Automated model-based android GUI testing using multi-level GUI Comparison Criteria

happyBiker, haxxorrman

Flatulenza, University of Prone

homemadebike.9999999@stud.unipronax.fritt, lexusGTsportage.99999999@stud.unipronax.fritt

January 24, 2017

Overview

GUI testing automatico(1)

Perché dedicare particolare attenzione alle applicazioni Android?

Attualmente le applicazioni android comprendono quasi il 90% del mercato di software mobile.

Cos'è il GUI testing?

È una metodologia di testing che esplora una parte dello spazio degli stati del programma testato dando gli input al programma tramite l'interfaccia grafica.

Perché automatizzare il testing?

Il motivo principale è il costo, l'esecuzione manuale dei test ha il costo eccessivo per molte applicazioni. Inoltre l'esecuzione manuale dei test richiede tanto tempo e a volte non è in grado di individuare tutti gli errori.

GUI testing automatico(2)

Attualmente sul mercato si trovano dei tool per il GUI testing automatico che generano gli input in modo alleatorio, tentando di generare una sequenza di input che provoca il malfunzionamento del software. Nel testo viene citato un tool chiamato *Android Monkey* che permette questo tipo di test.

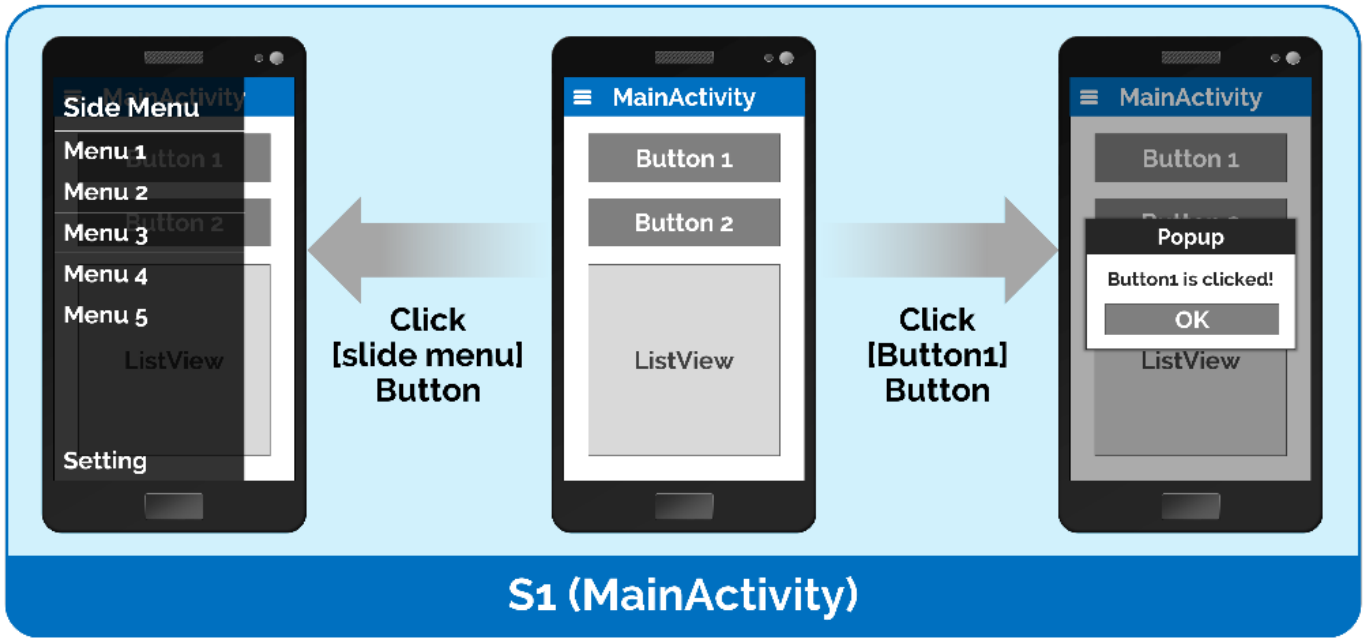
Dato che si tratta essenzialmente di una ricerca locale nello spazio degli stati, questi metodi sono soggetti alle problematiche degli algoritmi di ricerca locale. Per esempio possono valutare lo stesso stato più volte.

L'approccio proposto dagli autori è di costruire un grafo degli stati della GUI, utilizzando dei criteri di equivalenza (GUICC) per distinguere gli stati.

Modello GUI

Un altro approccio prevede l'uso del modello della GUI. Il modello GUI è un automa a stati finiti, dove gli stati dell'automa corrispondono agli stati dell'interfaccia grafica. L'automa transisce da uno stato all'altro al verificarsi di un evento. Il modello GUI viene rappresentrato da un multigrafo dove i nodi corrispondono agli stati e gli archi corrispondono alle transizioni. Ogni arco è etichettato dall'evento che provoca la transizione. Il concetto fondamentale necessario per la costruzione di questo grafo è il criterio di equivalenza degli stati.

Le applicazioni Android spesso generano parte dell'interfaccia grafica in modo dinamico. Per modellare correttamente la GUI biogna tener conto delle componenti dinamiche.



Un criterio d'equivalenza debole come "Nome Activity" in questo caso non sarebbe in grado di accorgersi della differenza tra gli stati.

Multi-Level GUI Comparison Criteria (Multi-Level GUICC)

La struttura a livelli definita dagli autori del paper è composta da:

- **Nome pacchetto**. Quando l'activity corrente non appartiene allo stesso pacchetto dell'applicazione testata significa che l'applicazione testata è stata terminata oppure ha invocato un altra applicazione, in ogni caso lo stato corrente è considerato lo stato terminale.
- **Nome Activity**. Quando l'activity corrente ha un nome diverso da quella precedente si tratta di uno stato diverso.
- **Composizione Widget non-eseguibili**
- **Composizione Widget eseguibili**
- **Contenuti**

Generazione del modello

Dato che le applicazioni Android in genere non contengono un modello della GUI costruito dallo sviluppatore è stato sviluppato un metodo per dedurre il modello (*Model learning*) da un applicazione tramite *reverse engineering*(*GUI Ripping*).

Generazione del modello(2)

Per costruire il grafo della GUI vengono eseguiti iterativamente questi passaggi:

- Generare un evento amissibile nello stato corrente.
- Verificare tramite i criteri d'equivalenza se lo stato della GUI è cambiato. In caso affermativo ci sono tre casi:
  - Il nuovo stato è già presente come nodo nel grafo. Allora si aggiunge un arco dallo stato precedente al nuovo stato etichettato dall'evento che porta in questo stato.
  - Non esiste un nodo corrispondente, allora abbiamo scoperto uno stato nuovo. Viene generato un nuovo nodo e generato l'arco come nel primo caso.
  - Lo stato risultato è uno stato terminale, allora l'applicazione è terminata in seguito all'evento.(Terminazione normale o anormale dovuta ad un errore/eccezione).

  In caso contrario si aggiunge un *loop*(un arco che origina e termina nello stesso nodo) etichettato dall'evento.

Architettura del sistema

Motore di testing

Viene eseguito lato controllore.

- **Esecutore test** Sceglie l'input da mandare al software testato secondo un algoritmo(BFS).
- **Generatore del grafo** descritto sopra.
- **Error checker** Controlla la presenza di errori.

Strato di comunicazione

Serve per trasferire informazioni fra la macchina controllore e il dispositivo Android che esegue il software testato.

Event agent

Software eseguito sul dispositivo Android. Riceve messaggi dal contollore e genera input al software testato corrispondenti.

Goal to reach

Osservare come la scelta dei GUICC può influenzare il comportamento di una AUT (Application Under Test).

Sono utili i GUICC ?

Question 1

In che modo i *GUICC* influenzano la generazione di modelli di comportamento delle app Android ?

Question 2

I GUICC influenzano la quantità di codice ottenuto ed eseguito durante la fase di GUI testing ?

Question 3

I GUICC influenzano la capacità di scovare errori durante la fase di GUI testing ?

Question 1

Question 1

In che modo i *GUICC* influenzano la generazione di modelli di comportamento delle app Android ?

Per rispondere hanno preso alcune app, sia open source che proprietarie, e ne hanno generato vari GUI graphs, basandosi per ognuno su un diverso livello di "specificità" dei GUICC .

Nella raccolta dei dati , i grafi generati con C-Lv 1 non sono stati considerati, poichè è un criterio troppo poco discriminante per ottenere risultati validi.

Recall

C-Lv 1 : due ScreenNodes differiscono per il package di appartenenza.

Grafi generati con vari Criteria Levels

No	App name	C-Lv2		C-Lv4		C-Lv5	
		#SN	#EE	#SN	#EE	#SN	#EE
1	*Alogcat*	*5*	*45*	*15*	*247*	*76*	*269*
2	*Anycut*	*8*	*33*	*8*	*33*	*9*	*42*
3	*Mileage*	*16*	*117*	*69*	*532*	*81*	*618*
4	*Sanity*	*1*	*4*	*2*	*7*	*49*	*552*
5	*DalvikExplorer*	*16*	*178*	*30*	*301*	**S/E**	**S/E**
6	*MyLock*	*2*	*24*	*5*	*51*	*10*	*101*
7	*Bequick*	*2*	*7*	*60*	*250*	*71*	*351*
8	*Syncmypix*	*4*	*11*	*20*	*96*	*20*	*115*
9	*TippyTipper*	*4*	*29*	*13*	*102*	*19*	*175*
10	*WhoHasMyStuff*	*7*	*37*	*24*	*143*	*26*	*180*

No	App name	C-Lv2		C-Lv3 ∼ C-Lv5		
		#SN	#EE	C-Lv	#SN	#EE
1	*Google Translate*	*5*	*41*	***4***	***55***	***871***
2	*Advanced Task Killer*	*4*	*27*	***3***	***12***	***178***
3	*Alarm Clock Xtreme Free*	*4*	*81*	***4***	***23***	***363***
4	*GPS Status & Toolbox*	*3*	*12*	***5***	***49***	***592***
5	*Music Folder Player Free*	*7*	*37*	***5***	***30***	***295***
6	*Wifi Matic*	*2*	*9*	***5***	***20***	***160***
7	*VNC Viewer*	*6*	*43*	***5***	***7***	***60***
8	*Unified Remote*	*6*	*94*	***5***	***20***	***160***
9	*Clipper*	*2*	*17*	***5***	***36***	***195***
10	*Lifetime Alarm Clock*	*2*	*5*	***5***	***60***	***529***

Osservazioni

- C-Lvs superiori generano più nodi e più archi ;
- C-Lv 2 è un criterio meno "espressivo" di C-Lv5 ;
- ATTENZIONE: app come X (per la gestione del ...) o come Davik (gestione dei processi) generano stati infiniti e il grafo esplode .

Question 1

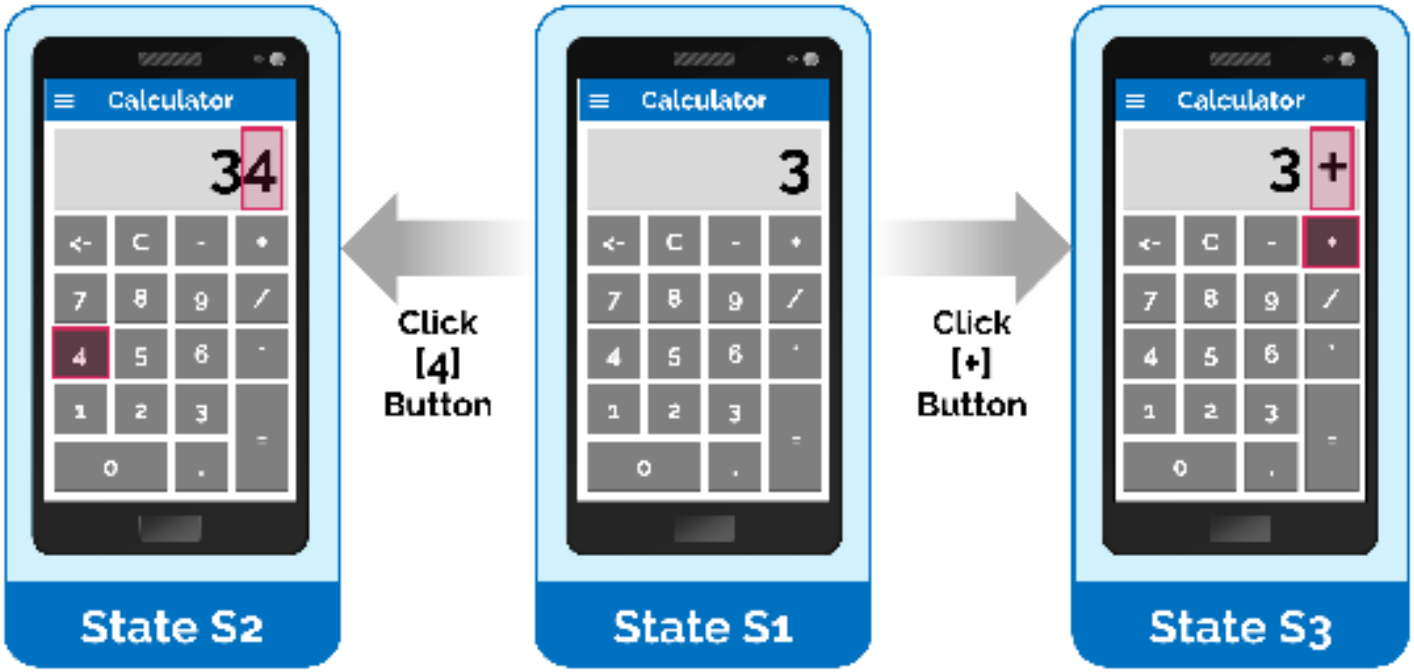
Question 1

In che modo i *GUICC* influenzano la generazione di modelli di comportamento delle app Android ?

Risposta

Criteri più "specializzati" possono rendere il GUI states graph più completo ed esaustivo , ma livelli eccessivamente fine-grained di selezione non migliorano la qualità del grafo .

Una semplice Calcolatrice ...



Una calcolatrice è una app particolare : è indistinguibile per ogni **C-Lv** precedente al quinto

Quando si inizia a comparare **in base al testo inserito** però il numero di stati esplode : ogni nuovo carattere (numero o operatore) inserito corrisponde ad un nuovo ScreenNode.

Wider is not better ...

Testare applicazioni simili a Calculator **è complicato** , poichè più specifico è **Max C-Lv** , più grande è il grafo risultante.

Grafi molto grandi talvolta contengono ScreenNodes diversi da cui possono essere avviati eventi uguali , quindi i test perdono molto tempo su input simili.

Inoltre non è detto che con grafi più grandi si riescano a coprire tutte le funzionalità messe a disposizione da un'app.

Come misurare oggettivamente la quantità di funzionalit testate ?

Per avere una misura oggettiva del numero di funzionalità testate , si può misurare il numero di linee di codice effettivamente eseguite durante un test sul grafo .

In generale , maggiore è il numero di linee di codice eseguite , maggiore è il numero di funzionalità testate.

Question 2

Question 2

I GUICC influenzano la quantità di codice ottenuto ed eseguito durante la fase di GUI testing ?

I test sono stati effettuati sempre sullo stesso insieme di app open source e proprietarie.

Linee di codice eseeguite nelle App testate

No	Package name	Method coverage			Statement coverage		
		A	*C-Lv*	*M*	*A*	*C-Lv*	*M*
1	*Alogcat*	*46%*	*4*	***65%***	*39%*	***5***	***56%***
2	*Anycut*	*23%*	*4*	***69%***	*19%*	*4*	***55%***
3	*Mileage*	*22%*	***5***	***43%***	*18%*	***5***	***33%***
4	*Sanity*	*n/a*					
5	*DalvikExplorer*	*65%*	*4*	***70%***	*57%*	*4*	***64%***
6	*MyLock*	*11%*	*4*	***12%***	*10%*	*4*	***11%***
7	*Bequick*	*24%*	***5***	***39%***	*21%*	***5***	***39%***
8	*Syncmypix*	*10%*	*4*	***24%***	*6%*	*4*	***17%***
9	*TippyTipper*	*42%*	***5***	***65%***	*36%*	***5***	***61%***
10	*WhoHasMyStuff*	*39%*	***5***	***62%***	*35%*	*4*	***51%***
Average		**31%**	**50%**		**27%**	**43%**	

Osservazioni

- le colonne "A" contengono la percentuale di codice eseguito durante un test con Max C-Lv pari a 2 , ossia test eseguiti con il criterio degli activity name;
- le colonne "C-Lv" e "M" vanno lette insieme : "C-Lv" indica il livello a cui viene raggiunta la massima percentuale di codice eseguito "M";
- ci sono alcune eccezioni , dovute però a specifici casi particolari;
- si nota chiaramente che livelli più specializzati di GUICC coprono una maggior percentuale di codice eseguito

Question 2

Question 2

I GUICC influenzano la quantità di codice ottenuto ed eseguito durante la fase di GUI testing ?

Risposta

Il livello di GUICC influenza in modo significativo la percentuale di codice eseguito .

Question 3

Question 3

I GUICC influenzano la capacità di scovare errori durante la fase di GUI testing ?

Runtime errors

No	C-Lv	App name	Error type
1	C-Lv5	*Mileage*	Fatal error (Fatal signal 11 <F/libc>)
2		*Sanity*	Fatal exception (RuntimeException <E/AndroidRuntime>)
3		*Sanity*	Fatal exception (RuntimeException <E/AndroidRuntime>)
4	C-Lv4	*Mileage*	Fatal error (Fatal signal 11 <F/libc>)

- 4 tipi di errori fondamentali ;
- tutti rilevati con grafi "fine-grained" , ossia con criteri di comparazione a basso livello (C-Lv 4/5) .

Question 3

Question 3

I GUICC influenzano la capacità di scovare errori durante la fase di GUI testing ?

Risposta

Oltre a influenzare la quantità di codice eseguito , il livello di GUICC influenza anche molto la capacità di "error detection" del test.