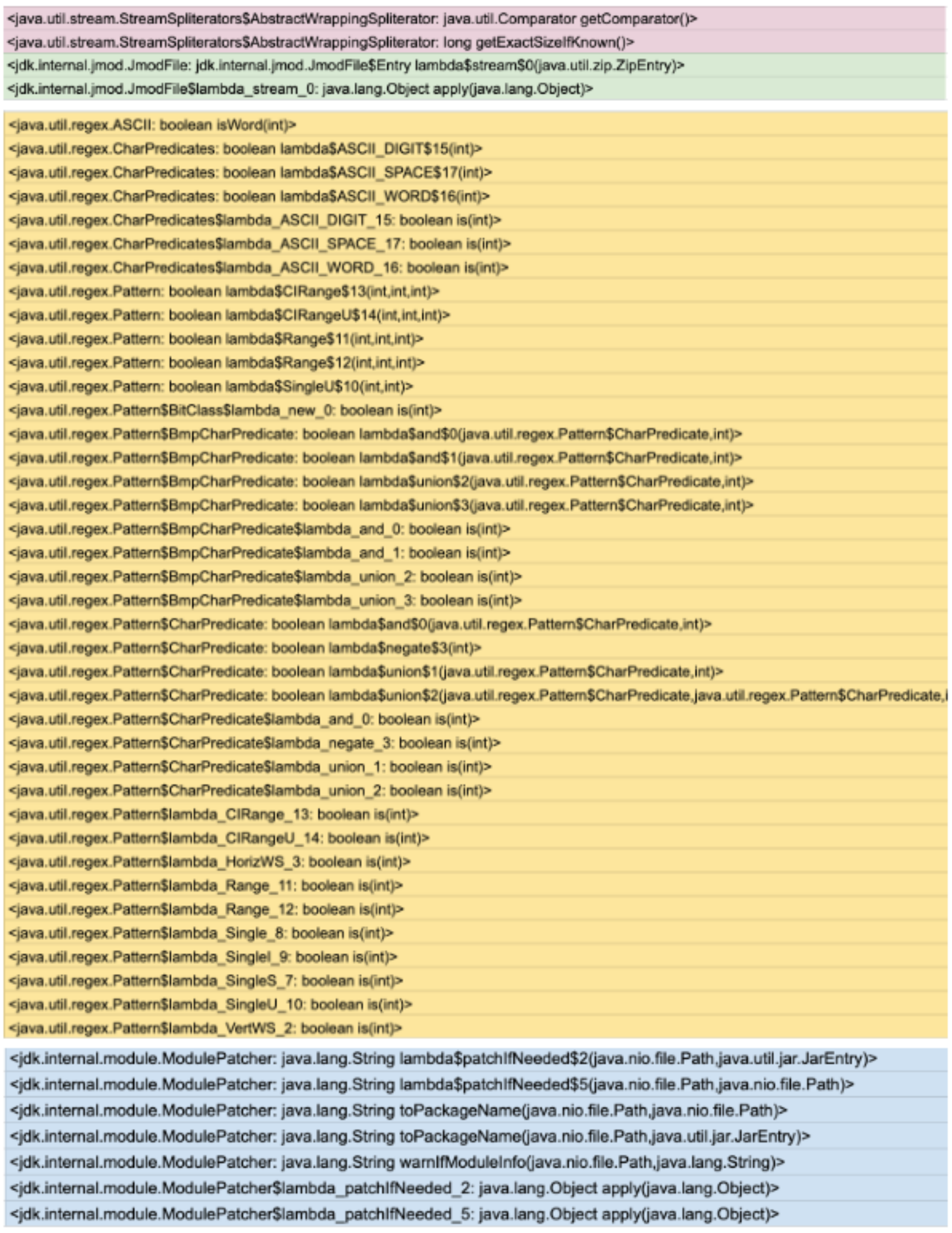
Open [redacted]

[redacted]

Hi,

I was running soot on 30 sampled configurations * 112 target programs to detect nondeterministic behaviors. I ran each configuration-program combination 5 times and compared the 5 results to detect inconsistent results. All 112 target programs come from the CATS-Microbenchmark. I observed that certain groups of nodes are always missing together.

Below are some examples of these missing groups of nodes.



```
<java.util.stream.StreamSpliterators$AbstractWrappingSpliterator: java.util.Comparator getComparator()>
<java.util.stream.StreamSpliterators$AbstractWrappingSpliterator: long getExactSizeIfKnown()>
<jdk.internal.jmod.JmodFile: jdk.internal.jmod.JmodFile$Entry lambda$stream$0(java.util.zip.ZipEntry)>
<jdk.internal.jmod.JmodFile$lambda_stream_0: java.lang.Object apply(java.lang.Object)>

<java.util.regex.ASCII: boolean isWord(int)>
<java.util.regex.CharPredicates: boolean lambda$ASCII_DIGIT$15(int)>
<java.util.regex.CharPredicates: boolean lambda$ASCII_SPACE$17(int)>
<java.util.regex.CharPredicates: boolean lambda$ASCII_WORD$16(int)>
<java.util.regex.CharPredicates$lambda_ASCII_DIGIT_15: boolean is(int)>
<java.util.regex.CharPredicates$lambda_ASCII_SPACE_17: boolean is(int)>
<java.util.regex.CharPredicates$lambda_ASCII_WORD_16: boolean is(int)>
<java.util.regex.Pattern: boolean lambda$CIRange$13(int,int,int)>
<java.util.regex.Pattern: boolean lambda$CIRangeU$14(int,int,int)>
<java.util.regex.Pattern: boolean lambda$Range$11(int,int,int)>
<java.util.regex.Pattern: boolean lambda$Range$12(int,int,int)>
<java.util.regex.Pattern: boolean lambda$SingleU$10(int,int)>
<java.util.regex.Pattern$BitClass$lambda_new_0: boolean is(int)>
<java.util.regex.Pattern$BmpCharPredicate: boolean lambda$and$0(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$BmpCharPredicate: boolean lambda$and$1(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$BmpCharPredicate: boolean lambda$union$2(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$BmpCharPredicate: boolean lambda$union$3(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$BmpCharPredicate$lambda_and_0: boolean is(int)>
<java.util.regex.Pattern$BmpCharPredicate$lambda_and_1: boolean is(int)>
<java.util.regex.Pattern$BmpCharPredicate$lambda_union_2: boolean is(int)>
<java.util.regex.Pattern$BmpCharPredicate$lambda_union_3: boolean is(int)>
<java.util.regex.Pattern$CharPredicate: boolean lambda$and$0(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$CharPredicate: boolean lambda$negate$3(int)>
<java.util.regex.Pattern$CharPredicate: boolean lambda$union$1(java.util.regex.Pattern$CharPredicate,int)>
<java.util.regex.Pattern$CharPredicate: boolean lambda$union$2(java.util.regex.Pattern$CharPredicate,java.util.regex.Pattern$CharPredicate,i
<java.util.regex.Pattern$CharPredicate$lambda_and_0: boolean is(int)>
<java.util.regex.Pattern$CharPredicate$lambda_negate_3: boolean is(int)>
<java.util.regex.Pattern$CharPredicate$lambda_union_1: boolean is(int)>
<java.util.regex.Pattern$CharPredicate$lambda_union_2: boolean is(int)>
<java.util.regex.Pattern$lambda_CIRange_13: boolean is(int)>
<java.util.regex.Pattern$lambda_CIRangeU_14: boolean is(int)>
<java.util.regex.Pattern$lambda_HorizWS_3: boolean is(int)>
<java.util.regex.Pattern$lambda_Range_11: boolean is(int)>
<java.util.regex.Pattern$lambda_Range_12: boolean is(int)>
<java.util.regex.Pattern$lambda_Single_8: boolean is(int)>
<java.util.regex.Pattern$lambda_SingleI_9: boolean is(int)>
<java.util.regex.Pattern$lambda_SingleS_7: boolean is(int)>
<java.util.regex.Pattern$lambda_SingleU_10: boolean is(int)>
<java.util.regex.Pattern$lambda_VertWS_2: boolean is(int)>

<jdk.internal.module.ModulePatcher: java.lang.String lambda$patchIfNeeded$2(java.nio.file.Path,java.util.jar.JarEntry)>
<jdk.internal.module.ModulePatcher: java.lang.String lambda$patchIfNeeded$5(java.nio.file.Path,java.nio.file.Path)>
<jdk.internal.module.ModulePatcher: java.lang.String toPackageName(java.nio.file.Path,java.nio.file.Path)>
<jdk.internal.module.ModulePatcher: java.lang.String toPackageName(java.nio.file.Path,java.util.jar.JarEntry)>
<jdk.internal.module.ModulePatcher: java.lang.String warnIfModuleInfo(java.nio.file.Path,java.lang.String)>
<jdk.internal.module.ModulePatcher$lambda_patchIfNeeded_2: java.lang.Object apply(java.lang.Object)>
<jdk.internal.module.ModulePatcher$lambda_patchIfNeeded_5: java.lang.Object apply(java.lang.Object)>
```

These groups of nodes are missing on many programs and configurations. I didn't observe any strong patterns they shared in common.

These call graphs are printed out by adding a SceneTransformer in "wjtp". Here is the interface SootInterface I used to invoke soot and print out the call graphs.

To reproduce

A minimum set of options that can reproduce this problem is as follows:
`-pp -w -p cg.spark on-fly-cg:false,enabled:true -p cg.spark enabled:true -p cg.spark pre-jimplify:true`

Input file

And a target program that can be used to reproduce this problem is CL1.

Could you offer some insights regarding the inconsistency in results?

Thank you in advance for any feedback!

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

Notifications

Unsubsc

You're receiving notifications be
thread.

[redacted]

[redacted] Author

Hello Soot team, I'm following up on this issue and would greatly appreciate any insights you could provide. Thank you!

[redacted] Author

Hi team, any feedback would be really helpful! Thank you so much in advance!

[redacted] Contributor

Sorry for the late reply, we're all quite busy at the moment.

Can you elaborate a bit on the problem? If I understand you correctly, you are using the SPARK callgraph algorithm (albeit with a weird command line - why is SPARK enabled twice?). There are some fairly non-standard options with on-the-fly CG being disabled and an explicit setting for pre-jimplification. I wonder whether you observe the same behavior when enabling SPARK but otherwise sticking to the default options.

Further, I'm not sure I understand the issue. You write that some nodes are "always missing together". Does this mean that these nodes are always missing? Or does it mean that sometimes all nodes of a group are there and sometimes all nodes in the group are missing? That would be the difference between inconsistent behavior and simply an incomplete CG.

[redacted] Author

Hi Steven, thank you so much for your time and valuable feedback!

We've conducted testing on 30 sampling configurations, encompassing 28 options, which were generated using the 2-way covering array methodology. Consequently, certain option combinations may appear somewhat non-standard. This non-deterministic behavior has been observed in a total of 20 configurations.

Because the original configurations we generated are quite extensive, each comprising 28 options. The configuration I used in this particular example was derived from an original configuration through bisection. Despite this refinement process, it still produces inconsistent results.

> I wonder whether you observe the same behavior when enabling SPARK but otherwise sticking to the default options.

I have tried running the same test case while enabling SPARK, not disabling on-the-fly CG, but keeping all other settings at their default values. And the results I obtained with this configuration are deterministic.

> Further, I'm not sure I understand the issue. You write that some nodes are "always missing together". Does this mean that these nodes are always missing? Or does it mean that sometimes all nodes of a group are there and sometimes all nodes in the group are missing?

My apologies for any confusion. The statement means that all nodes of a group sometimes are there and sometimes are missing, within a particular group, all nodes occasionally appear/disappear together.

> That would be the difference between inconsistent behavior and simply an incomplete CG.

The analysis that led to these inconsistent results all finished successfully. We have excluded the timed-out runs.

@StevenArzt