Static Analysis for Android: GUIs, Callbacks, and Beyond

Atanas (Nasko) Rountev
Joint work with
Dacong Yan
Shengqian Yang
Haowei Wu
Yan Wang
Hailong Zhang
Ohio State University

PRESTO: Program Analyses and Software Tools Research Group
Support by NSF CCF-1319695 and Google Faculty Research Award
Take-Home Messages

- Android software is important
- Foundations for control-flow and data-flow static analysis for Android are weak
- Need to use Android-specific semantics in static analysis algorithms: challenges and opportunities
- Many open problems
  - Foundational: static modeling of control/data flow
  - Analysis uses: beyond security
Take-Home Messages

- Android software is important
- Foundations for control-flow and data-flow static analysis for Android are weak
- Need to use Android-specific semantics in static analysis algorithms: challenges and opportunities
- Many open problems
  - Foundational: static modeling of control/data flow
  - Analysis uses: beyond security

**Exciting area for program analysis research**
Importance of Android

- Very large number of devices and apps
  - Estimate: 1.3 billion devices will be shipped in 2015
  - 1.5 million apps in Google Play, many thousands in other app stores (e.g., Amazon Appstore)

- Rapid growth and widespread use in daily life
  - Beyond phones and tablets: wearables, appliances, ...

- For PL and SE researchers: improved software quality and developer productivity through better program understanding, checking, transformation, optimization, testing, debugging, ...
Foundations for Static Analysis

- **Control-flow analysis**
  - Traditional: intra- and inter-procedural CFGs
  - Android: *event-driven* control flow; managed by the framework; often uses concurrency

- **Data-flow analysis**
  - Traditional: associate a lattice element with each CFG node; propagate using node transfer functions
  - Android: *silently propagates data* through the framework code; *special values* (e.g., integers used as ids); complex *Android-specific semantics* for some CFG nodes
Foundations for Static Analysis

- **Control-flow analysis**
  - Traditional: intra- and inter-procedural CFGs
  - Android: *event-driven* control flow; managed by the framework; often uses concurrency

- **Data-flow analysis**
  - Traditional: associate a lattice element with each CFG node; propagate using node transfer functions
  - Android: *silently propagates data* through the framework code; *special values* (e.g., integers used as ids); complex *Android-specific semantics* for some CFG nodes

We do not know how to perform general control/data-flow analysis for Android
Two Building Blocks of Control-Flow Analysis

- GUI events and their handlers [CGO’14]
  - More generally, what is the **structure of the GUI**?
  - Challenges: modeling of high-level semantics for Android constructs; many features and variations

- GUI changes triggered by event handlers [ICSE’15]
  - More generally, what is the **behavior of the GUI**?
  - Limited focus: GUI-event control flow in the main thread
Two Building Blocks of Control-Flow Analysis

- GUI events and their handlers [CGO’14]
  - More generally, what is the structure of the GUI?
  - Challenges: modeling of high-level semantics for Android constructs; many features and variations

- GUI changes triggered by event handlers [ICSE’15]
  - More generally, what is the behavior of the GUI?
  - Limited focus: GUI-event control flow in the main thread

- Should we include the framework code? No
  - Unlike whole-program analysis for Java, we “embed” the high-level semantics of android.* classes in the analysis
  - Benefits and disadvantages
Two Building Blocks of Control-Flow Analysis

- GUI events and their handlers [CGO’14]
  - More generally, what is the structure of the GUI?
  - Challenges: modeling of high-level semantics for Android constructs; many features and variations

- GUI changes triggered by event handlers [ICSE’15]
  - More generally, what is the behavior of the GUI?
  - Limited focus: GUI-event control flow in the main thread
Windows, Widgets, and Handlers

- **GUI elements**
  - **Activity**: on-screen window with GUI widgets (**views**)
    - Other windows as well: **menus** and **dialogs**
  - **Event handlers**: defined in **listener objects** and associated with views to respond to user actions

- **Need to model statically:**
  - Views and their hierarchical structure
  - Association of views with activities
  - Association of views with listeners
  - Variables that refer to views, activities, and listeners

- **Underneath, this is a form of** **points-to analysis**
MyActivity.java:
    class MyActivity extends Activity {
        void onCreate() {
            this.setContentView(R.layout.main); // Inflate
            View a = this.findViewById(R.id.my_btn); // FindView
            Button b = (Button) a;
            ButtonListener c = new ButtonListener();
            b.setOnClickListener(c); // SetListener }
    }

ButtonListener.java:
    class ButtonListener implements OnClickListener {
        void onClick(View d) { ... } }

main.xml:
    <RelativeLayout ...
        <Button android:id="@+id/my_btn" ... />
    </RelativeLayout>
MyActivity.java:

```java
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener 
    }
}
```

ButtonListener.java:

```java
class ButtonListener implements OnClickListener {
    void onClick(View d) { ... } 
}
```

main.xml:

```xml
<RelativeLayout ...>
    <Button android:id="@+id/my_btn" ... />
</RelativeLayout>
```
MyActivity.java:
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener }  }

ButtonListener.java:
class ButtonListener implements OnClickListener {
    void onClick(View d) { ... }  }

main.xml:
<RelativeLayout ...
    <Button android:id="@+id/my_btn" ... />
</RelativeLayout>
MyActivity.java:

```java
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener
    }
}
```

ButtonListener.java:

```java
class ButtonListener implements OnClickListener {
    void onClick(View d) { ... }
}
```

main.xml:

```xml
<RelativeLayout ...>
    <Button android:id="@+id/my_btn" ... />
</RelativeLayout>
```
MyActivity.java:

class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener }  }

ButtonListener.java:

class ButtonListener implements OnClickListener {
    void onClick(View d) { ... }  }

main.xml:

<RelativeLayout ...
    <Button android:id="@+id/my_btn" ... />
</RelativeLayout>
Android-Specific Semantics

- **Inflate**: create a hierarchy of views from XML specs and attach to an activity or to a view
- **CreateView**: programmatically create with `new V`
- **FindView**: look up a view from activity or from ancestor view (e.g., using integer id)
- **SetListener**: associate view and listener
- **AddView**: parent-child relationship for two views
- **SetId**: programmatically set the id of a view

This high-level semantics is integrated with a standard constraint-based points-to-like analysis for Java
class MyActivity extends Activity {
    void onCreate() {
      this.setContentView(R.layout.main); // Inflate
      View a = this.findViewById(R.id.my_btn); // FindView
      Button b = (Button) a;
      ButtonListener c = new ButtonListener();
      b.setOnClickListener(c); // SetListener } }  
  void onClick(View d) { ... } }

Propagation edges and relevant nodes
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main); // Inflate
        View a = this.findViewById(R.id.my_btn); // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c); // SetListener } }

    void onClick(View d) { ... } }

Propagation edges and relevant nodes
1 class MyActivity extends Activity {
2   void onCreate() {
3      this.setContentView(R.layout.main); // Inflate
4      View a = this.findViewById(R.id.my_btn); // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c); // SetListener }
8   ...
9   void onClick(View d) { ... } }

class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener
    }
    void onClick(View d) {
    }
}
1 class MyActivity extends Activity {
2   void onCreate() {
3      this.setContentView(R.layout.main);  // Inflate
4      View a = this.findViewById(R.id.my_btn);  // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c);  // SetListener
8    }  // onCreate
9  }
10 }
11 void onClick(View d) { ... }  // on click
1 class MyActivity extends Activity {
2    void onCreate() {
3        this.setContentView(R.layout.main); // Inflate
4        View a = this.findViewById(R.id.my_btn); // FindView
5        Button b = (Button) a;
6        ButtonListener c = new ButtonListener();
7        b.setOnClickListener(c); // SetListener } }
8  void onClick(View d) { ... } }
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main); // Inflate
        View a = this.findViewById(R.id.my_btn); // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c); // SetListener } }

    void onClick(View d) { ... } }

Propagation edges and relevant nodes
class MyActivity extends Activity {
    void onCreate() {
        this setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener } }
    void onClick(View d) { ... } }

Propagation edges and relevant nodes
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main); // Inflate
        View a = this.findViewById(R.id.my_btn); // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c); // SetListener } }

<RelativeLayout ...
<Button android:id="@+id/my_btn" ... />
</RelativeLayout>
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c);  // SetListener
    }
    ...
    ...
    ....

    <RelativeLayout ...> 
    <Button android:id="@+id/my_btn" ... />
    </RelativeLayout>
1  class MyActivity extends Activity {
2    void onCreate() {
3      this.setContentView(R.layout.main); // Inflate
4      View a = this.findViewById(R.id.my_btn); // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c); // SetListener } }

10  <RelativeLayout ...>
11   <Button android:id="@+id/my_btn" ... />
12  </RelativeLayout>

Property edges and relevant nodes

[Diagram of RelativeLayout and Button nodes with view id and child connections]
1 class MyActivity extends Activity {
2   void onCreate() {
3     this.setContentView(R.layout.main); // Inflate
4     View a = this.findViewById(R.id.my_btn); // FindView
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main);  // Inflate
        View a = this.findViewById(R.id.my_btn);  // FindView
    }
}
```java
1  class MyActivity extends Activity {
2    void onCreate() {
3      this.setContentView(R.layout.main);  // Inflate
4      View a = this.findViewById(R.id.my_btn);  // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c);  // SetListener }  }
```

Property edges and relevant nodes:

- **MyActivity**
- **RelativeLayout**
- **inflate**
- **id:my_btn**
- **Button**

**Property edges and relevant nodes:**

1. **MyActivity** to **RelativeLayout** (root)
2. **RelativeLayout** to **inflate**
3. **inflate** to **Button** (child)
4. **id:my_btn** to **Button** (view id)
```java
1  class MyActivity extends Activity {
2    void onCreate() {
3      this.setContentView(R.layout.main);  // Inflate
4      View a = this.findViewById(R.id.my_btn);  // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c);  // SetListener }  }
```
class MyActivity extends Activity {

    void onCreate() {
        this setContentView(R.layout.main); // Inflate
        View a = this.findViewById(R.id.my_btn); // FindView
        Button b = (Button) a;
        ButtonListener c = new ButtonListener();
        b.setOnClickListener(c); // SetListener
    } // onCreate
} // MyActivity
1 class MyActivity extends Activity {
2    void onCreate() {
3      this.setContentView(R.layout.main);  // Inflate
4      View a = this.findViewById(R.id.my_btn);  // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c);  // SetListener
8    }
9  }

Property edges and relevant nodes
class MyActivity extends Activity {
    void onCreate() {
        this.setContentView(R.layout.main); // Inflate
        View a = this.findViewById(R.id.my_btn); // FindView
        Button b = (Button) a;
        ButtonListener c =
        b.setOnClickListener(c); // SetListener } }
```java
1  class MyActivity extends Activity {
2    void onCreate() {
3      this.setContentView(R.layout.main);  // Inflate
4      View a = this.findViewById(R.id.my_btn);  // FindView
5      Button b = (Button) a;
6      ButtonListener c = new ButtonListener();
7      b.setOnClickListener(c);  // SetListener } } 
```
Implementation

- **Input:** Java bytecode and relevant XML files
- **Output**
  - Parent-child hierarchy for views
  - Association of windows with root views
  - Association of views with listeners
  - Variables/fields pointing to views, activities, listeners
- **Algorithm (in Soot)**
  - Initial constraint graph from app code
  - Solve propagation constraints for ids, windows, listeners
  - Fixed-point computation for the flow of views
- **Fast running time; reasonable precision; room for improvement (precision & Android GUI features)**
Two Building Blocks of Control-Flow Analysis

- GUI events and their handlers [CGO’14]
  - More generally, what is the structure of the GUI?
  - Challenges: modeling of high-level semantics for Android constructs; many features and variations

- GUI changes triggered by event handlers [ICSE’15]
  - More generally, what is the behavior of the GUI?
  - Limited focus: GUI-event control flow in the main thread
Control Flow in the UI Thread

- Given the windows, views, and handlers, what is the possible control flow due to GUI events?

- A GUI event triggers **callbacks** for
  - **Event handling**: e.g., `onClick`
  - **Window lifetime management**: e.g., if `onClick` starts a new activity, `onCreate` will be called on this activity by the framework code

**What are all possible sequences of such callbacks?**
Control Flow in the UI Thread

- Given the windows, views, and handlers, what is the possible control flow due to GUI events?

- A GUI event triggers **callbacks** for
  - **Event handling**: e.g., onClick
  - **Window lifetime management**: e.g., if onClick starts a new activity, onCreate will be called on this activity by the framework code

More generally: **What are all possible sequences of callbacks?** What are the values for their framework-provided parameters? Key question for control/data-flow analysis, but no good answer yet
class Main extends Activity {
    void onCreate() {
        ... // Add four ImageButton widgets with ids R.id.infoBtn, R.id.helpBtn, R.id.manageBtn, and R.id.multiBtn
        EventHandler handler = new EventHandler();
        ... // Associate handler with each of the four buttons
    }
}

class EventHandler implements OnClickListener {
    void onClick(View v) {
        switch (v.getId()) {
            case R.id.infoBtn:
                Intent info = new Intent(DirectoryInfo.class); startActivity(info); break;
            case R.id.helpBtn:
                Intent help = new Intent(HelpManager.class); startActivity(help); break;
            case R.id.manageBtn:
                AlertDialog dialog = ...; dialog.show(); break;
            default: ...; break;
        }
    }
}
class Main extends Activity {
    void onCreate() {
        ... // Add four ImageButton widgets with ids R.id.infoBtn,
        ... // R.id.helpBtn, R.id.manageBtn, and R.id.multiBtn
        EventHandler handler = new EventHandler();
        ... // Associate handler with each of the four buttons }
}

class EventHandler implements OnClickListener {
    void onClick(View v) {
        switch (v.getId()) {
            case R.id.infoBtn:
                Intent info = new Intent(DirectoryInfo.class); startActivity(info); break;
            case R.id.helpBtn:
                Intent help = new Intent(HelpManager.class); startActivity(help); break;
            case R.id.manageBtn:
                AlertDialog dialog = ...; dialog.show(); break;
            default: ...; break; }
    }
}
Our Goals

- Representation of callback ordering constraints
  - Within a window and across windows
  - Consider the **invocation context** of a callback: the framework-provided parameter representing the **view** (for event handlers) or the **window** (for lifecycle callbacks)

- Context-sensitive interprocedural analysis of callbacks to find ordering constraints

- Client analysis: GUI model construction for program understanding and testing
Given [handler, widget], what could be the next callback executed after handler completes?

```java
void onClick(View v) {
    switch (v.getId()) {
        case R.id.infoBtn:
            Intent info = new Intent(DirectoryInfo.class);
            startActivity(info); break;
        case R.id.helpBtn:
            Intent help = new Intent(HelpManager.class);
            startActivity(help); break;
        case R.id.manageBtn:
            AlertDialog dialog = …;
            dialog.show(); break;
        default: …; break; }
```

Trigger: API call to open a new window, or to close the current one.
Analysis Algorithm

- Given [handler, widget], which trigger statements are reached? Can we avoid all triggers?
- Interprocedural reachability on the control-flow graphs of handler and its transitive callees
  - Matching of calls and returns during traversal
  - Finds reachable triggers
  - Determines if the exit of handler is reached without going through triggers
- But: we also need to take into account widget
void onClick(View v) {
    switch (v.getId()) {
        case R.id.infoBtn:
            Intent info = new Intent(DirectoryInfo.class); startActivity(info); break;
        case R.id.helpBtn:
            Intent help = new Intent(HelpManager.class); startActivity(help); break;
        case R.id.manageBtn:
            AlertDialog dialog = ...; dialog.show(); break;
        default: ...; break;
    }
}

Context-insensitive analysis of [onClick, info-btn]

    triggers = { startActivity(info), startActivity(help), dialog.show() }

can avoid triggers = true

Context-sensitive analysis of [onClick, info-btn]

    triggers = { startActivity(info) }

can avoid triggers = false
Adding Context Sensitivity

- Given [handler, widget], which control-flow graph edges are feasible under context widget?
  
  \[
  \text{switch(v.getId()) for info-btn: which branch is feasible?}
  \]

- Interprocedural constant propagation of
  
  - references: v at v.getId() can only be the static widget info-btn
  
  - integers for widget ids: v.getId() can only be the integer constant R.id.infoBtn
  
  - boolean: \((x == y)\) and \((x != y)\) for widget ids
Evaluation

- Implementation in our GATOR analysis toolkit
  - web.cse.ohio-state.edu/presto/software
  - Based on the Soot framework
  - Latest version released in April; planning a new version in the summer

- Evaluated on 20 open-source applications
  - **Cost**: less than 1 min for most of the analyzed applications
  - **Precision**: context sensitivity significantly improves CCFGs and GUI models derived from them
Average Out-degree of Event Handler Nodes

[Bar chart showing the average out-degree of event handler nodes for various applications, with categories for context-insensitive and context-sensitive nodes.]
Example of a Client: GUI Model

- Nodes are **windows**, edges are **transitions** triggered by event handlers
  - Not complete: does not consider default events and general window-close effects
  - For program understanding and test generation

- Comparison of
  - Context-insensitive and context-sensitive analysis
  - **Precise static solution** (constructed manually)
  - **GUI ripping tool**: which edges from the precise static solution are discovered during ripping
Open Questions

Foundations for control-flow and data-flow analysis for Android are weak

- Proper abstractions for GUI-driven control flow: more complex than the CCFG; we have some new results (similar to “ICFG valid path traversal”)
- Other callbacks: e.g., battery changes; GPS reads; ...
- Concurrency: e.g., AsyncTask
- Implicit data flow through the framework code
- Dependences between analyses: e.g., GUI analysis and intent analysis depend on each other
Open Questions

Need to use Android-specific semantics in static analysis algorithms

- Semantics is unspecified, complex, and evolving
- How do we capture the relevant aspects of the high-level semantics? How do we verify that our understanding is correct? How do we evolve these specifications with new Android releases? How do we evaluate the “coverage” of this semantics by different static analyses?
- Will probably have to employ run-time analysis
Open Questions

Diversify the uses of static analysis for Android

- A lot of work related to security, but what else?
- Evolution due to API/device changes
- Automated test generation (we have some initial work)
- Performance optimizations: responsiveness, energy
- Static checking: leaks, concurrency, etc.
- Detection of app cloning
- More ...
Thank you
Questions?