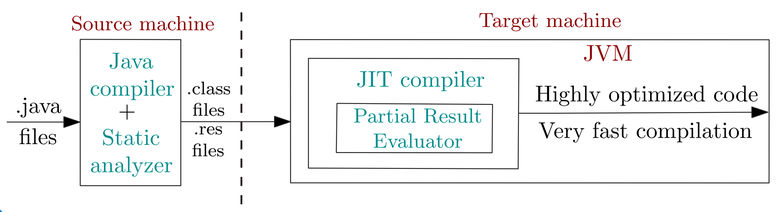
A Study of the Impact of Callbacks in Staged Static+Dynamic Partial Analysis

Aditya Anand†

Indian Institute of Technology Mandi

Staged Analysis

- Java programs are compiled statically as well as just-in-time (JIT).
- JIT-compilation time also affects the execution-time of programs.
- Typical JIT compilers perform imprecise analyses (e.g., intraprocedural) and sacrifice precision.
- Recent approaches [1, 2] perform partial analysis of available program statically, and record the dependencies on the unavailable portions as *conditional values*.



Impact of Callbacks

```
                Library-Overridden Method
1  class A {
2    void foo(X p) {
3      Y m = new Y(); // O3
4      p.f = m;
5      this.bar(m);
6    }
7    void bar(Y q) {
8      Z r = new Z(); // O8
9      q.g = r;
10   }
11 } /* class A */
```

- The statically unknown argument passed to foo from a library method may affect:
  - object O_3 in foo.
  - object O_8 in bar.

Study Setup

1. Static Analysis (in Soot):

- Extend an existing tool [3] that generates dependencies for each program element.
- Identify the dependencies that may get affected by possible callbacks from libraries.

2. Benchmarks:

- 10 benchmarks from the DaCapo 9.12 suite using the "default" input size.

References

[1] Manas Thakur and V. Krishna Nandivada. PYE: A Framework for Precise-Yet-Efficient Just-In-Time Analyses for Java Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS), July 2019.*

[2] Aditya Anand and Manas Thakur. 2022. Principles of Staged Static+Dynamic Partial Analysis. *In Proceedings of the 29th Static Analysis Symposium (SAS 2022). Springer International Publishing, 30 pages.*

[3] Nikhil T R, Dheeraj Yadav, and Manas Thakur. 2021. Stava. [https://github.com/CompL-IITMandi/stava].

Generation of Conditional Values

```
1  class X {
2      Y f;
3  }
4  class Y {
5      Z g;
6  }
```

```
1  class A {
2    void foo(X p) {
3      Y m = new Y(); // O3
4      p.f = m;
5      this.bar(m);
6    }
```

```
7    void bar(Y q) {
8      Z r = new Z(); // O8
9      q.g = r;
10   }
11 } /* class A */
```

- Conditional Value for Object O_3: {⟨caller,⟨argument,1⟩, ⟨A.bar,⟨parameter,1⟩⟩}
- Conditional Value for Object O_8: {⟨caller,⟨argument,1⟩⟩}

Finding Callback-Affected Methods/Objects

```
1  Procedure callbackObjectsAndMethods()
2      foreach application class c do
3          if c overrides or implements a library method m then
4              Mark m and all its parameters as callback_affected.
5      repeat
6          foreach callback_affected method mtd do
7              foreach object obj ∈ mtd do
8                  if obj has dependency on a callback_affected formal parameter then
9                      Mark obj as a callback_affected object.
10         foreach callback_affected object obj do
11             if obj is passed as the kth argument to another method n then
12                 Mark n and its kth parameter as callback_affected.
13     until fixpoint;
```

1. Identify library-overridden and implemented methods in application classes, and mark all their parameters as potentially callback-affected.
2. For all the callback-affected methods, if any of their local objects depend on the existing callback-affected methods and objects, mark those also as callback-affected.
3. Finally, if any of the callback-affected objects is passed to another method, say as the k^{th} argument, mark the callee as well as the corresponding parameter as callback-affected.
4. Perform the previous two operations till a fixed point.

Evaluation Results

Benchmark	Callback methods	Total methods	Callback objects	Total objects
avrora	43	3181	147	13344
batik	896	6934	1865	34137
lusearch	123	1971	338	9054
luindex	69	1998	171	10260
pmd	595	5941	1301	30016
sunflow	49	2428	104	14136
h2	639	4777	1315	27610
xalan	821	6396	2131	31488
fop	968	11470	2387	74590
eclipse	1515	29419	3505	79443

- On average, 7.7% application methods are identified as callback-affected (either directly or through parameters).
- 4.1% application objects contain dependencies related to the parameters involved in potential callbacks.

Conclusion and Future Work

- Static-analysis results by prior static+dynamic schemes *may be unsound* in presence of corresponding callbacks at run-time.
- Overall low percentage of objects affected by callbacks enhances the confidence on the potential of existing static+dynamic schemes.
- It is important to develop schemes that maintain precision and soundness in their presence.
- Future research: Extend the idea of staged partial analysis to efficiently and precisely impart soundness in presence of dynamic features such as callbacks.

ACM Student Research Competition (SPLASH 2022), Auckland, New Zealand.

† Author address: ud21002@students.iitmandi.ac.in; School of Computing and Electrical Engineering, IIT Mandi, Himachal Pradesh, India.

• PhD Advisor: Manas Thakur, manas@iitmandi.ac.in; School of Computing and Electrical Engineering, IIT Mandi, Himachal Pradesh, India.