Static Analysis for Android: GUIs, Callbacks, and Beyond

Atanas (Nasko) Rountev

Ohio State University

Program Analyses and Software Tools Research Group

Joint with my students Dacong Yan, Shengqian Yang, Haowei Wu, Yan Wang, Hailong Zhang, Chandrasekar Swaminathan, Sufian Latif

Support by NSF awards 1319695/1526459 and Google Faculty Research award

Take-Home Messages

Weak foundations for static control-flow and data-flow analysis for Android GUIs

- Progress in the last few years [CGO14][ICSE15][AST15][PhD14][PhD15]
- Many open problems [SOAP16]

Useful GUI models built via static analysis

- Static analysis of resource leaks [CC16]
- Automated test generation [AST16][AST18]
- Responsiveness profiling [MobileSoft17]

Interesting problems beyond plain Android

- GUI analysis and testing for Android Wear [ICSE17]

Importance of Android

- Large number of devices and apps
  - 2.4 billion devices
  - 3.7 million apps in Google Play; many other app stores
- Widespread use in daily life
  - Phones, tablets, electronics, wearables, appliances, auto
- For SE and PL researchers: improved software quality and developer productivity through better program understanding, checking, transformation, optimization, testing, debugging, security analysis
  - Need static analysis machinery as a critical building block

Foundations for Static Analysis

Control-flow analysis

– Traditional: control-flow graphs
– Android: event-driven framework-managed control flow

Data-flow analysis

– Traditional: associate a solution with each graph node; propagate along graph edges and paths
– Android: silently propagates data through the framework code; special values (e.g., integers used as ids); complex Android-specific semantics for some graph nodes

Foundations for Static Analysis

Control-flow analysis

- Traditional: control-flow graphs
- Android: event-driven framework-managed control flow

Data-flow analysis

- Traditional: associate a solution with each graph node; propagate along graph edges and paths
- Android: silently propagates data through the framework code; special values (e.g., integers used as ids); complex Android-specific semantics for some graph nodes

We still don't know how to perform general control-flow and data-flow analysis for Android

Two Building Blocks of Control-Flow Analysis

GUI widgets, events, and handlers [CGO14][PhD14]

- What is the structure of the GUI?
- Challenge: modeling of Android API semantics

GUI changes due to event handlers [ICSE15][ASE15][PhD15]

- What is the behavior of the GUI?
- Challenge: complex sequences of callbacks

Two Building Blocks of Control-Flow Analysis

GUI widgets, events, and handlers [CGO14][PhD14]

- What is the structure of the GUI?
- Challenge: modeling of Android API semantics

GUI changes due to event handlers [ICSE15][ASE15][PhD15]

- What is the behavior of the GUI?
- Challenge: complex sequences of callbacks

Windows, Widgets, and Handlers

GUI elements

- Activity: on-screen window with GUI widgets (views)
- Event handlers: defined in listener objects

Need to model statically:

- Views and their hierarchical structure
- Association of views with activities
- Association of views with 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:

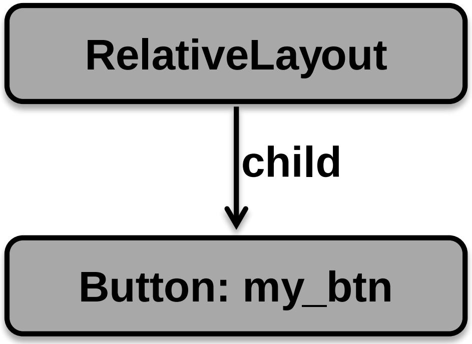
```
class MyActivity extends Activity {
  void onCreate() {
    this.setContentView(R.layout.main);  // Inflate
    View a = this.findViewById(R.id.my_btn);  // FindView
    Button b = (Button) a;
    ButtonListener c = new ButtonListener();
    b.setOnClickListener(c);  // SetListener  }  }
```

ButtonListener.java:

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

main.xml:

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



MyActivity.java:

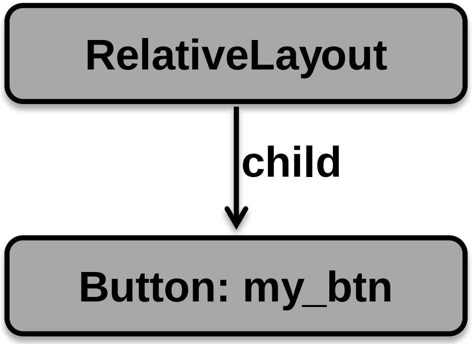
```
class MyActivity extends Activity {
   void onCreate() {
      this.setContentView(R.layout.main);  // Inflate
      View a = this.findViewById(R.id.my_btn);  // FindView
      Button b = (Button) a;
      ButtonListener c = new ButtonListener();
      b.setOnClickListener(c);  // SetListener  }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

ButtonListener.java:

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

main.xml:

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



MyActivity.java:

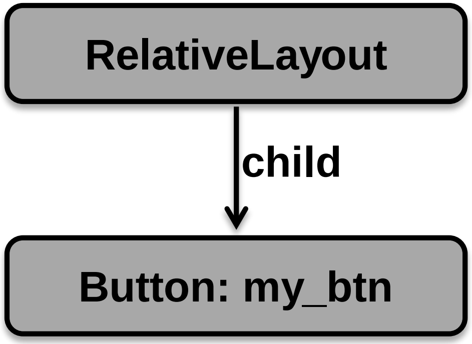
```
class MyActivity extends Activity {
   void onCreate() {
      this.setContentView(R.layout.main);  // Inflate
      View a = this.findViewById(R.id.my_btn);  // FindView
      Button b = (Button) a;
      ButtonListener c = new ButtonListener();
      b.setOnClickListener(c);  // SetListener  }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

ButtonListener.java:

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

main.xml:

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



MyActivity.java:

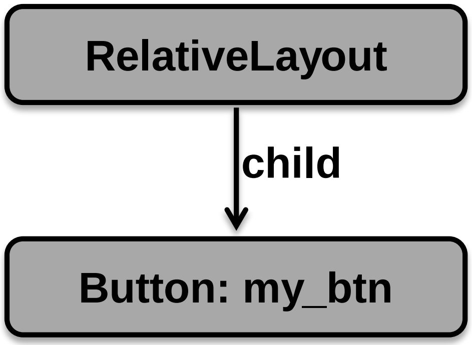
```
class MyActivity extends Activity {
  void onCreate() {
    this.setContentView(R.layout.main);  // Inflate
    View a = this.findViewById(R.id.my_btn);  // FindView
    Button b = (Button) a;
    ButtonListener c = new ButtonListener();
    b.setOnClickListener(c);  // SetListener  }  }
```

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**

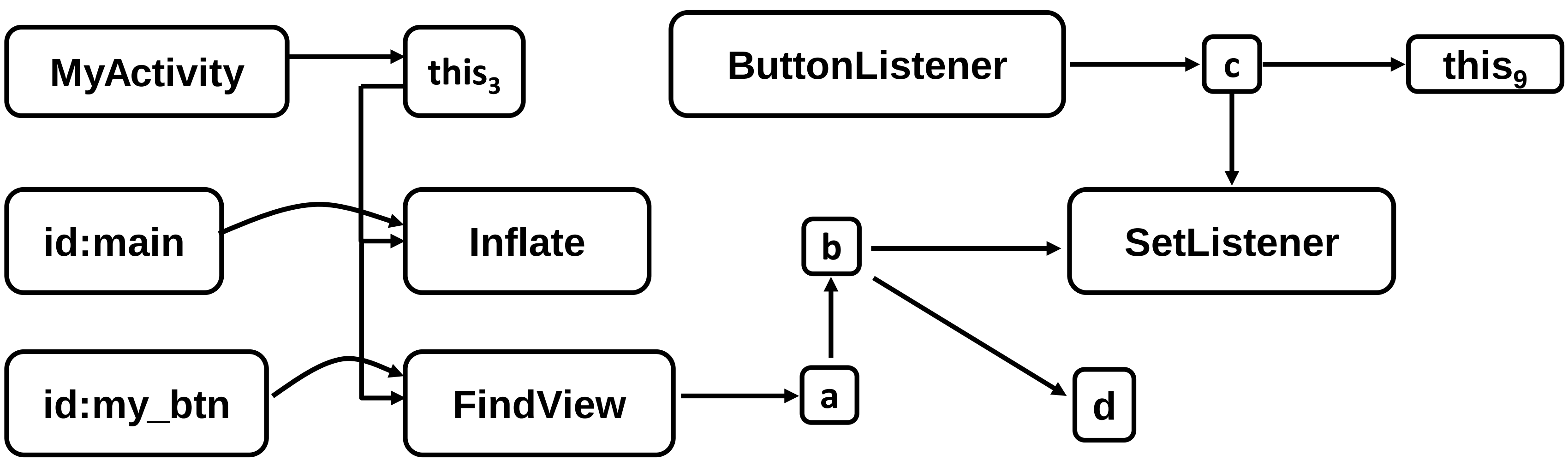
FindView: look up a view in hierarchy

SetListener: associate view and listener

AddView: parent-child relationship for two views

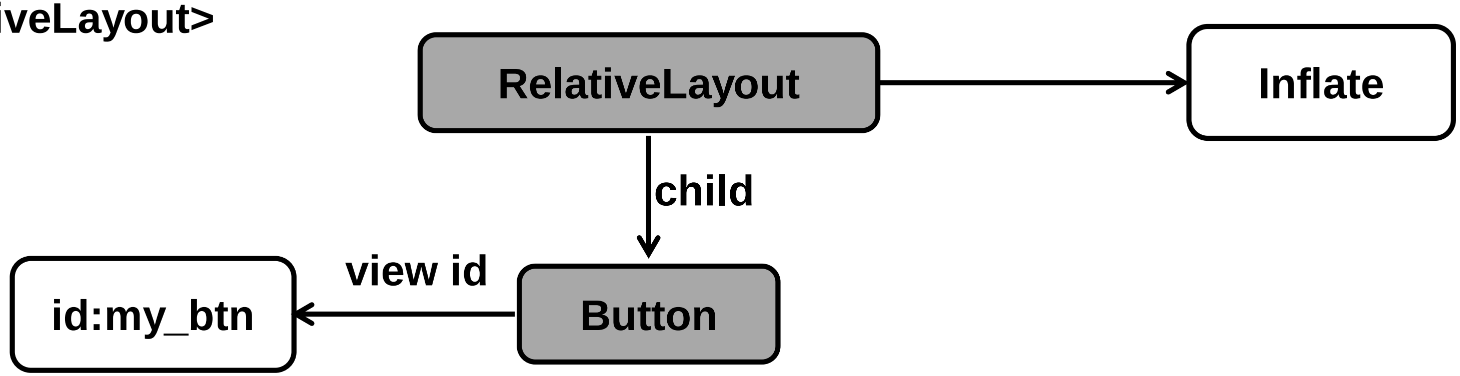
SetId: programmatically set the id of a view

```
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) { ... }  }
```

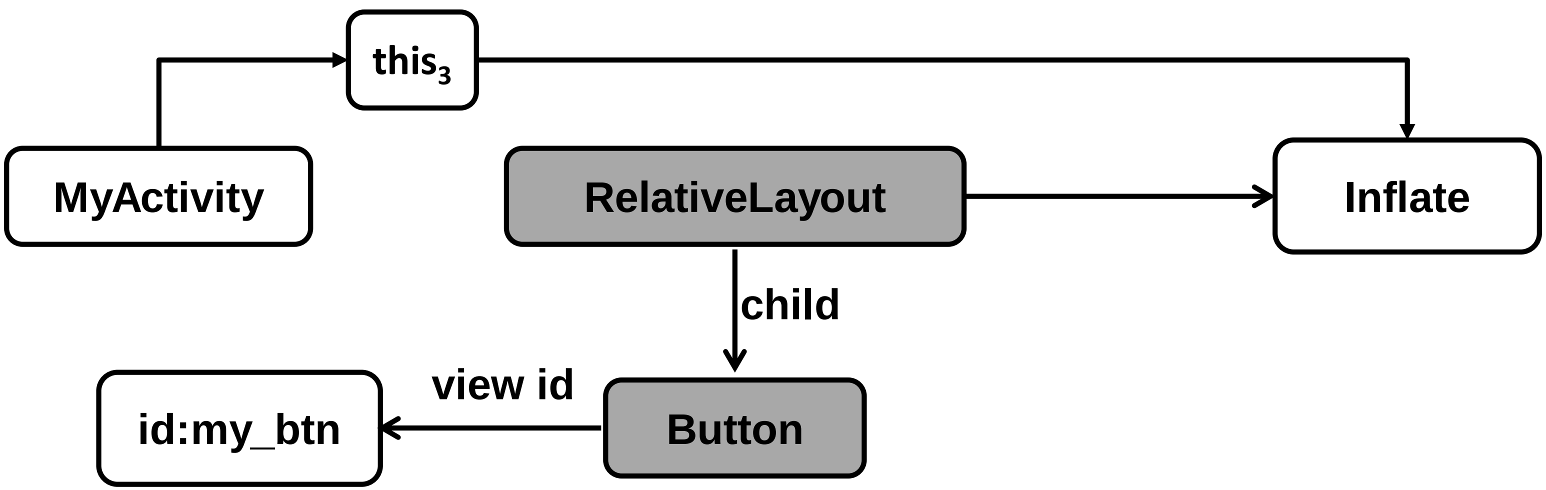


```
class MyActivity extends Activity {
  void onCreate() {
    this.setContentView(R.layout.main);  // Inflate
```

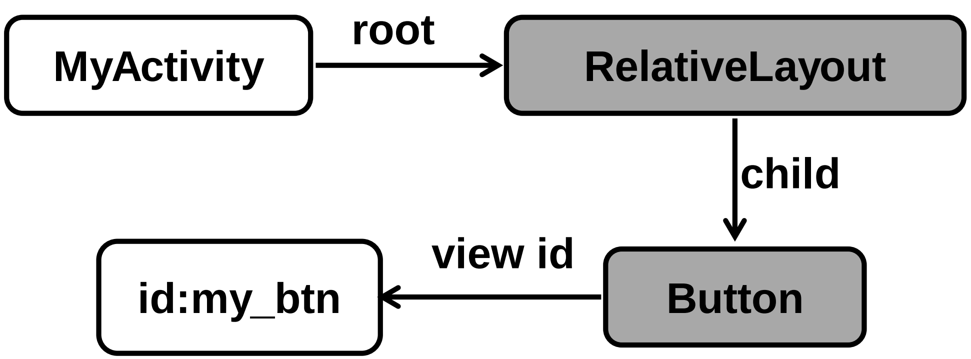
```
<RelativeLayout ...>
    <Button android:id="@+id/my_btn" ... />
</RelativeLayout>
```



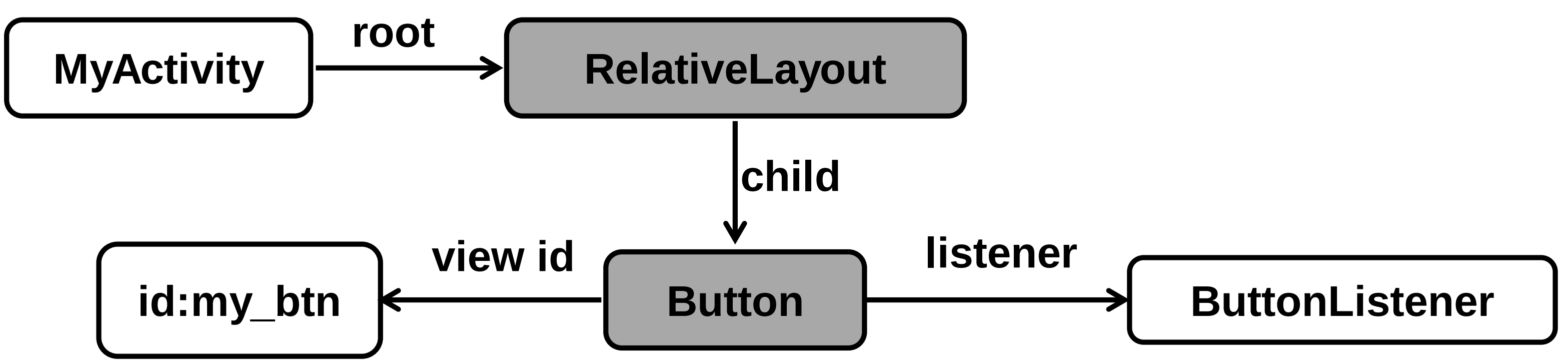
```
class MyActivity extends Activity {
  void onCreate() {
    this.setContentView(R.layout.main);  // Inflate
```



```
class MyActivity extends Activity {
  void onCreate() {
    this.setContentView(R.layout.main);  // Inflate
```



```
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  }  }
```



Implementation and Evaluation

GATOR : Program analysis toolkit for Android
– http://web.cse.ohio-state.edu/presto

Analysis implementation
– Input: Dalvik bytecode and relevant XML files
– Bytecode → Soot's intermediate representation
– Propagation for ids, windows, listeners, views
– Output: static abstractions of activities, dialogs, menus, view hierarchies, listeners

Good precision and running time; room for improvement (precision, cost, Android features)

Two Building Blocks of Control-Flow Analysis

GUI widgets, events, and handlers [CGO14][PhD14]
- What is the structure of the GUI?
- Challenge: modeling of Android API semantics

GUI changes due to event handlers [ICSE15][ASE15][PhD15]
- What is the behavior of the GUI?
- Challenge: complex sequences of callbacks

Control-Flow Analysis of Android GUIs

Event-driven control flow

- **Event handler callback** responds to a GUI event
- The callback can trigger a **window transition**
- Additional **lifecycle callbacks** during transition

Key observation: the effects of a GUI event depend on the **history** of prior events

What are all possible sequences of GUI events, windows transitions, and related callbacks?

Our Solution

Window transition graph (WTG)

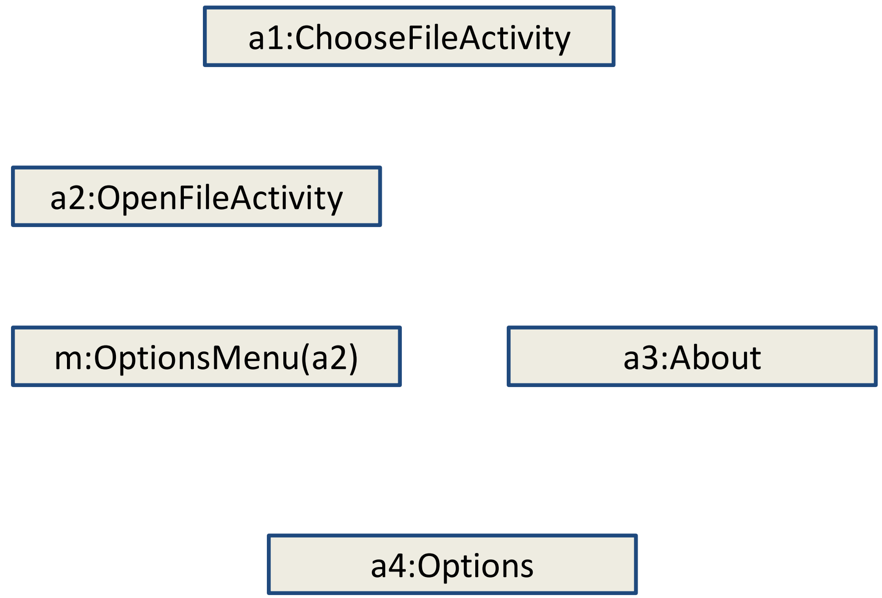
– Static model to represent possible sequences of GUI events, windows, and callbacks

Static analysis to build the WTG

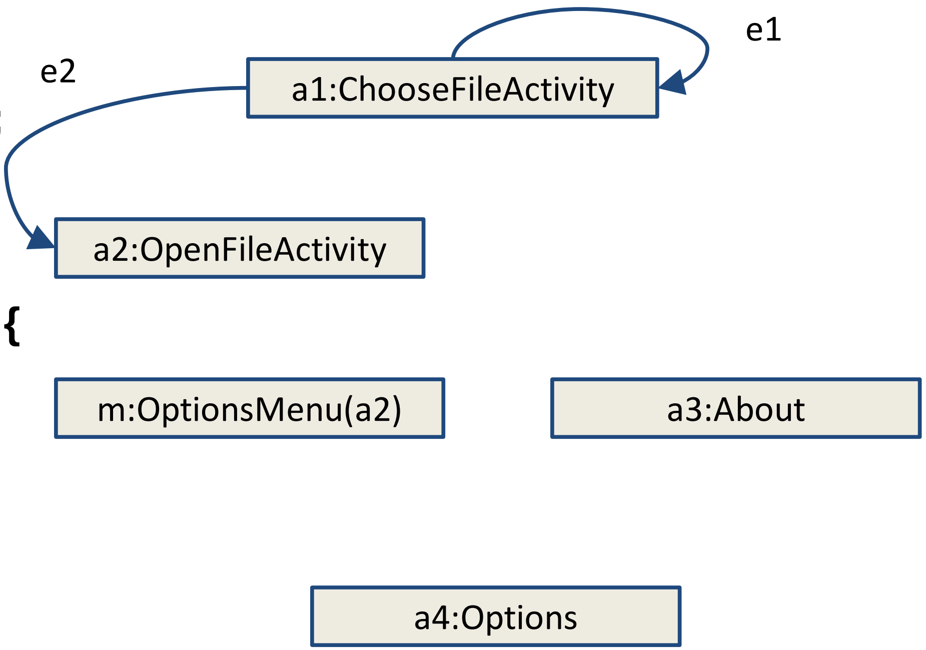
– Key new abstraction: **window stack**, which represents the stack of currently-alive windows

```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
   if (…) return;
   Intent i = new Intent(OpenFileActivity.class);
   startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
   if (item == aboutItem) {
     startActivity(new Intent(About.class)); }
   if (item == optionsItem) {
     startActivity(new Intent(Options.class));
     this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
   startActivity(new Intent(About.class));
   this.finish(); } } }

class About extends Activity { ... }
```

```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
  if (…) return;
  Intent i = new Intent(OpenFileActivity.class);
  startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
  if (item == aboutItem) {
   startActivity(new Intent(About.class)); }
  if (item == optionsItem) {
   startActivity(new Intent(Options.class));
   this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
  startActivity(new Intent(About.class));
  this.finish(); } } }

class About extends Activity { ... }
```



```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
  if (…) return;
  Intent i = new Intent(OpenFileActivity.class);
  startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
  if (item == aboutItem) {
   startActivity(new Intent(About.class)); }
  if (item == optionsItem) {
   startActivity(new Intent(Options.class));
   this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
  startActivity(new Intent(About.class));
  this.finish(); } } }

class About extends Activity { … }
```



Example: information for edge **e2**

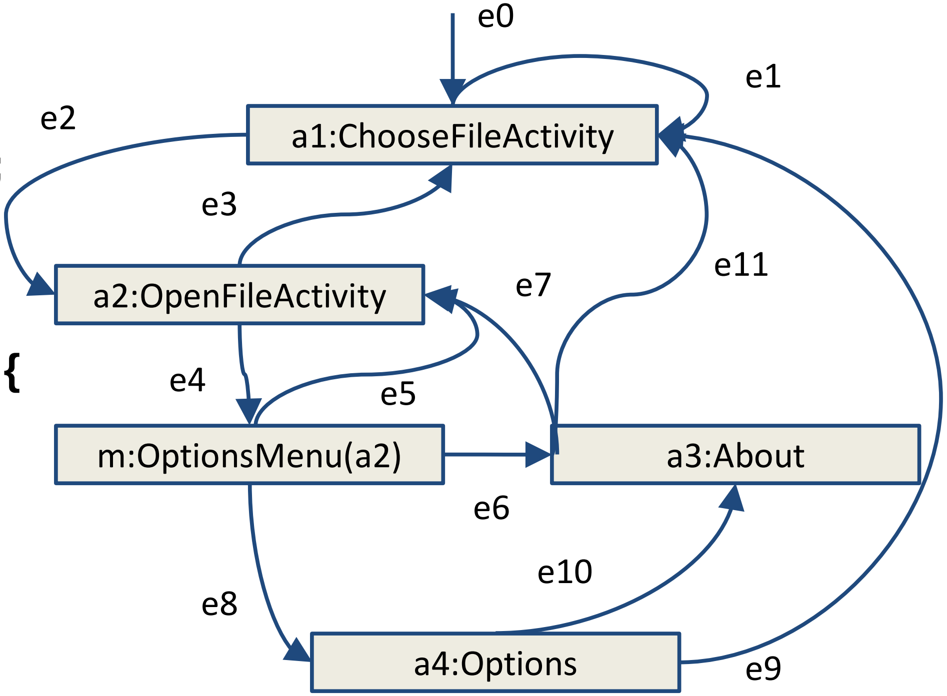
widget: item

event type: click

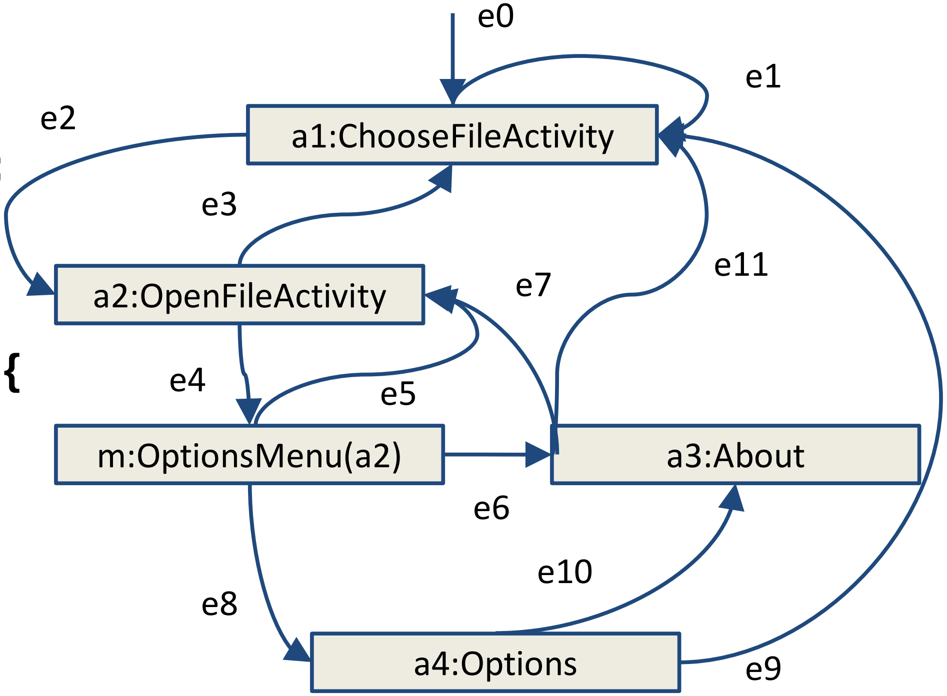
window stack: push(a2)

callbacks: onItemClick(item), onPause(a1), onCreate(a2), onStart(a2), onResume(a2), onStop(a1)

```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
  if (…) return;
  Intent i = new Intent(OpenFileActivity.class);
  startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
  if (item == aboutItem) {
   startActivity(new Intent(About.class)); }
  if (item == optionsItem) {
   startActivity(new Intent(Options.class));
   this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
  startActivity(new Intent(About.class));
  this.finish(); } } }

class About extends Activity { … }
```



```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
   if (…) return;
   Intent i = new Intent(OpenFileActivity.class);
   startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
   if (item == aboutItem) {
     startActivity(new Intent(About.class)); }
   if (item == optionsItem) {
     startActivity(new Intent(Options.class));
     this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
   startActivity(new Intent(About.class));
   this.finish(); } } }

class About extends Activity { ... }
```



Example: information for edge **e6**

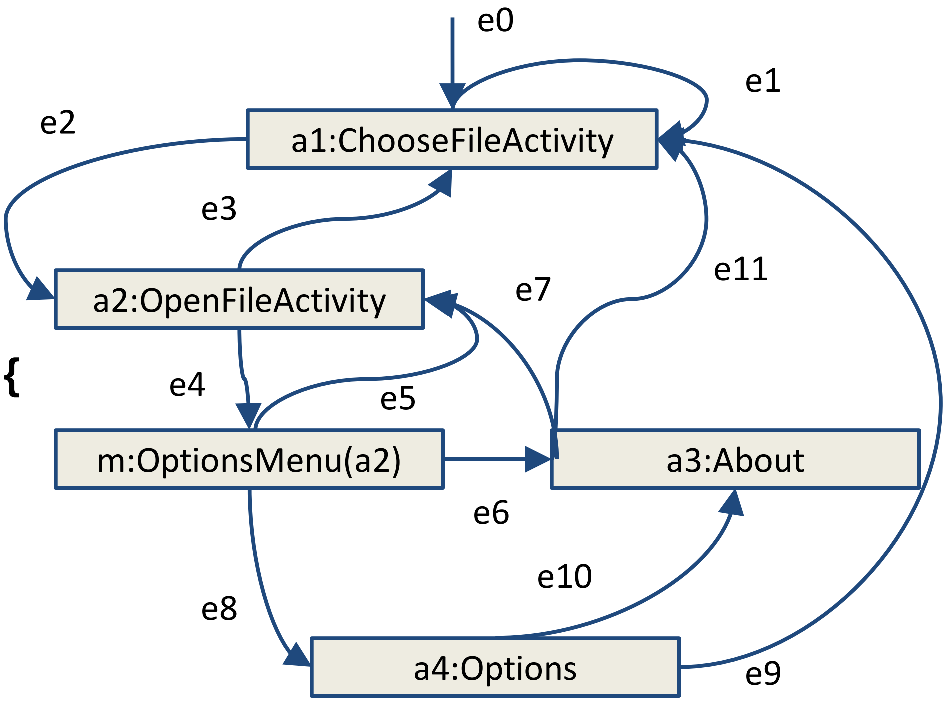
widget: aboutItem

event type: click

window stack: pop(m) push(a3)

callbacks: ...

```
class ChooseFileActivity extends Activity {
 void onItemClick(ListView l, View item) {
  if (…) return;
  Intent i = new Intent(OpenFileActivity.class);
  startActivity(i); } }

class OpenFileActivity extends Activity {
 void onOptionsItemSelected(MenuItem item) {
  if (item == aboutItem) {
   startActivity(new Intent(About.class)); }
  if (item == optionsItem) {
   startActivity(new Intent(Options.class));
   this.finish(); } }

class Options extends Activity {
 void onClick(View v) {
  startActivity(new Intent(About.class));
  this.finish(); } } }

class About extends Activity { ... }
```



Example: information for edge **e8**

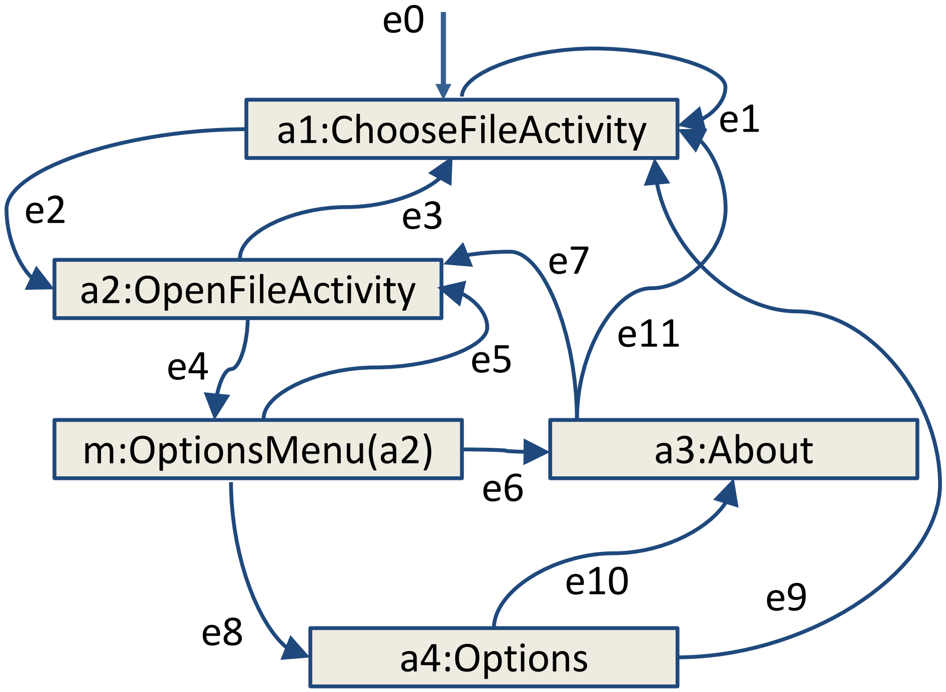
widget: optionsItem

event type: click

window stack: pop(m) pop(a2) push(a4)

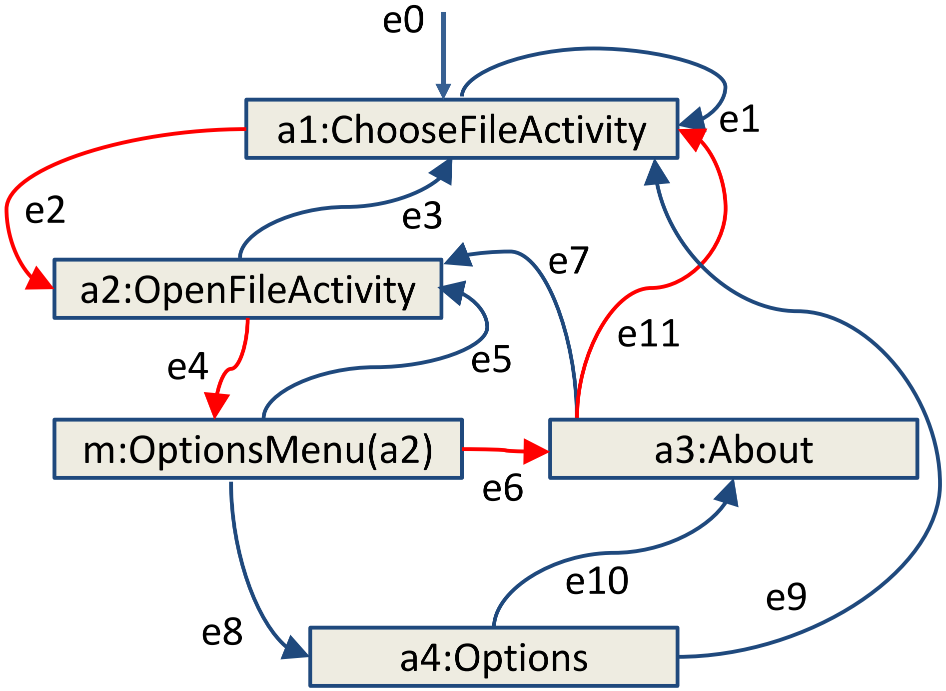
callbacks: ...

Final Graph



e0: launch, push(a1)
e1: item, click, —
e2: item, click, push(a2)
e3: back, pop(a2)
e4: menu, push(m)
e5: back, pop(m)
e6: aboutItem, click, pop(m) push(a3)
e8: optionsItem, click, pop(m) pop(a2) push(a4)
e7: back, pop(a3)
e9: back, pop(a4)
e10: btn, click, pop(a4) push(a3)
e11: back, pop(a3)

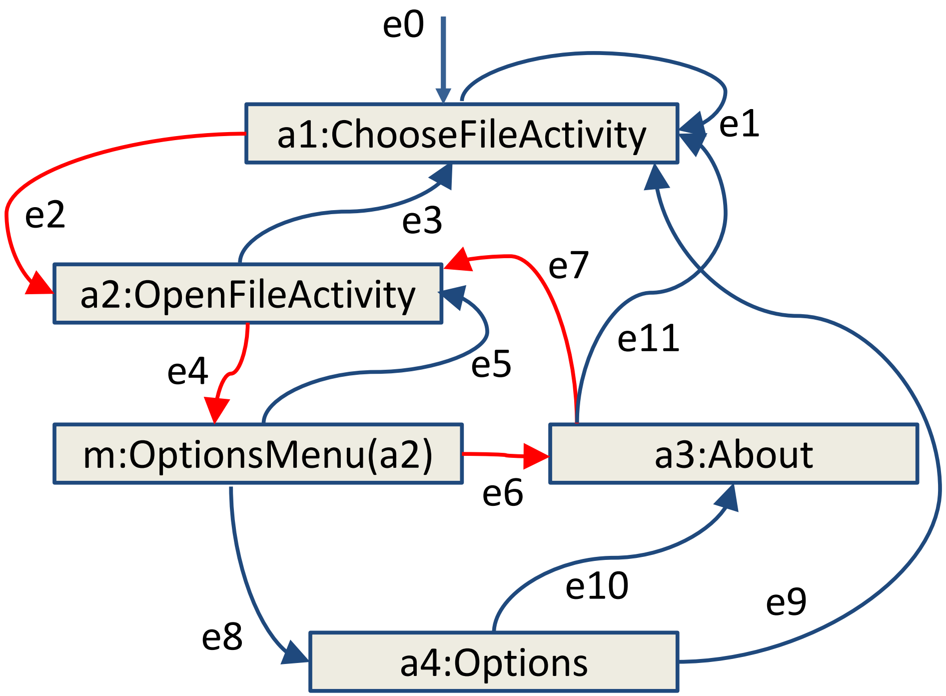
Path Validity



e0: launch, push(a1)
e1: item, click, —
e2: item, click, push(a2)
e3: back, pop(a2)
e4: menu, push(m)
e5: back, pop(m)
e6: aboutItem, click, pop(m) push(a3)
e8: optionsItem, click, pop(m) pop(a2) push(a4)
e7: back, pop(a3)
e9: back, pop(a4)
e10: btn, click, pop(a4) push(a3)
e11: back, pop(a3)

Invalid path: **push(a2) push(m) pop(m) push(a3) pop(a3)**
The top of the stack should be **a2**, but the last node on the path is **a1**

Path Validity

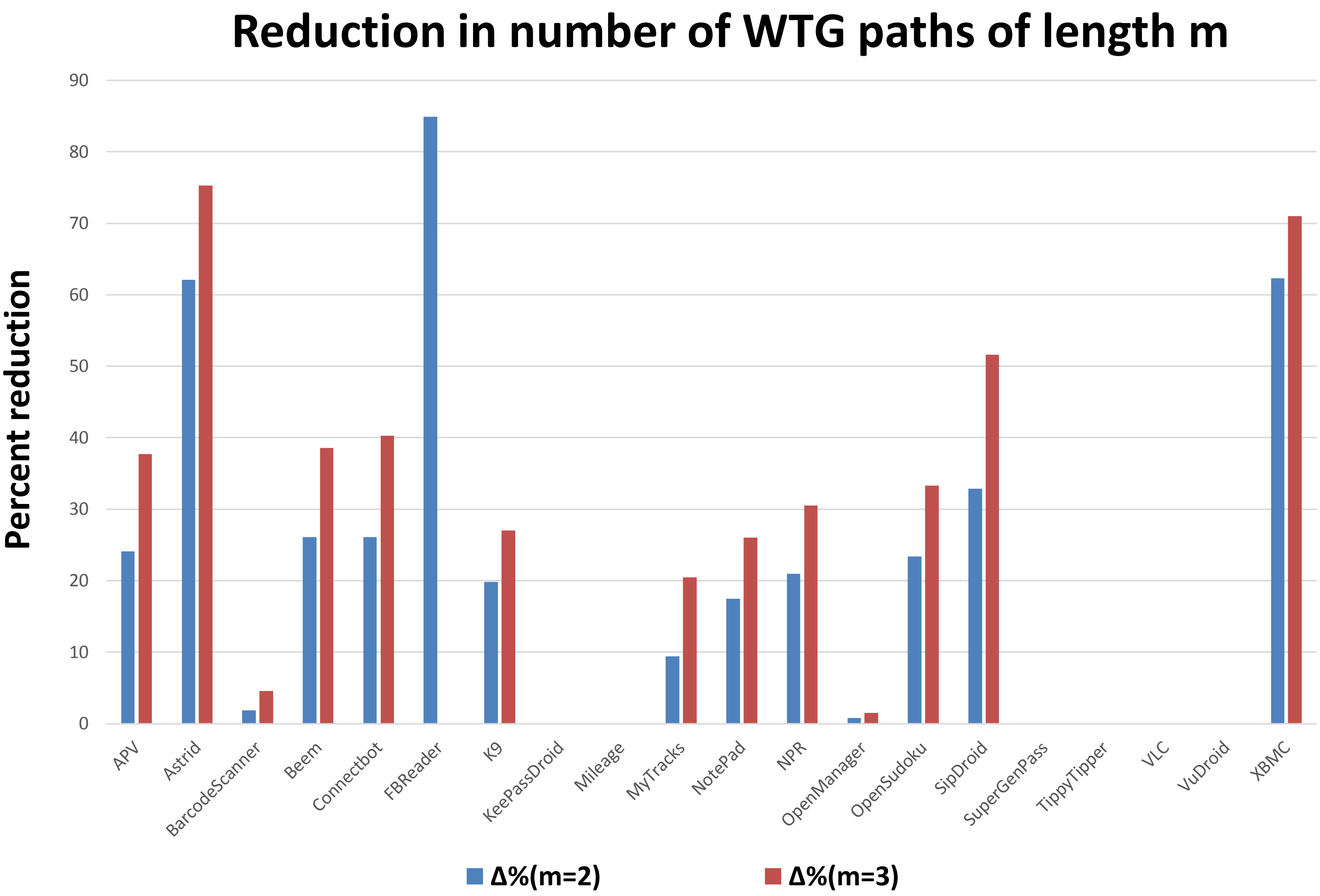


e0: launch, push(a1)
e1: item, click, —
e2: item, click, push(a2)
e3: back, pop(a2)
e4: menu, push(m)
e5: back, pop(m)
e6: aboutItem, click, pop(m) push(a3)
e8: optionsItem, click, pop(m) pop(a2) push(a4)
e7: back, pop(a3)
e9: back, pop(a4)
e10: btn, click, pop(a4) push(a3)
e11: back, pop(a3)

Valid path: **push(a2) push(m) pop(m) push(a3) pop(a3)**

The top of the stack should be **a2**, and indeed the last node on the path is **a2**

Importance of Path Validity Check



Take-Home Messages

Weak foundations for static control-flow and data-flow analysis for Android GUIs

- Progress in the last few years [CGO14][ICSE15][AST15][PhD14][PhD15]
- Many open problems [SOAP16]

Useful GUI models built via static analysis

- Static analysis of resource leaks [CC16]
- Automated test generation [AST16][AST18]
- Responsiveness profiling [MobileSoft17]

Interesting problems beyond plain Android

- GUI analysis and testing for Android Wear [ICSE17]

Take-Home Messages

Weak foundations for static control-flow and data-flow analysis for Android GUIs

- Progress in the last few years [CGO14][ICSE15][AST15][PhD14][PhD15]
- Many open problems [SOAP16]

Useful GUI models built via static analysis

- Static analysis of resource leaks [CC16]
- Automated test generation [AST16][AST18]
- Responsiveness profiling [MobileSoft17]

Interesting problems beyond plain Android

- GUI analysis and testing for Android Wear [ICSE17]

Resource Leak Detection

Resource leaks can drain the battery

- Mismanagement of energy-intensive resources such as the GPS and hardware sensors

Leak patterns

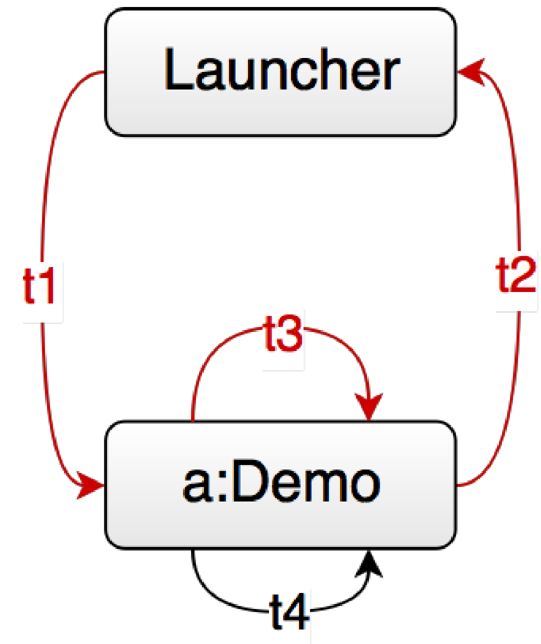
- Defined two patterns of run-time behavior in Android GUIs that can cause energy leaks

Algorithms for static detection

- Designed a static control-flow and data-flow analysis to detect potential leaks

GPS Leak

```
class Demo extends Activity {
  void onCreate() { … }
  void onResume() {
    Button b = ...;
    OnClickListener l = new OnClickListener() {
      void onClick(View v) {
        Manager.instance.registerListeners(); } };
    b.setOnClickListener(l); }
  void onDestroy() { … }
}

class Manager implements LocationListener {
  static Manager instance = new Manager();
  void registerListeners() {
      LocationManager lm = ...;
      lm.requestLocationUpdates(this); } }
```



Defect sequence:
t1, t3, t2
onCreate(a), onResume(a), onClick(b), onDestroy(a)

Leak Patterns

Pattern 1: Lifetime containment

- An activity w acquires an energy-intensive resource but does not release it by the time w is destroyed
- $T = \langle t_1, t_2 \dots t_n \rangle$
  - t_1 triggers $push(w)$ and t_n triggers $pop(w)$
  - push/pop sequence between the two is balanced
  - callbacks along T acquire an energy-intensive resource but do not release it

Pattern 2: Long-wait state

- An activity w acquires an energy-intensive resource and enters a long-wait state without releasing the resource

Static Detection

Callbacks $[c_1, o_1], [c_2, o_2] \dots [c_m, o_m]$ **along a path**

- For c_i invoked with context o_i: compute set A_i of acquired resources and set R_i of released resources
- Need constant propagation and several traversals of c_i's control-flow graph

Leak: if a resource is in A_i but not in $R_{i+1}, \dots, R_m$

Evaluation and Conclusions

Compared with prior work on dynamic leak detection [Liu et al. TSE 2014]

- All GUI-based defects discovered by that prior work were also discovered by our static analysis
- 3 new defects found

Precision

- 17 defects reported; 16 validated on a physical device
- Only 1 false positive, but arguably still a problem

Static resource leak detection in Android GUIs is feasible and precise

Take-Home Messages

Weak foundations for static control-flow and data-flow analysis for Android GUIs

– Progress in the last few years [CGO14][ICSE15][AST15][PhD14][PhD15]

– Many open problems [SOAP16]

Useful GUI models built via static analysis

– Static analysis of resource leaks [CC16]

– Automated test generation [AST16][AST18]

– Responsiveness profiling [MobileSoft17]

Interesting problems beyond plain Android

– GUI analysis and testing for Android Wear [ICSE17]

Take-Home Messages

Weak foundations for static control-flow and data-flow analysis for Android GUIs

– Progress in the last few years [CGO14][ICSE15][AST15][PhD14][PhD15]

– Many open problems [SOAP16]

Useful GUI models built via static analysis

– Static analysis of resource leaks [CC16]

– Automated test generation [AST16][AST18]

– Responsiveness profiling [MobileSoft17]

Interesting problems beyond plain Android

– GUI analysis and testing for Android Wear [ICSE17]

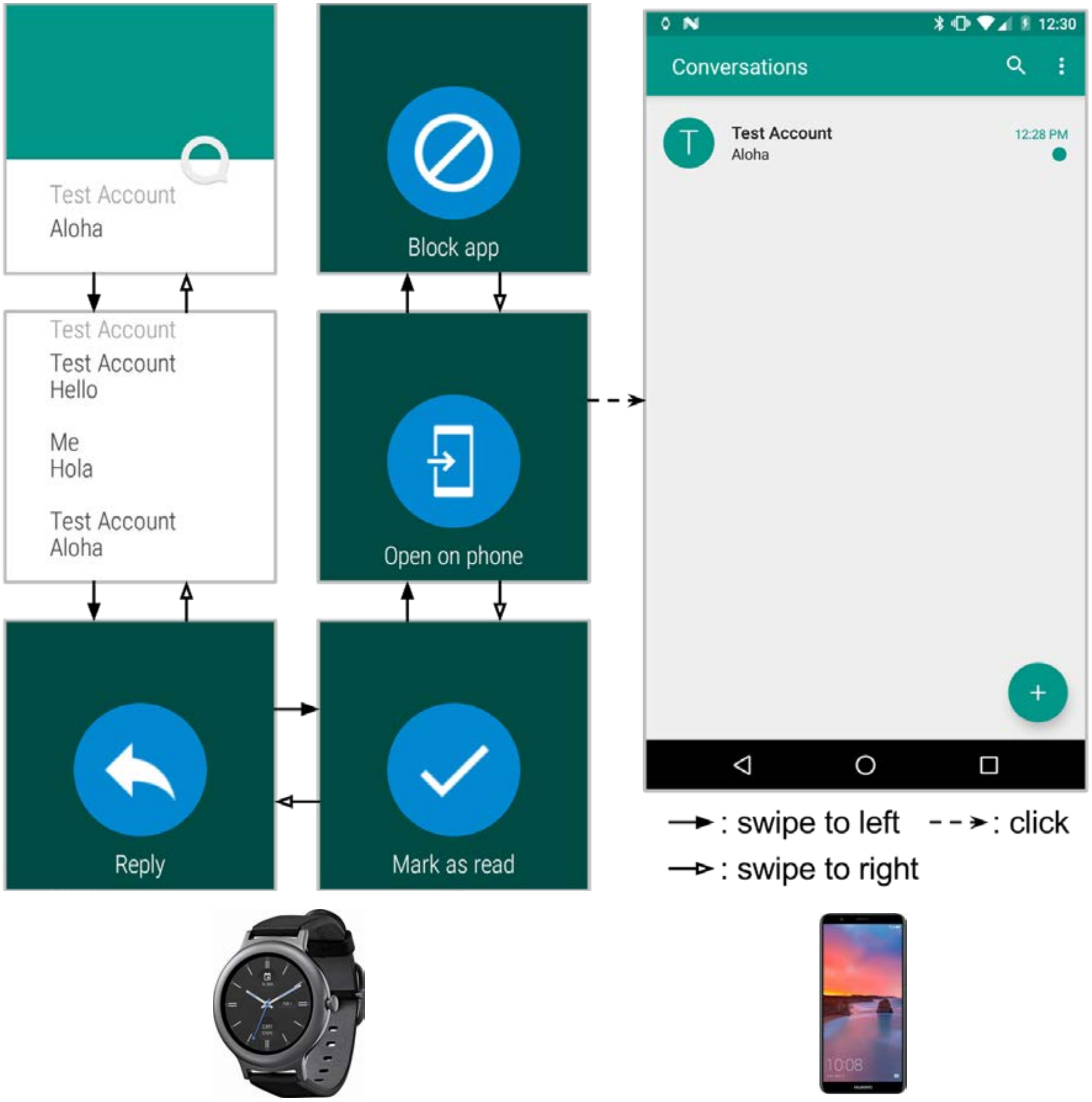
Background

Android Wear (AW)

- Google's platform for wearable devices (e.g., smartwatches)
- AW apps can run independently, or in conjunction with companion app in the handheld device

Open problem: notifications that are issued on the handheld but displayed on the wearable GUI

GUI Example

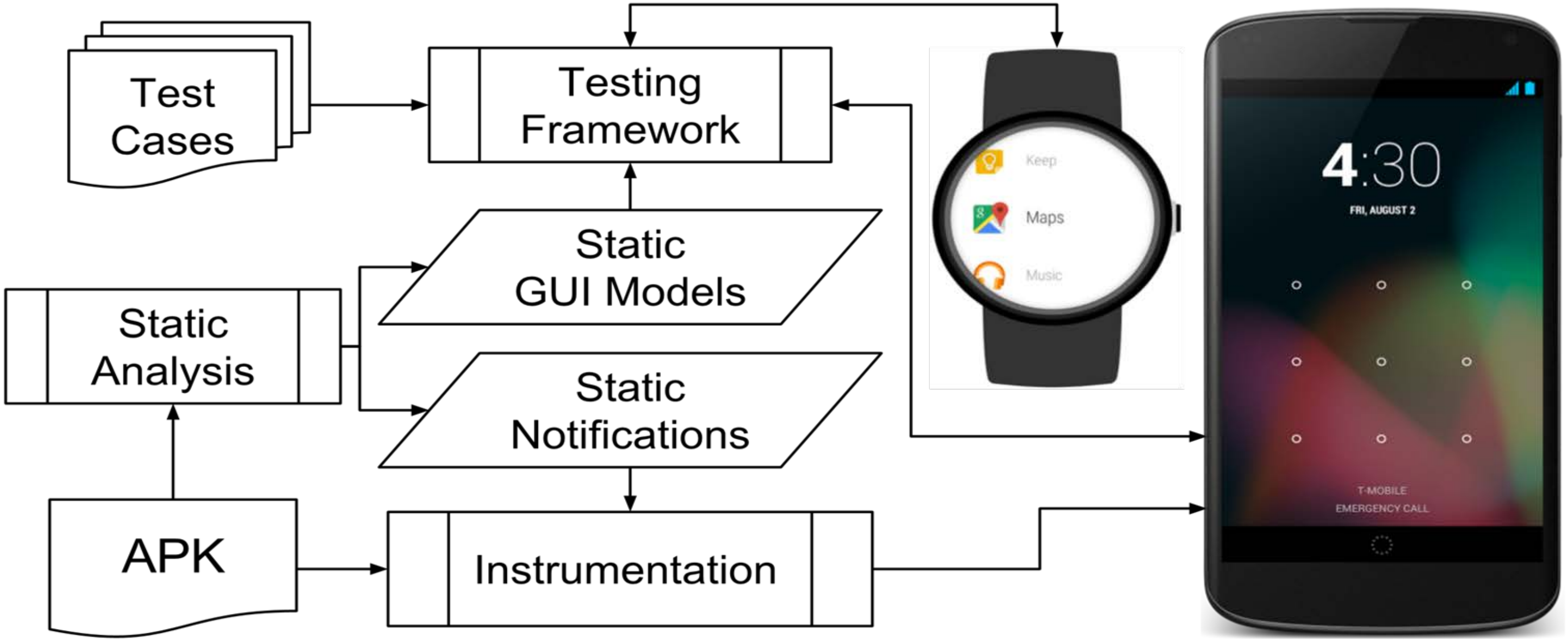


Test Account
Aloha
Block app
Test Account
Test Account
Hello
Me
Hola
Test Account
Aloha
Open on phone
Reply
Mark as read
12:30
Conversations
Test Account
Aloha
12:28 PM
T
: swipe to left
: click
: swipe to right

Abstractions for Static Analysis

x = addaction(y,z)	Add actions to wearable extender
x = setaction(y,z)	Set intent (for “Open on phone” action)
x = extend(y,z)	Attach wearable extender to notification builder
x = build(y)	Build notification from a notification builder
notify(x)	Issue notification
x = buildpending(y)	Wrap intent into pending intent
x = buildaction(y)	Building action from pending intent
x = addpage(y,z)	Add notification as page to another notification

Testing Tool



Testing framework

- **AW UIAutomator**: communicate with handheld and wearable
- **GUI crawler**: record GUI elements on the wearable, to check coverage

Instrumentation

- Insert & record IDs for GUI elements

What Next?

Stronger static analysis foundations

– Semantics: inference, validation, evolution

More uses of static GUI analysis

– Automated code rewriting for better performance
– …

Beyond Android phones and tablets

– Android Wear, Android Things, Android Auto
– Short-term: standalone Android Wear apps