# Typestate Analysis for Android Applications

Ashish Mishra, Y.N. Srikant, Aditya Kanade

Indian Institute of Science

# Outline

# Android Applications

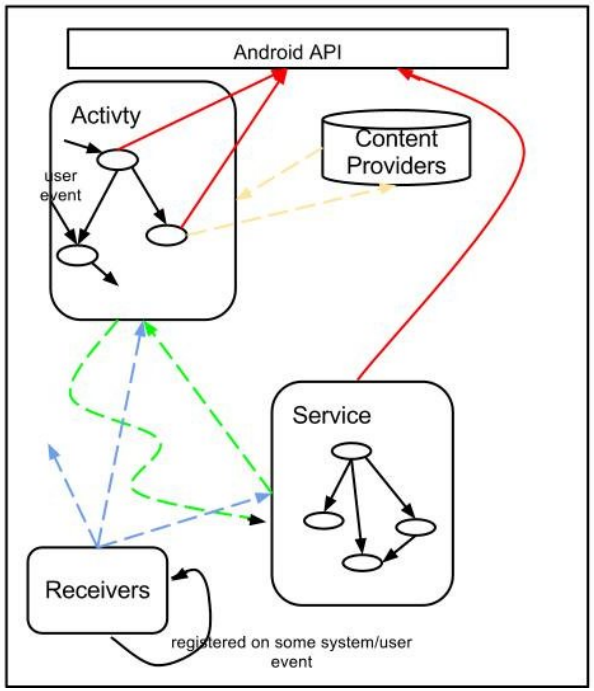
## Android Components

- Android applications are set of code segments called *Components* which interact with each other via a variant of IPC Binder called as *Intent*.
- Four different kinds of Components, *viz.* Activity , Service, ContentProvider and BroadcastReceiver.
- Each application (app) runs in a separate process for security and interact with other apps and System processes using *Asynchronous* Intents.

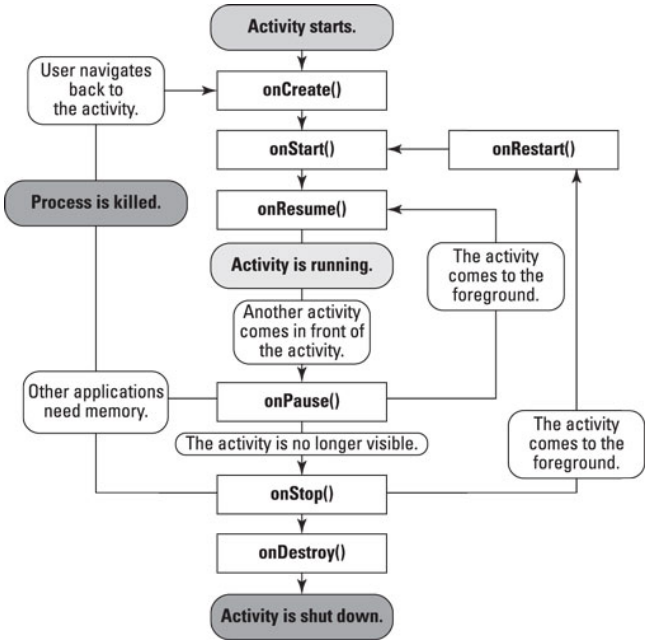
## Control Flow in Android Applications

- Apps are developed to run in an environment(Android Framework) which controls the major chunk of control flow in applications.
- Android apps have a complex control and data flow, which can be attributed to
  - Asynchronous method calls and message passing via Intent passing.
  - Life cycle callbacks from the System services to app components.
- Analysis of such programs need a precise modeling of this control and data flow in Android apps, failing which may lead both to imprecise and unsound results.

# Android Applications




Inter Component Communications




Component Life Cycle

source http://www.developer.android.com/

# Typestate

## Quoting Storm and Yemini

" ... Whereas the *type* of a data object determines the set of operations ever permitted on it, the *typestate* determines the subset of these operations which are permitted in a particular context".

## Typestate Analysis

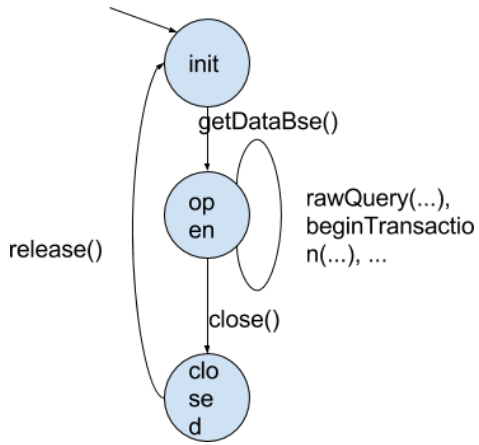
- The resource objects and many other Java objects, put a stateful restriction on the permitted operations on them.
- *Iterators* allow *next* operations only when the iterator has another element and the operation is invalid otherwise.
- Android system resource APIs for Camera, MediaPlayer etc. follow a much richer set of restriction rules.
- These bugs are hard to find because-
  - The point of detection is far separated from there point of origin by the runtime both in terms of time and space.
  - Caused by the semantics of these objects, independent from the normal semantics of the program.
  - Traces generated contain no information regarding the validity of the State of the object.

# A Database Android App.

```
1   FirstActivity extends Activity {
2       SQLiteDatabase mydatabase = null;
3       onCreate(Bundle savedInstanceState) {
4         myDBhelper = new MyDBHelper(this);
5         mydatabase = myDBhelper.
                getWritableDatabase();
6   Intent intent =
7  new Intent(this, DataBaseActivity.class);
8         startActivity(intent);
9       }
10      onStart() {
11        Cursor resultSet =
12  mydatabase.rawQuery("Select * from myTable",null
        );
13    [...]
14      }
15      onPause() {
16        mydatabase.close();
17      }
18    }
```

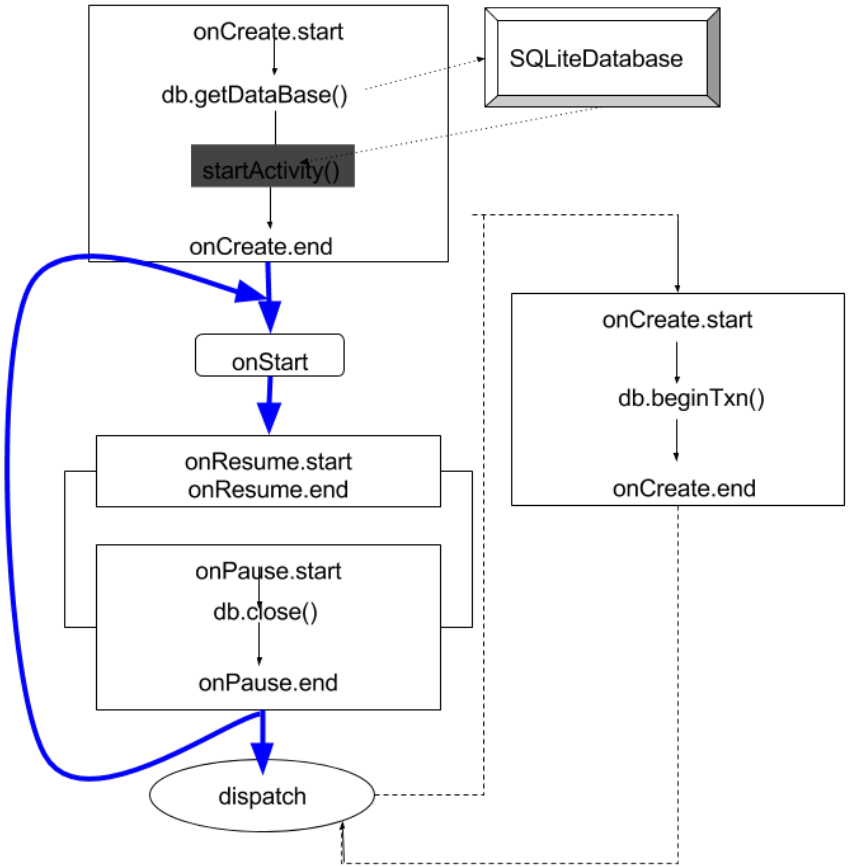
A DataBase Application

```
19  DataBaseActivity extends Activity {
20      onCreate(Bundle savedInstanceState) {
21      FirstActivity.mydatabase.beginTransaction
              ();
22      FirstActivity.mydatabase.close();
23      //ERROR
24      }
25  }
```




State Automaton for SQLiteDatabase

# Asynchronous Inter-Component Control Flow Graph

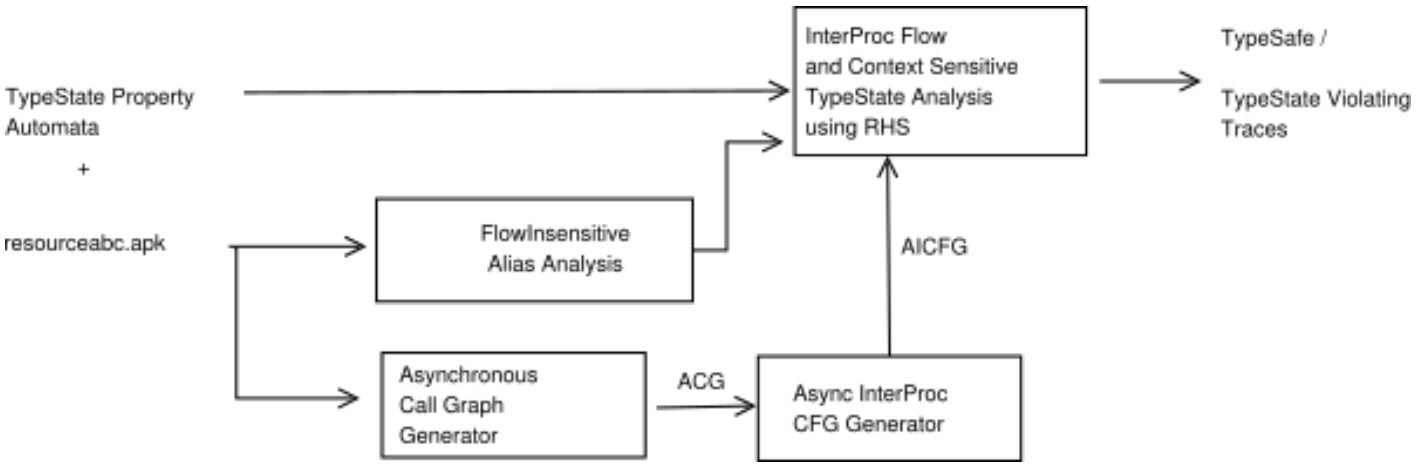



Asynchronous CFG for the Example app

# Approach Overview

## Analysis

- A precise and correct modelling of Android Asynchronous control flow, call backs and Component life cycles.
- Generating AICFG , an asycnhronous Inter-procedural Control Flow Graph for the application.
- A flow insensitive whole app , alias analysis to soundly track state changes. A flow and context sensitive whole app, Typestate analysis, using graph reachability based inter-procedural analysis.




Flow Diagram for Typestate analysis

# Results

- First work on Typestate analysis for Android applications.
- We add 10 new test benchmarks in Android benchmarks suite DroidBench, from the Soot Android analysis group.
- We Compare our results against the Typestate analysis build over the CFG used by other works (*IccTA*[1] and *AmanDroid*[2]).
- We perform better than them, both in terms of FPs and TPs.

| | | Model Based Analysis | | | | IccTA based Analysis | | | |
|---|---|---|---|---|---|---|---|---|---|
| AppName | Actual Violations | Violations found | TP | FN | FP | violations found | TP | FN | FP |
| Camra API | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| MediaPlayer API | 2 | 2 | 2 | 0 | 0 | 1 | 0 | 2 | 1 |
| SQLite API | 3 | 4 | 3 | 0 | 1 | 1 | 0 | 3 | 1 |
| DataBases | 2 | 2 | 2 | 0 | 0 | 1 | 0 | 2 | 1 |
| Files | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| SocketsFiles | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| Streams | 2 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 0 |

[1] *L. Li, et. al. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. In Proceedings of the 37th International Conference on Software Engineering (ICSE 2015), 2015*

[2] *F. Wei et. al. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS' 14, pages 1329-1341, New York, NY, USA, 2014. ACM.*

# Conclusion

- Gave a first precise and correct model for the Asynchronous control flow and life cycles and ICC in Android apps.
- Performed a first, sound Typestate analysis over android apps and compared the results against the control flow semantics used by other Android static analysis works.
- One limitation occurs due to the use of RHS for the analysis, which does not scale for big programs, increasing scalability is one future direction we aim at.

Thank you !
Happy to take Questions.