Analyzing Android Applications with Abstract Interpretation
Phillip Mates

Motivation
We aim to use abstract interpretation to provide a static analysis for verifying fine-grained application permissions on the Android platform.

The current Android permission model offers coarse, per-application specifications. This enables an app’s sub-programs to abuse all the permissions available to the greater host application:

- An advertisement library could access location APIs in a GPS App.
- A malicious user-plugin interpreter embedded in an application could allow arbitrary API usage at run-time.

Method
1. Build a concrete CESK interpreter for Dalvik bytecode.
2. Turn concrete into abstract by removing infinite bytecode.
   **Abstract Domains**: abstract value spaces as finite lattices. We use a flat domains: ‘string’, ‘number’, etc.
   **Addresses**: restrict address allocation to be finite, make the store map addresses to sets of values, and use joins in place of strong updates.
   **Frame Pointer & Time**: let time be the last k statements and use the current time for frame pointer allocation.

Background
**Dalvik bytecode** is a register-based variant of the Java bytecode used by the Android platform.

```
(new-instance v0 java/lang/StringBuilder)
(invoke-virtual {v0 v1} java/lang/StringBuilder/append [object java/lang/String])
```

**Abstract interpretation** is a sound, terminating approximation of a program’s concrete interpretation.

**CESK machine** is an abstract state machine consisting of Control, Environment, Store and Kontinuation components.

\[
\zeta \in \Sigma = \text{Stmts} \times \text{FramePointer} \times \text{Store} \times \text{Kont} \times \text{Time} \\
fp \in \text{FramePointer} \text{ an infinite set} \\
\sigma \in \text{Store} \rightarrow \text{fin Value} \\
\kappa \in \text{Kont} = \text{fin}(\text{Stmts, fp, a}_0) | \text{halt} \\
a \in \text{Addr} = \text{RegAddr} | \text{HeapAddr} | \text{KontAddr} \\
a_0, a_\kappa \text{ are elements in an infinite set} \\
t \in \text{Time} \text{ an infinite set} \\
alloc : \Sigma \rightarrow \text{FramePointer} \\
tick : \Sigma \rightarrow \text{Time}
\]

```
abstract 1-CFA visited states graph of Fibonacci
```

```
Control flow graph of Fibonacci
```

Results
```
(method (attrs public static) fib(int)int 
 ; parameter[0] : v2 (int) 
 (const/4 v0 1) 
 (if-gt v2 v0 l8c4) 
 (label l8c2) 
 (return v2) 
 (label l8c4) 
 (add-int/lit8 v0 v2 255) 
 (invoke-static {v0} org/ucombinator/FibonacciApp/fib int) 
 (move-result v0) 
 (add-int/lit8 v1 v2 254) 
 (invoke-static {v1} org/ucombinator/FibonacciApp/fib int) 
 (move-result v1) 
 (add-int v2 v0 v1) 
 (goto l8c2)
```