TABLE I: Distinct common patterns of nondeterminism.

Pattern ID: **CO-1** (See Figure 1a for an example)
Description: If one thread modifies a mutable collection while another thread is iterating over it, nondeterministic behavior can occur.
Solution: Avoid iterating directly over collections; use a snapshot iterator instead.
Occurrence in each tool: Soot (10), DroidSafe (1).
Related results: Soot-9 [1], 10 [2], [3], 11 [4], [5], 14 [6], 20 [7], 21 [8], 24 [9], 26 [10], 27 [11], 28 [12], DroidSafe-1 [13].

Pattern ID: **CO-2** (See Figure 1b for an example)
Description: Using data structures that are not thread-safe can result in exceptions due to concurrent access.
Solution: Avoid using thread-unsafe data structures; instead, use their thread-safe counterparts or other data structures that guarantee thread safety.
Occurrence in each tool: Soot (5), FlowDroid (1).
Related results: Soot-1 [14], 15 [15], 22 [16], 23 [17], 29 [18], FlowDroid-3 [19].

Pattern ID: **CO-3** (See Figure 1c for an example)
Description: Static analyzers sometimes use an intermediate data file for procedure attributes, but multi-threaded execution can make its presence nondeterministic. When this file is inaccessible, analyzers resort to alternative sources, introducing nondeterminism into the analysis results.
Solution: Avoid using multiple sources of intermediate data; rely on a single, deterministic source only.
Occurrence in each tool: Infer (12).
Related results: Infer-5 [20], 15 [21], 17 [22], 19 [23], 20 [24], 22 [25], 24 [26], 27 [27], 28 [28], 30 [29], 32 [30], 33 [31].

Pattern ID: **AL-1** (See Figure 1d for an example)
Description: Intermediate and final results can depend on the order of class and file processing. Nondeterministic behavior can occur if this order changes.
Solution: Induce a consistent order on classes and files (e.g., always iterate through user-provided classes first when searching for a main method) (Soot-19 [32]).
Occurrence in each tool: Infer (2), Soot (1).
Related results: Infer-13 [33], 29 [34], Soot-19 [32].

Pattern ID: **AL-2** (See Figure 1e for an example)
Description: Nondeterminism occurs due to the handling of mutually recursive functions. When the analysis starts with a procedure and follows the call chain back to the starting procedure, it breaks the cycle by returning an empty summary. Starting the analysis with a different procedure each time causes different procedures returning empty summaries, leading to nondeterministic results.
Solution: Necessary steps are needed to avoid the nondeterminism introduced by the mutually recursive functions. E.g., a possible mitigation strategy is to ignore class initializers in Java (Infer-16 [35]).
Occurrence in each tool: Infer (7).
Related results: Infer-1 [36], 4 [37], 16 [35], 21 [38], 25 [39], 34 [40], 35 [41].

Pattern ID: **AL-3** (See Figure 1f for an example)
Description: Nondeterminism arises from the modular, collaborative architecture of the analysis framework, where multiple interdependent analyses run in parallel and share results via central storage. If dependencies aren't satisfied initially, a continuation function handles them, reporting updates to central storage. This mechanism can cause nondeterminism due to inconsistencies between original and continuation handling. Variations in the ordering and timing of analyses can lead to nondeterministic outcomes.
Solution: Ensure that continuation and initial handling share identical logic. E.g., extracting the handling processes into a common method to avoid inconsistencies (OPAL-2 [42], [43], [44]).
Occurrence in each tool: OPAL (3).
Related results: OPAL-2 [42], [43], [44], 3 [45], 8 [46].

Pattern ID: **ODS-1** (See Figure 1g for an example)
Description: Nondeterminstic behavior occurs when an assumption is made about the iteration order of a data structure (e.g., that it will be the same as the insertion order) that does not hold.
Solution: Avoid using unordered data structures; instead, use their ordered counterparts or other data structures that guarantee iteration order.
Occurrence in each tool: Soot (6), WALA (4), FlowDroid (1), Infer (1).
Related results: Soot-2 [47], 7 [48], 12 [49], 16 [50], 17 [51], 18 [52], FlowDroid-2 [53], WALA-3 [54], 4 [55], 5 [56], 7 [57], Infer-31 [58].

Pattern ID: **SRV-1** (See Figure 1h for an example)
Description: Nondeterministic behavior occurs due to the usage of system-dependent or random values when naming or identifying intermediate files or variables (e.g., `System.identityHashCode`). This does not necessarily make the analysis result incorrect, but it makes comparing the results of multiple runs difficult.
Solution: Avoid using system-dependent or random values in this use case, instead defining deterministic naming or identification schemes.
Occurrence in each tool: WALA (2), Soot (1), Infer (1).
Related results: WALA-2 [59], [60], [61], 6 [62], [63], [64], Soot-6 [65], Infer-11 [66].

Pattern ID: **ET-1** (See Figure 1i for an example)
Description: Nondeterministic behavior arises from using timeouts for internal analysis, leading to flaky and incomplete results.
Solution: Avoid using timeouts for analysis, instead utilizing iteration limits to stop an analysis early.
Occurrence in each tool: WALA (1), Infer (1).
Related results: WALA-1 [67], Infer-3 [36].

```
// the following is a snapshot iterator; this is
     necessary because it can happen that phantom
     methods are added during resolution.
- Iterator<SootMethod> methodIt =
    cl.getMethods().iterator();
+ Iterator<SootMethod> methodIt = new
    ArrayList<SootMethod>(cl.getMethods().iterator());
```

(a) Soot-9 (CO-1) [1]

```
+ import java.util.concurrent.ConcurrentHashMap;
+ import java.util.concurrent.CopyOnWriteArrayList;
- public static Map method2Locals2REALTypes =
    new HashMap();
  ...
+ public static Map method2Locals2REALTypes = new
    ConcurrentHashMap();
  ...
- static List<Transformer> jbcotransforms = new
    ArrayList<Transformer>();
+ static List<Transformer> jbcotransforms = new
    CopyOnWriteArrayList<>();
```

(b) Soot-15 (CO-2) [15]

```
+ (*AnalysisCallbacks.proc_resolve_attributes
+ introduces non-determinism because it first looks
+ for the attributes in a summary and if it doesn't
+ find it then returns the value in the attribute
+ column from the capture DB. The plan is to eliminate
+ these calls one by one, replacing with Attributes.load
+ which only looks at the capture DB attribute column.*)
+ let get_formals pname =
+ AnalysisCallbacks.proc_resolve_attributes pname |>
+ Option.map  f:ProcAttributes.get_formals
+ let get_formals pname =
+ Attributes.load pname |>
+ Option.map  f:ProcAttributes.get_formals
// Some analysis logic ...
```

(c) Infer-5 (CO-3) [20]

```
+ // try to infer a main class from the command line if
+ // none is given.
+ for (Iterator<SootClass> classIter =
+ Options.v().classes().iterator();
+ classIter.hasNext();) {
  + SootClass c = (SootClass) classIter.next();
  + if (c.declaresMethod ("main",
  + new SingletonList(ArrayType.v(RefType.v("java.lang.
  + String"), 1)),VoidType.v())) {
    + G.v().out.println("No main class given. Inferred '"

    + +c.getName()+"' as main class.");
    + setMainClass(c);
    + return;
  + }
+ }
```

(d) Soot-19 (AL-1) [32]

Fig. 1: Illustrative examples of common patterns.

```
+ let pname = Procdesc.get_proc_name proc_desc in
+ if Procname.is_java_class_initializer pname then
+ (*Mitigation: We ignore Java class initializer
+ to avoid non-deterministic FP.*)
  + None
+ else
// Some analysis logic ...
```

(e) Infer-16 (AL-2) [35]

```
- propertyStore(objectType, TypeImmutability.key) match {
- // Some analysis logic ...}
+ checkTypeImmutability(propertyStore(objectType,
                          + TypeImmutability.key)) ...
+ // Extract a common method used in both original
+ // and continuation handling.
+ def checkTypeImmutability(result: EOptionP[FieldType,
+ TypeImmutability]): Unit = result match {
+ // Some analysis logic ...}
```

(f) OPAL-2 (AL-3) [42], [43], [44]

```
- Set<SootClass> res = new HashSet<>();
+ // This has to be a list such that an iterator
+ // walks the hierarchy upwards. Otherwise,
+ // iterating on it might return an interface
+ // with more callees than the nearest
+ // superclass leading to more definitions.
+ ArrayList<SootClass> res = new ArrayList<>();
```

(g) FlowDroid-2 (ODS-1) [53]

```
Jimple jimple = Jimple.v();
- Local vnew = jimple.newLocal("tmp", useType);
- vnew.setName("tmp$" + System.identityHashCode(vnew));
+ Local vnew = localGenerator.generateLocal(useType);
```

(h) Soot-6 (SRV-1) [65]

```
+ // Call graph construction is terminated after
+ // MAX_ITERATIONS runs of the outer fixed point
+ // loop of call graph construction.
+ public void worked(int units) {
  + noteElapsedTime();
  + iterations++;
  + if (iterations >= MAX_ITERATIONS) {
    + throw CancelRuntimeException.make
    + ("should have run long enough");}}
```

(i) WALA-1 (ET-1) [67]

References

[1] mbenz89, "Fixing a (possible) concurrentmodificationexception #1016," https://github.com/soot-oss/soot/pull/1016, Sep 2018.

[2] muthu30011997, "Concurrentmodificationexception when using ... #1199," https://github.com/soot-oss/soot/issues/1199, August 2019.

[3] michael emmi, "Avoid modification exception in line number adder #1886," https://github.com/soot-oss/soot/pull/1886, July 2022.

[4] Zongyin-Hao, "[java.util.concurrentmodificationexception]caused by collections.synchronizedlist in sootclass #1811," https://github.com/soot-oss/soot/issues/1811, December 2021.

[5] CirQ, "Returning view of method list to avoid cme #1953," https://github.com/soot-oss/soot/pull/1953, February 2023.

[6] michael emmi, "Avoid concurrent modification exception #1980," https://github.com/soot-oss/soot/pull/1980, June 2023.

[7] StevenArzt, "077b50c," https://github.com/soot-oss/soot/commit/077b50c, August 2016.

[8] P. Burchard, "Fixes concurrent modification exception," https://github.com/soot-oss/soot/commit/833c428, March 2023.

[9] ericbodden, "bugfix against concurrent modification exception," https://github.com/soot-oss/soot/commit/d2e2d3c, September 2012.

[10] StevenArzt, "db3072e," https://github.com/soot-oss/soot/commit/db3072e, October 2015.

[11] ——, "fe424e3," https://github.com/soot-oss/soot/commit/fe424e3, August 2015.

[12] ——, "1ad7449," https://github.com/soot-oss/soot/commit/1ad7449, June 2023.

[13] mgordon, "a885626," https://github.com/MIT-PAC/droidsafe-src/commit/a885626, January 2015, do not follow exceptional paths and fix conurrent modification exception by using exceptional graph.

[14] mbenz89, "fbe472a," https://github.com/soot-oss/soot/commit/fbe472a, August 2019.

[15] gnom7, "1661ea8," https://github.com/soot-oss/soot/pull/807/commits/1661ea8, November 2017.

[16] MarcMil, "Replace with concurrent hash set," https://github.com/soot-oss/soot/commit/44e65f0, March 2021.

[17] StevenArzt, "fixed a strange concurrency issue," https://github.com/soot-oss/soot/commit/f679df4, July 2017.

[18] ——, "Replace with concurrent hash set," https://github.com/soot-oss/soot/commit/a44516c, March 2021.

[19] ——, "a5c9e34," https://github.com/secure-software-engineering/FlowDroid/commit/a5c9e34, May 2022.

[20] N. Gorogiannis, "1afd05a," https://github.com/facebook/infer/commit/1afd05a, Jul 2021.

[21] ——, "41c4021," https://github.com/facebook/infer/commit/41c4021, Jul 2021.

[22] ——, "567fa3f," https://github.com/facebook/infer/commit/567fa3f, Jul 2021.

[23] ——, "66d9ce1," https://github.com/facebook/infer/commit/66d9ce1, July 2021.

[24] ——, "6a6d888," https://github.com/facebook/infer/commit/6a6d888, Jul 2021.

[25] ——, "89ba2cc," https://github.com/facebook/infer/commit/89ba2cc, Jul 2021.

[26] ——, "a138156," https://github.com/facebook/infer/commit/a138156, Jul 2021.

[27] ——, "aa85648," https://github.com/facebook/infer/commit/aa85648, Jul 2021.

[28] ——, "ac158b2," https://github.com/facebook/infer/commit/ac158b2, Jul 2021.

[29] ——, "c10ddea," https://github.com/facebook/infer/commit/c10ddea, Jul 2021.

[30] ——, "e00fe73," https://github.com/facebook/infer/commit/e00fe73, Jul 2021.

[31] ——, "eb1c95a," https://github.com/facebook/infer/commit/eb1c95a, Jul 2021.

[32] ericbodden, "f78cff7," https://github.com/soot-oss/soot/commit/f78cff7, January 2010.

[33] C. Calcagno, "346d3e6," https://github.com/facebook/infer/commit/346d3e6, Sep 2016.

[34] A. Hajdu, "afedeb6," https://github.com/facebook/infer/commit/afedeb6, Dec 2021.

[35] skcho, "4c3679c," https://github.com/facebook/infer/commit/4c3679c, Sep 2021.

[36] J. Villard, "Infer is not deterministic," https://github.com/facebook/infer/issues/1110, June 2019.

[37] skcho, "0b5376a," https://github.com/facebook/infer/commit/0b5376a, Feb 2024.

[38] J. Villard, "88e6363," https://github.com/facebook/infer/commit/88e6363, Jun 2023.

[39] N. Gorogiannis, "a477d06," https://github.com/facebook/infer/commit/a477d06, Jul 2022.

[40] skcho, "f26573c," https://github.com/facebook/infer/commit/f26573c, May 2021.

[41] J. Villard, "fcd6e14," https://github.com/facebook/infer/commit/fcd6e14, May 2023.

[42] roterEmil, "fixed non determinism issue in field immutability," https://github.com/opalj/opal/pull/100, Jan 2023.

[43] ——, "1354b63," https://github.com/opalj/opal/commit/1354b63, Jan 2023.

[44] ——, "ffc7005," https://github.com/opalj/opal/commit/ffc7005, Jan 2023.

[45] ——, "4e0fb9c," https://github.com/opalj/opal/commit/4e0fb9c, Sep 2022.

[46] F. Kübler, "55cdd02," https://github.com/opalj/opal/commit/55cdd02, Jul 2019.

[47] Auzel, "fixed flaky tests #1802," https://github.com/soot-oss/soot/pull/1802/files, November 2021.

[48] Lex999, "Order of phi statements made deterministic. #1065," https://github.com/soot-oss/soot/pull/1065, November 2018.

[49] michael emmi, "Use linked variants of hash sets and maps. #1309," https://github.com/soot-oss/soot/pull/1309, March 2020.

[50] R. Vallee-Rai, "Got rid of the annoying non-determinism." https://github.com/soot-oss/soot/pull/807/commits/d436b7a, March 1999.

[51] olhotak, "2c88b38," https://github.com/soot-oss/soot/commit/2c88b38, January 2005.

[52] ericbodden, "44c81a4," https://github.com/soot-oss/soot/commit/44c81a4, August 2012.

[53] timll, "a420a71," https://github.com/secure-software-engineering/FlowDroid/commit/a420a71, March 2023.

[54] msridhar, "sort cnf clauses before simplification for greater determinism," https://github.com/wala/WALA/commit/a04a8d6, February 2008.

[55] ——, "Change bitvectorrepository to use linkedlists," https://github.com/drzhonghao/walaforgrapa/commit/7579f56, February 2013.

[56] sjfink, "2ee8657," https://github.com/drzhonghao/walaforgrapa/commit/2ee8657, September 2008.

[57] ——, "f9aa9e6," https://github.com/wala/WALA/commit/f9aa9e6, March 2009.

[58] S. Blackshear, "fd10580," https://github.com/facebook/infer/commit/fd10580, Sep 2017.

[59] mherzberg, "Different call graphs for apk on subsequent runs #88," https://github.com/wala/WALA/pull/88, August 2015.

[60] mohrm, "using hashcodecomparator in class loading may and does cause instabilities #133," https://github.com/wala/WALA/pull/133, January 2017.

[61] ——, "get rid of hashcodecomparator and all its usages #134," https://github.com/wala/WALA/pull/134, January 2017.

[62] sjfink, "f0a64f7," https://github.com/wala/WALA/commit/f0a64f7, May 2008.

[63] ——, "9956ead," https://github.com/wala/WALA/commit/9956ead, May 2008.

[64] ——, "f1ddb98," https://github.com/wala/WALA/commit/f1ddb98, May 2008.

[65] michael emmi, "Deterministic generation of local variable names. #1281," https://github.com/soot-oss/soot/pull/1281, January 2020.

[66] J. Villard, "24e6ae9," https://github.com/facebook/infer/commit/24e6ae9, Dec 2016.

[67] msridhar, "Limit how long kawa call graph construction runs #904," https://github.com/wala/WALA/pull/904, July 2021.