Detection of Java Basic Thread Misuses Based on Static Event Analysis

Baoquan Cui, Miaomiao Wang, Chi Zhang, Jiwei Yan, Jun Yan and Jian Zhang

Email: cuibq@ios.ac.cn
Institute of Software, Chinese Academy of Sciences (ISCAS)
University of Chinese Academy of Sciences (UCAS)

ASE 2023 Presented by Baoquan Cui (崔保全) Sep 13

Typical Error-prone Example of Thread (1)

```
class  Destructible {
        // resource release
        void destroy(){…}
}
```

```
class  T extends Thread{
        Destructible  d;
        void set(Destructible d){ this. d = d}
        void run(){…}      }
```

```
void main(…){//start the thread and try to destroy the object d
        T t = new T();
        Destructible d = new Destructible();
        t.set(d);
        t.start();
        d.destroy();
}
```

- An object ***d*** may destroy itself
- But active thread ***t*** will block the release of the object ***d***
  - As the thread ***t*** has a strong reference of its field ***d***
- A memory leak happens (until the thread finishes)
- Called Hard To Release (HTR) in this paper

Typical Error-prone Example of Thread (2)

```
class  T extends Thread{
        void run(){
                int i = 0;
                while(i<1000000){
                        // task
                }
        }
}
```

```
// start the thread and try to interrupt it
void main(…){
        Thread t = new T();
        t.start();
        …
        t.interrupt();
}
```

- The interruption will lose response
  - As a running thread will only set the interrupted status.
- The thread will continue executing. (Unnecessary processor usage and time wasting)
- Called Interrupt NoResponding (INR) in this paper

A Real Execution of Program with INR

// start the thread and interrupt it immediately

// task: print number

```
public static void main(String[] args) {
    T t = new T();
    t.start();
    t.interrupt();
}
static class T extends Thread {
    public void run() {
        try {
            int i = 0;
            while (i < 10000) {
                i++;
                System.out.print(i);
                System.out.print(" ");
                if (i % 10 == 0) {
                    System.out.println();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

//output

```
9841 9842 9843 9844 9845 9846 9847 9848 9849 9850
9851 9852 9853 9854 9855 9856 9857 9858 9859 9860
9861 9862 9863 9864 9865 9866 9867 9868 9869 9870
9871 9872 9873 9874 9875 9876 9877 9878 9879 9880
9881 9882 9883 9884 9885 9886 9887 9888 9889 9890
9891 9892 9893 9894 9895 9896 9897 9898 9899 9900
9901 9902 9903 9904 9905 9906 9907 9908 9909 9910
9911 9912 9913 9914 9915 9916 9917 9918 9919 9920
9921 9922 9923 9924 9925 9926 9927 9928 9929 9930
9931 9932 9933 9934 9935 9936 9937 9938 9939 9940
9941 9942 9943 9944 9945 9946 9947 9948 9949 9950
9951 9952 9953 9954 9955 9956 9957 9958 9959 9960
9961 9962 9963 9964 9965 9966 9967 9968 9969 9970
9971 9972 9973 9974 9975 9976 9977 9978 9979 9980
9981 9982 9983 9984 9985 9986 9987 9988 9989 9990
9991 9992 9993 9994 9995 9996 9997 9998 9999 10000
```

- The interruption has no response
- Unless the following statement is added
  - if(isInterrupted()){break;}

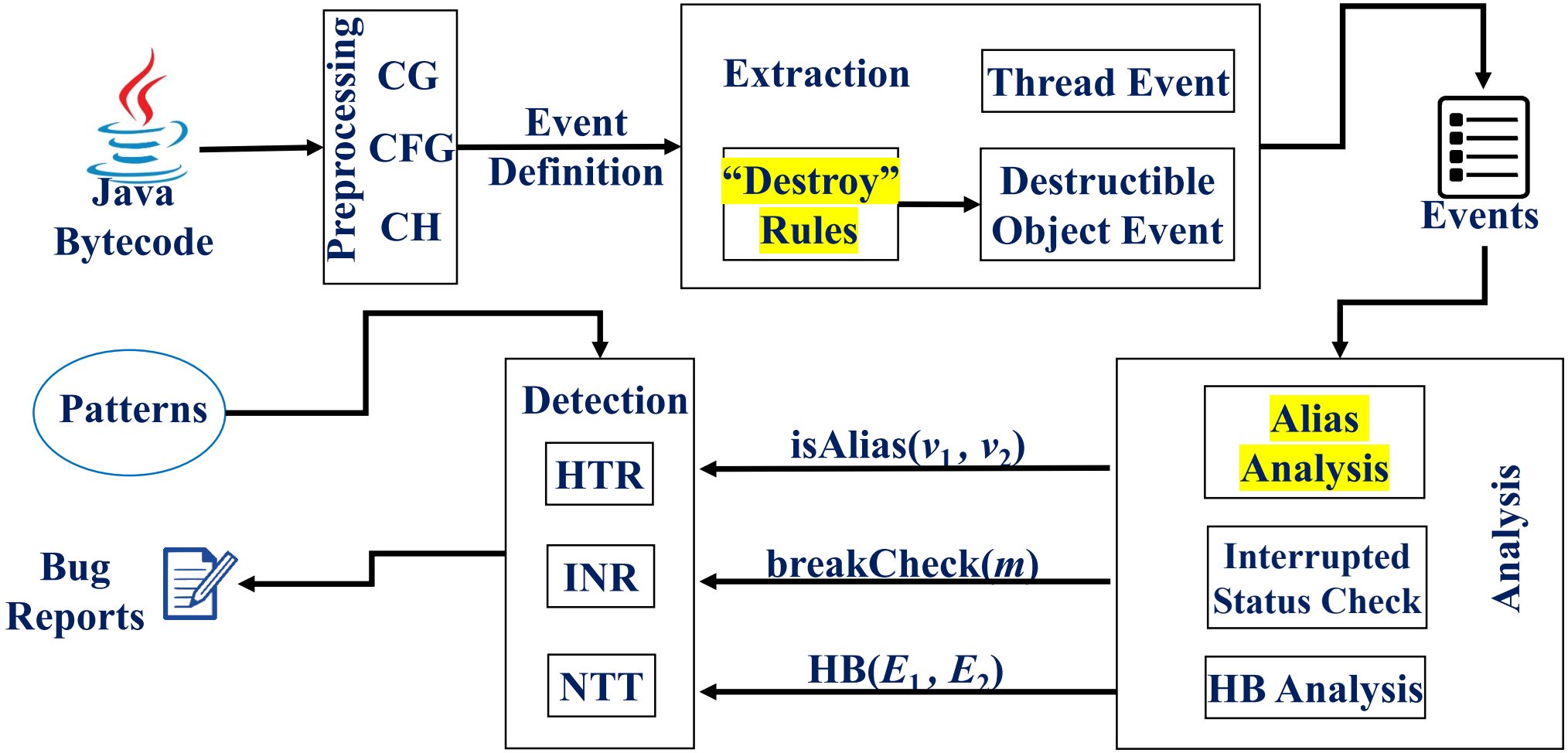
Related works and approaches

- AsyncChecker [1]
  - For AsyncTask (*android.os.AsyncTask*)
    - Deprecated in Android API level 30
    - Destructible classes: Activity and View
  - Path-sensitive approach
    - Time consuming

- Other works
  - Focus on the data race in Java/Android concurrent programs

[1]Pan et al. Static asynchronous component misuse detection for Android applications. ESEC/SIGSOFT FSE 2020

Overview of Static Event Analysis Approach



Static Event Analysis Approach

```
class  T extends Thread{
        void run(){
                if(isInterrupted()){break;}
        }
}

void main(…){
        T t = new T();
        Destructible d = new Destructible();
        t.set(d);
        t.start();
        d.destroy();
        t.interrupt();
}
```

Interrupted Check **Event**

Thread SetField **Event**

Thread Start **Event**

Destroy **Event**

Thread Interrupt **Event**

Static Event Analysis Approach

- **HTR:** Thread Start Event ⋀ Thread SetField Event
  - **Which method has the destruction semantics?**
- **INR:** Thread Start Event ⋀ Thread Interrupt Event
  ⋀ ¬(∃ Interrupted Check Event)
- **NTT:** Destroy Event happens before Thread Interrupt Event

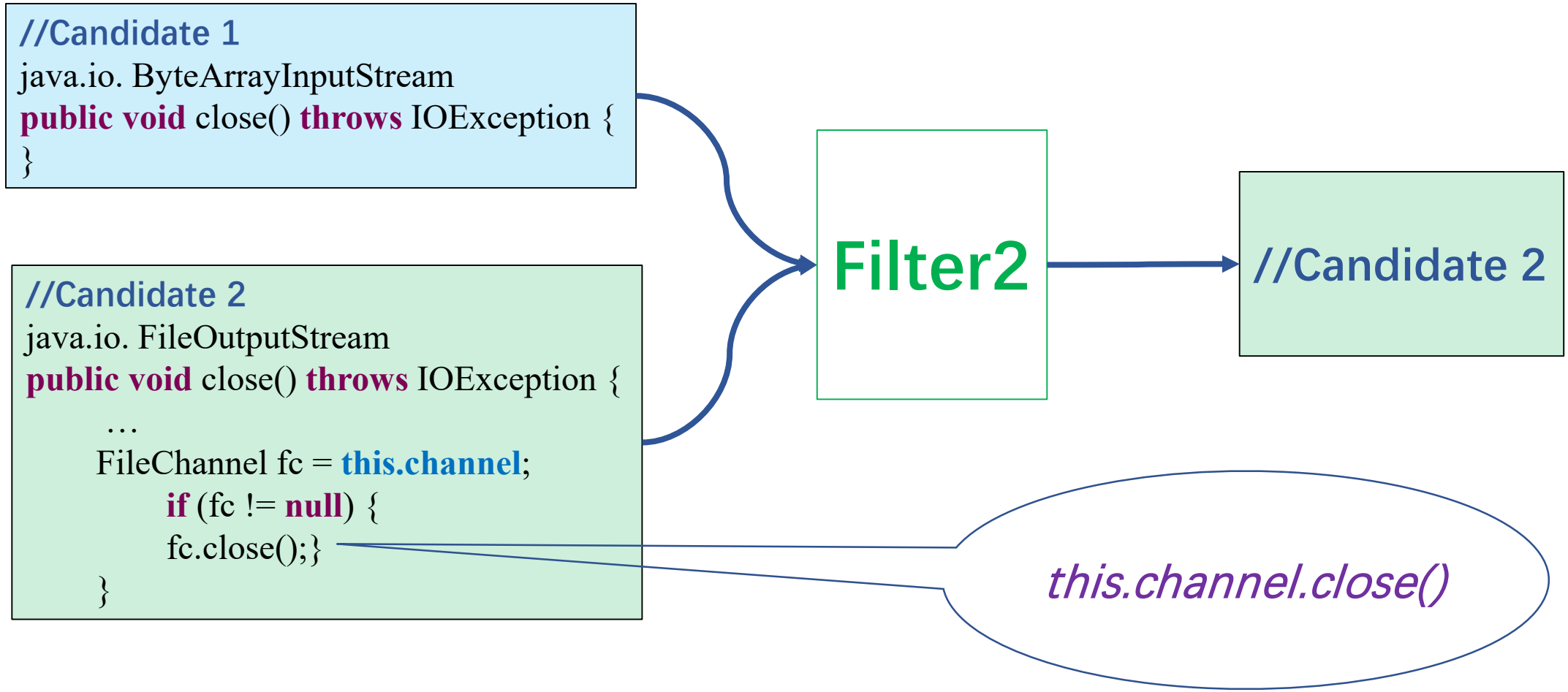
- How to be compatible with **Runnable** (*java.lang.Runable*)?

Identify Method with Destruction Semantics

- **Filter1:** filter out candidate(s)
  - Whose name contains “destroy” or
  - Whose name equals “close” or
  - Which is modified by the annotation “@PreDestroy”

- **Filter2:** keep the method with the destroy operation statement(s)
  - Assigns its field with the value null: `this.f = null;`
  - Invokes the “destroy” method of its field: `this.f.destroy();`
  - Contains JNI invocation.

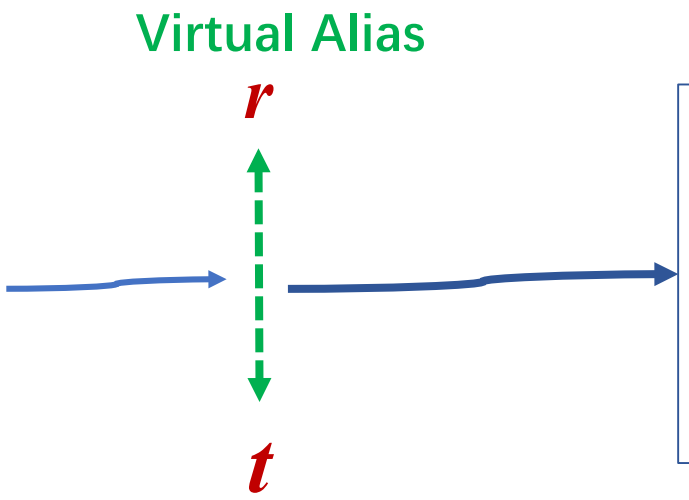
Identify Method with Destruction Semantics



Compatible with Runnable

- Construct a virtual alias relationship between a thread *t* and its real task: Runnable *r*

```
void main(…){
    Runnable r = …;
    t = new Thread(r);
    t.start();
}
```



- Event of *t* ⇔ Event of *r*
  - Start Event
  - Interrupt Event
  - Interrupted Check Event
  - …

Evaluation

- Developed a tool named Leopard based on FlowDroid.
- Dataset: 9 large Java Programs and 147 Android apps.
- **RQ1:** Can Leopard find the thread related misuses in Java programs and Android apps?
- **RQ2:** How efficient is Leopard against the existing approach?
- **RQ3:** Do developers take the misuse as a serious problem?

Detection Ability of Leopard

Project	Fork[1]	Star	LOC	HTR	INR	NTT	Time (s)
cxf[44]	1.3K	758	1,066K	22	58	7	6,184
hivemq[45]	211	738	737K	1	4	1	251
jetty[46]	1.8K	3.3K	181K	7	12	3	53
junit5[36]	1.2K	5.1K	31K	1	2	0	12
micronaut[47]	922	5.5K	531K	12	19	5	953
quarkus[48]	1.8K	9.7K	434K	17	34	3	685
spring[49]	33.3K	47.1K	1,507K	5	20	2	321
tomcat[35]	4.1K	6K	193K	6	18	1	59
wildfly[50]	2.1K	2.7K	1,443K	8	9	8	1,035

in Java Programs
220 Misuses

Tool	HTR	INR	NTT	Total
Leopard	**333 (95)**	**94 (34)**	**296 (89)**	**723 (103)**

723 Misuses in
103 Apps

M(N) = #Miuses (#Apps)

Leopard VS AsyncChecker on Apps

- Misuses detected by AsyncChecker increase by only a few (42 to 47) as time goes up from 5 minutes to 30 minutes

- AsyncChecker has found 47 misuses in 15 apps
- Leopard has found 723 misuses in 103 apps while
  - Average Time: 60s Max Time: 402s

Tool		HTR	INR	NTT	Total
AsyncChecker	5min	18 (14)	9 (5)	15 (12)	42 (15)
	30min	19 (14)	12 (5)	16 (12)	47 (15)
Leopard		**333 (95)**	**94 (34)**	**296 (89)**	**723 (103)**

Leopard VS AsyncChecker on Apps

- **Manually Check 78 misuses in 15 Apps for *Precision, Recall and F_1***
  - **Where AsyncChecker can found misuses**

App Package Name	AsyncChecker									Leopard								
	HTR			INR			NTT			HTR			INR			NTT		
	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
ch.hgdev.toposuite	1	0	3	0	0	0	1	0	3	4	0	0	0	0	0	4	0	0
com.dosse.bwentrain.androidPlayer	1	0	0	5	0	0	1	0	1	1	0	0	2	1	2	1	0	1
com.ghostsq.commander	1	0	1	1	0	2	1	0	1	2	0	0	3	0	0	2	0	0
com.github.axet.audiorecorder	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0
com.jovial.jrpn	1	0	1	1	0	0	0	0	2	2	0	0	1	0	0	2	0	0
com.spisoft.quicknote	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
de.reimardoeffinger.quickdic	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
de.rochefort.childmonitor	2	0	0	0	0	0	1	0	1	2	0	0	0	0	0	2	0	0
godau.fynn.usagedirect	1	0	1	0	0	0	1	0	1	2	0	0	0	0	0	2	0	0
godau.fynn.usagedirect.system	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
menion.android.whereyougo	1	0	1	0	0	1	1	0	1	2	0	0	1	0	0	2	0	0
namlit.siteswapgenerator	3	0	6	0	0	0	3	0	6	9	0	0	0	0	0	9	0	0
net.justdave.nwsweatheralertswidget	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
net.sourceforge.opencamera	2	0	0	1	0	0	2	0	0	2	0	0	1	0	0	2	0	0
xyz.myachin.downloader	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
Sum	30	0	13	12	1	6	15	0	17	31	0	0	12	1	2	31	0	1

- Recall and F_1 value of Leopard are outperform
- With only a little decrease in Precision

	Precision	*Recall*	F_1
AsyncChecker:	100%,	53.8%,	0.700
Leopard :	98.7%,	96.1%,	0.974

Confirmed Issues from Developers

App	Fork	Star	#Download	#Misuse	Issue ID
VocableTrainer [52]	10	27	500K+	1	93
toposuite[53]	2	12	5K+	4	3
APK-Explorer-Editor[54]	53	278	7.8K+	**1***	29
LRC-Editor[55]	9	43	100K+	3	35
Nextcloud[56]	1.5K	3.2K	100K+	7	1069
TRIfA[57]	52	220	5K+	14	350
AppManager[58]	174	2.3K	80K	1	854
Siteswap Generator[59]	3	13	1K+	9	55
TC Slim[60]	66	1.1K	10K+	2	36
blabber.im[61]	16	41	-	**6***	674
OSMDashboard[62]	8	52	500+	**1***	169
Ghost Commander[51]	-	-	1,000K+	**1***	93
Offline Puzzle Solver[63]	-	1	-	**1***	1
FitoTrack[64]	48	161	5K+	3	400
Conversations[65]	1.3K	4.2K	100K+	**2***	4366
monocles chat[66]	7	14	-	**6***	44
ccgt[67]	4	11	-	1	7
Notes[68]	121	769	10K+	**1***	1574
FreeRDP[69]	23.5K	8.6K	10K+	**2***	8158
Total	-	-	-	66 (**21***)	-

Confirmed: 66

*Fixed: 21

Detection of Java Basic Thread Misuses Based on Static Event Analysis

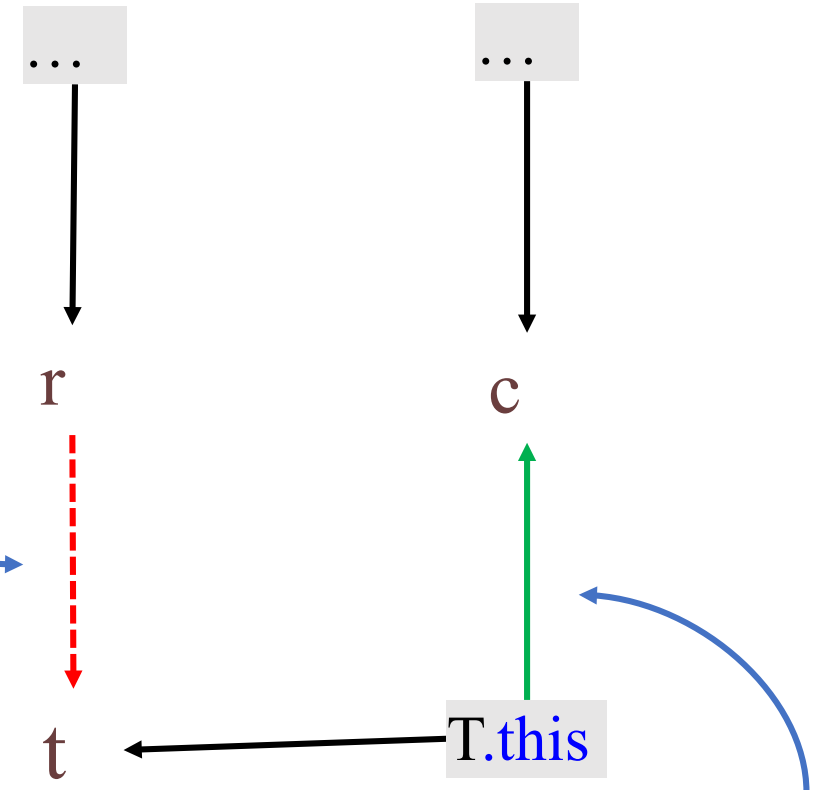
- HardToRelease (HTR) | Interrupt NoResponding (INR) | …
- Destruction Semantic Method Identification
- Compatible with Runnable

Question?

Email: cuibq@ios.ac.cn
Institute of Software, Chinese Academy of Sciences (ISCAS)
University of Chinese Academy of Sciences (UCAS)

Backward alias analysis

```
1  static void entry(…){
2      Runnable r = …;
3      t = new T(r);
4      t.start();  }
5  class  T extends Thread{
6          T(Runnable r){  super(r);  }
7          void run(){ doTask(); }
8          void doTask(){
9            c = Thread.currentThread();
10                  …
11         }
12}
```



Evaluation- Event Coverage

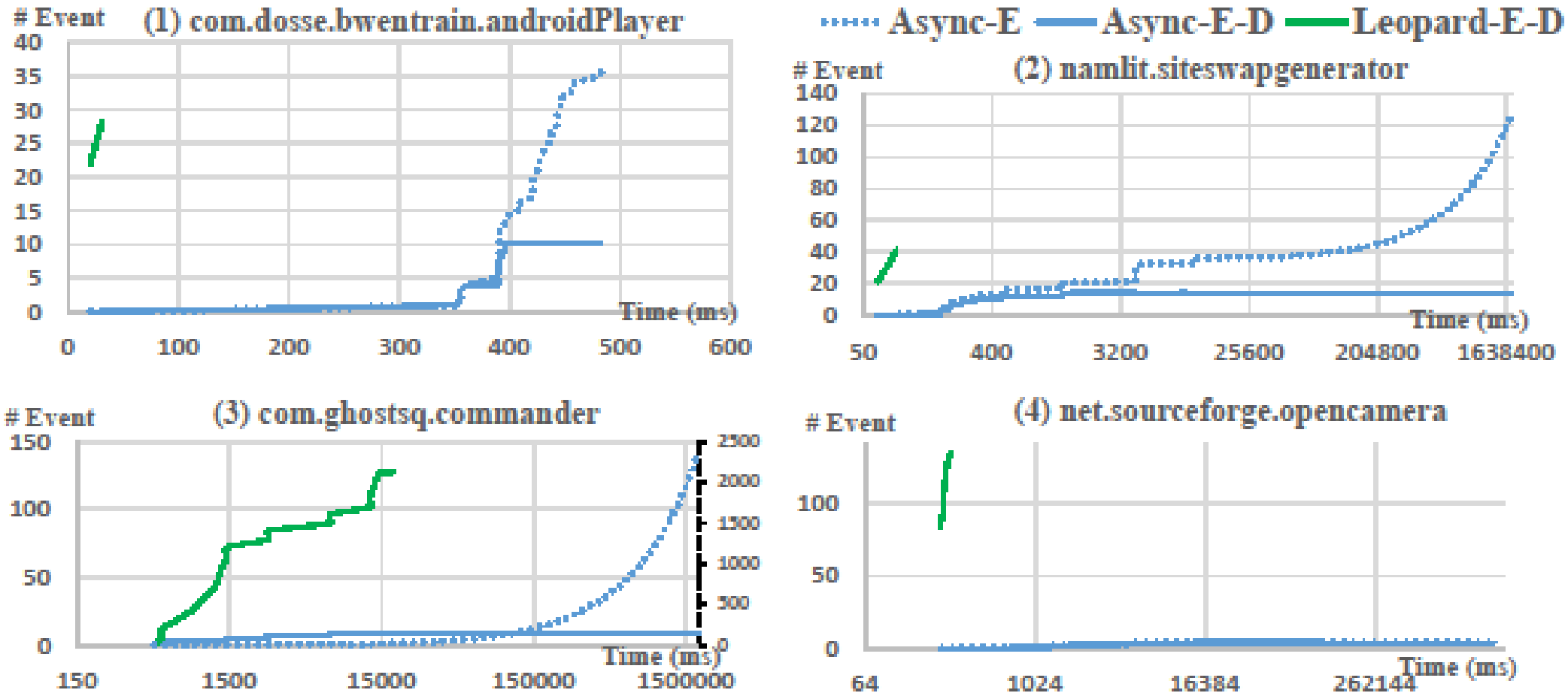


Figure 4: Event Coverage. Async-E means the number of events covered during detection of AsyncChecker. Async-E-D and Leopard-E-D mean the number of events covered during detection of AsyncChecker and Leopard after deduplication, respectively. The second vertical axis on the right in the subfigure (3) is for Async-E only

Algorithm (HTR)

Algorithm 1: HTRMiuseDetection

Input: startEvents, setFieldsEvents

```
1 for each s ∈ startEvents do
2     for each f ∈ setFieldEvents do
3         if isAlias(s.caller, f.caller) ∧
            isDestructibleClass(f.arg.getClass()) then
4             recordHTRMiuse(s); // record
```

Algorithm (INR)

Algorithm 2: INRMisuseDetection

```
   Input: startEvents, allClasses, interruptEvents
1  unCheckedClasses = ∅;
2  for each c ∈ allClasses do
3      if c inherits from Runnable then
4          if BreakCheck(c.run()) then
5              unCheckedClasses.add(c);
6  for each s ∈ startEvents do
7      for each c ∈ s.caller.pointClasses do
8          for each i ∈ interruptEvents do
9              if unCheckedClasses.contains(c)
10             ∧ isAlias(s.caller, i.caller) then
11                 recordINRMiuse(s); // record
```

Algorithm (NTT)

Algorithm 3: NTTMisuseDetection

```
   Input: startEvents, interruptEvents, destroyEvents
1  for each s ∈ startEvents do
2      iFlag = false;
3      for each d ∈ destroyEvents do
4          if s.caller references d.caller then
5              for each i ∈ interruptEvents do
6                  if isAlias(s.caller, i.caller) then
7                      if ¬HB(i, d) then
8                          recordNTTMiuse(s); // record
9      if ¬ destroyEvents.isEmpty() ∧ interruptEvents.isEmpty() then
10         recordNTTMiuse(s); // record
```