Static analysis of Android programs

Étienne Payet
Fausto Spoto

Information and Software Technology 54 (2012) 1192–1201

Salih Safa BACANLI

Abstract

- Our goal is to extend the Julia static analyzer, based on abstract interpretation, to perform formally correct analyses of Android programs. This article is an in depth description of such an extension,of the difficulties that we faced and of the results that we obtained.

- We have extended the class analysis of the Julia analyzer, which lies at the heart of many other analyses, by considering some Android key specific features

- Classcast, dead code, nullness and termination analysis are done.
- Formally correct results in at most 7 min and on standard hardware.
- As a language, Android is Java with an extended library for mobile and interactive applications, hence based on an event-driven architecture. (WRONG)

Introduction

- Klocwork is based on *Syntactical* checks.
- If no applicable pattern is found, bugless (!)

- Julia has AI.
- *Semantic* Checks.
- If no bugs found, the code is bugless. (?)

Julia fundamentals

- Julia analyzes Java bytecode. Dalvik different.
- Event Handlers can be seen as dead code. *actionPerformed* is problematic. (?)

Android Structure

- Activities (code interacting with user through a visual interface),
- Services (background operations with no interaction with the user),
- Content providers (DB)
- broadcast receivers (objects reacting to broadcast messages).
- Event handlers
- XML manifest file components of an application.
- XMLfiles describe the visual layoutof the activities

```java
public class LunarLander extends Activity {
  private LunarView mLunarView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // tell system to use the layout defined in our XML file
    setContentView(R.layout.lunar_layout);
    // get handles to the LunarView from XML
    mLunarView = (LunarView) findViewById(R.id.lunar);
    // give the LunarView a handle to a TextView
    mLunarView.setTextView((TextView) findViewById(R.id.text));
  }
}
```

Fig. 1. A portion of the source code Android file `LunarLander.java`.

```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent">
 <com.example.android.lunarlander.LunarView android:id="@+id/lunar"
   android:layout_width="match_parent" android:layout_height="match_parent"/>
 <RelativeLayout
   android:layout_width="match_parent" android:layout_height="match_parent" >
  <TextView android:id="@+id/text"
    android:text="@string/lunar_layout_text_text"
    android:visibility="visible"
    android:layout_width="wrap_content"  android:layout_height="wrap_content"
    android:layout_centerInParent="true" android:gravity="center_horizontal"
    android:textColor="#88ffffff"        android:textSize="24sp"/>
 </RelativeLayout>
</FrameLayout>
```

Fig. 2. The XML layout file `lunar_layout.xml`.

✓Checks

- Equality (equals vs ==)
- The use of both kinds of checks on the same class type is hence a symptom of a potential bug(?)(if AND ed no problem)
- Static update
- The modification of a static field from inside a constructor or an instance method is legal but a symptom of a possible bug or, at least, of bad programming style. For this reason, we check when that situation occurs.

- Dead Code Check
  - Already done by javac. not possible in bytecode!
- Method redefinition check
  - already done by javac. not possible in bytecode!
- Hashcode and Equals override
  - hashcode in Lists...
- Nullness Check
  - how to avoid NullPointerException?

- Termination
  - Halting problem?
- international competition of termination analysis for Java bytecode on July 2010
- Classcast
  - checked by Eclipse not javac (Possible in bytecode)

- Julia does these on bytecode.
- Eclipse in source code.
- Why not doing the checks in compile time rather than doing them after compilation?

```
public class LunarLander extends Activity {
  private LunarView mLunarView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // tell system to use the layout defined in our XML file
    setContentView(R.layout.lunar_layout);
    // get handles to the LunarView from XML
    mLunarView = (LunarView) findViewById(R.id.lunar);
    // give the LunarView a handle to a TextView
    mLunarView.setTextView((TextView) findViewById(R.id.text));
  }
}
```

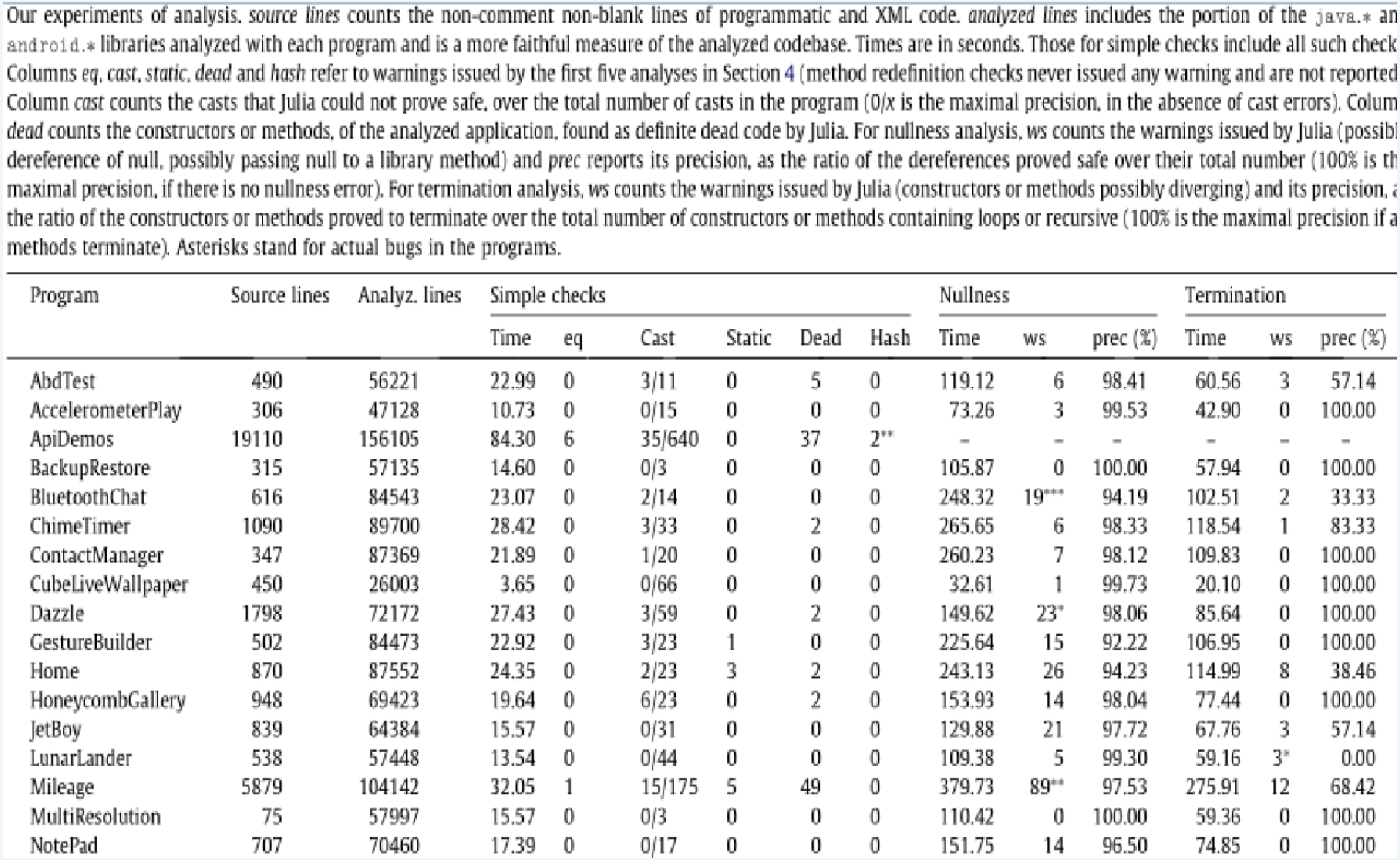
Fig. 1. A portion of the source code Android file `LunarLander.java`.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent">
 <com.example.android.lunarlander.LunarView android:id="@+id/lunar"
   android:layout_width="match_parent" android:layout_height="match_parent"/>
 <RelativeLayout
   android:layout_width="match_parent" android:layout_height="match_parent" >
  <TextView android:id="@+id/text"
    android:text="@string/lunar_layout_text_text"
    android:visibility="visible"
    android:layout_width="wrap_content"  android:layout_height="wrap_content"
    android:layout_centerInParent="true" android:gravity="center_horizontal"
    android:textColor="#88ffffff"        android:textSize="24sp"/>
 </RelativeLayout>
</FrameLayout>
```

Fig. 2. The XML layout file `lunar_layout.xml`.

Experiment Results

- *We have manually checked all the warnings in Table 1. Most of them look to us as false alarms, but a definite answer is difficult, since we are not the authors of those programs*
- *The most precise analysis is an analysis that reports only the actual nullness bugs and no false alarm. This means that its precision (according to our metrics) is 100% if there is no actual nullness bugs and slightly below 100% otherwise.*

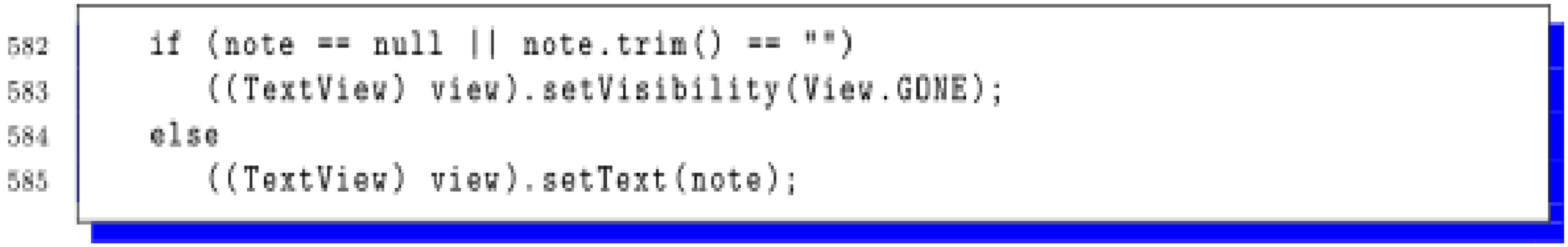


Our experiments of analysis. *source lines* counts the non-comment non-blank lines of programmatic and XML code. *analyzed lines* includes the portion of the `java.*` an `android.*` libraries analyzed with each program and is a more faithful measure of the analyzed codebase. Times are in seconds. Those for simple checks include all such check Columns *eq*, *cast*, *static*, *dead* and *hash* refer to warnings issued by the first five analyses in Section 4 (method redefinition checks never issued any warning and are not reported Column *cast* counts the casts that Julia could not prove safe, over the total number of casts in the program (0/*x* is the maximal precision, in the absence of cast errors). Colum *dead* counts the constructors or methods, of the analyzed application, found as definite dead code by Julia. For nullness analysis, *ws* counts the warnings issued by Julia (possibl dereference of null, possibly passing null to a library method) and *prec* reports its precision, as the ratio of the dereferences proved safe over their total number (100% is th maximal precision, if there is no nullness error). For termination analysis, *ws* counts the warnings issued by Julia (constructors or methods possibly diverging) and its precision, a the ratio of the constructors or methods proved to terminate over the total number of constructors or methods containing loops or recursive (100% is the maximal precision if a methods terminate). Asterisks stand for actual bugs in the programs.

Program	Source lines	Analyz. lines	Simple checks						Nullness			Termination		
			Time	eq	Cast	Static	Dead	Hash	Time	ws	prec (%)	Time	ws	prec (%)
AbdTest	490	56221	22.99	0	3/11	0	5	0	119.12	6	98.41	60.56	3	57.14
AccelerometerPlay	306	47128	10.73	0	0/15	0	0	0	73.26	3	99.53	42.90	0	100.00
ApiDemos	19110	156105	84.30	6	35/640	0	37	2**	–	–	–	–	–	–
BackupRestore	315	57135	14.60	0	0/3	0	0	0	105.87	0	100.00	57.94	0	100.00
BluetoothChat	616	84543	23.07	0	2/14	0	0	0	248.32	19***	94.19	102.51	2	33.33
ChimeTimer	1090	89700	28.42	0	3/33	0	2	0	265.65	6	98.33	118.54	1	83.33
ContactManager	347	87369	21.89	0	1/20	0	0	0	260.23	7	98.12	109.83	0	100.00
CubeLiveWallpaper	450	26003	3.65	0	0/66	0	0	0	32.61	1	99.73	20.10	0	100.00
Dazzle	1798	72172	27.43	0	3/59	0	2	0	149.62	23*	98.06	85.64	0	100.00
GestureBuilder	502	84473	22.92	0	3/23	1	0	0	225.64	15	92.22	106.95	0	100.00
Home	870	87552	24.35	0	2/23	3	2	0	243.13	26	94.23	114.99	8	38.46
HoneycombGallery	948	69423	19.64	0	6/23	0	2	0	153.93	14	98.04	77.44	0	100.00
JetBoy	839	64384	15.57	0	0/31	0	0	0	129.88	21	97.72	67.76	3	57.14
LunarLander	538	57448	13.54	0	0/44	0	0	0	109.38	5	99.30	59.16	3*	0.00
Mileage	5879	104142	32.05	1	15/175	5	49	0	379.73	89**	97.53	275.91	12	68.42
MultiResolution	75	57997	15.57	0	0/3	0	0	0	110.42	0	100.00	59.36	0	100.00
NotePad	707	70460	17.39	0	0/17	0	0	0	151.75	14	96.50	74.85	0	100.00

Simple Checks-Open Sudoku

- Use of *note.trim() ==""*
  - *Can be buggy-compile time (equals better)*
- not overriding hash function
  - Can make lists buggy.
  - ıf no list then fine (not specified)

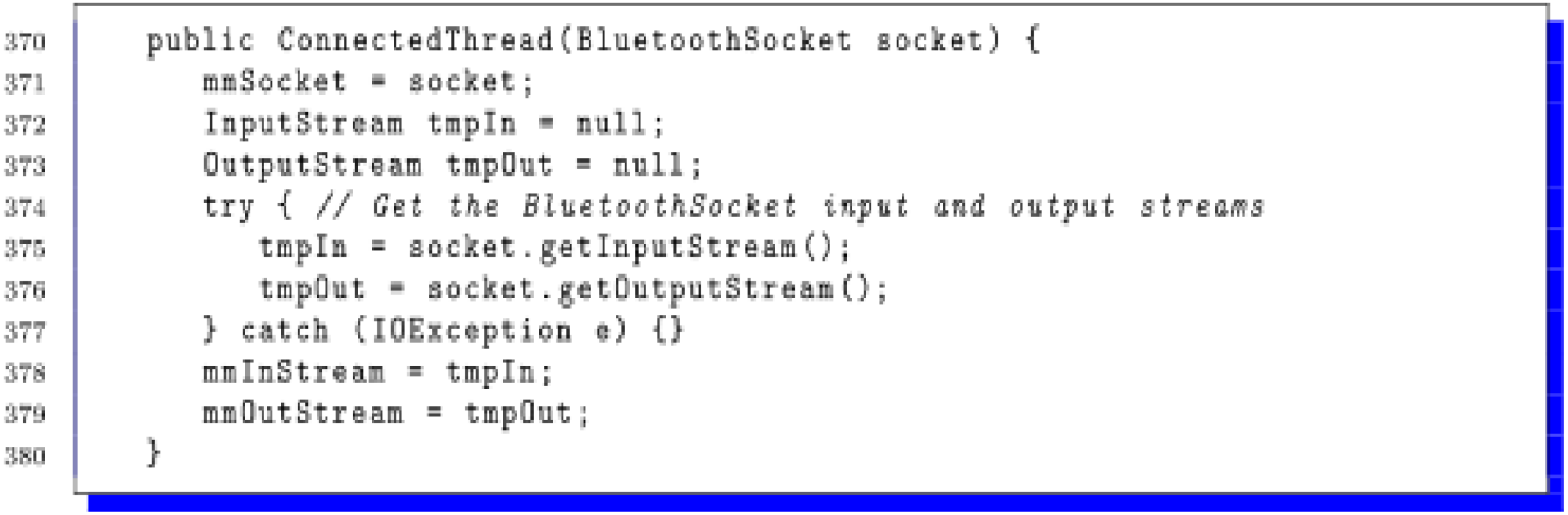


```
if (note == null || note.trim() == "")
   ((TextView) view).setVisibility(View.GONE);
else
   ((TextView) view).setText(note);
```

Fig. 3. A portion of method `setViewValue` defined in OpenSudoku.

Nullness Check

- If there is no bluetooth device, the objects will become null and there is no check for that. No exception handling!!!!

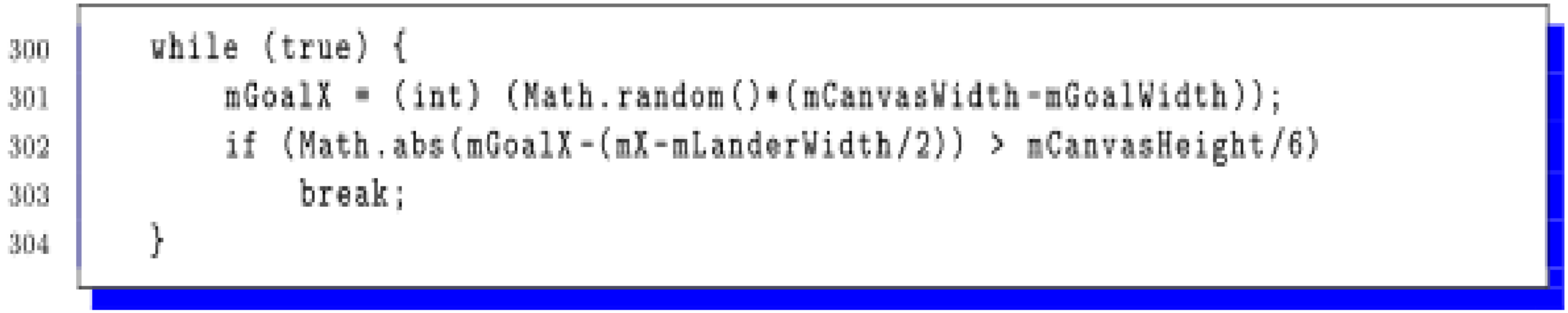


```
public ConnectedThread(BluetoothSocket socket) {
   mmSocket = socket;
   InputStream tmpIn = null;
   OutputStream tmpOut = null;
   try { // Get the BluetoothSocket input and output streams
      tmpIn = socket.getInputStream();
      tmpOut = socket.getOutputStream();
   } catch (IOException e) {}
   mmInStream = tmpIn;
   mmOutStream = tmpOut;
}
```

Fig. 6. The constructor of class `ConnectedThread` in BluetoothChat.

Termination Check

- *Most warnings issued by Julia about possibly diverging methods are false alarms.*



```
while (true) {
    mGoalX = (int) (Math.random()*(mCanvasWidth-mGoalWidth));
    if (Math.abs(mGoalX-(mX-mLanderWidth/2)) > mCanvasHeight/6)
        break;
}
```

Fig. 7. A portion of method `doStart` defined in LunarLander.

Conclusion

- Can check software in minutes with standard hardware
- Array of references are problematic as expected. Everything is pointer in Java!
- The size of the analyzed code is also problematic. For instance, we could not perform the nullness and termination analyses of ApiDemos
- GWT and Play applications in future!

ANY QUESTIONS ?