Designing Robust Software Analysis and Artificial Intelligence Approaches For Cybersecurity

Giacomo Iadarola
Research fellow (Assegnista di Ricerca) at IIT-CNR
PhD student at Department of Computer Science (University of Pisa)

TUTOR: Fabio Martinelli (IIT-CNR)

Interests: ***Software Testing and Analysis - Mobile Security***
Machine Learning - Cryptography (Blockchain)

ToDo: ***Adversarial Learning - Explicable AI***

Università di Pisa

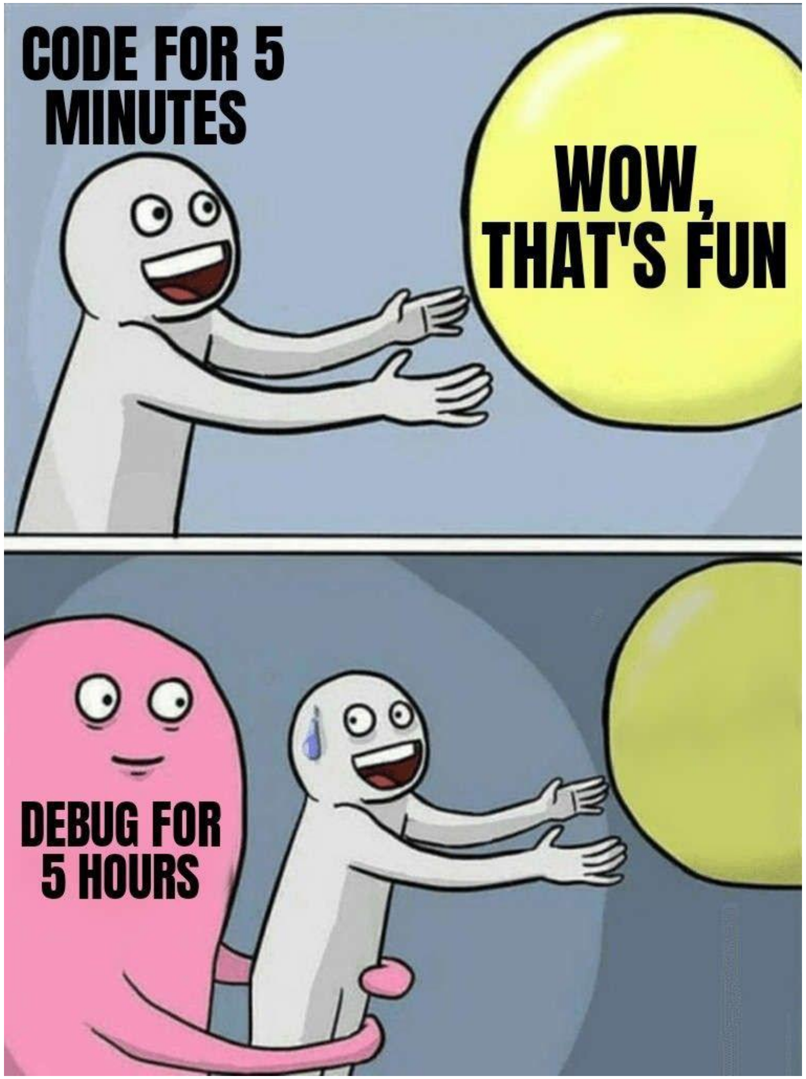
National Research Council of Italy

Istituto di Informatica e Telematica del CNR

Outline

- Introduction
- Let's talk about:
  - ➢ Software Testing and Analysis
  - ➢ Mobile Security
- Future Works
  - ➢ Adversarial Learning
- Conclusion

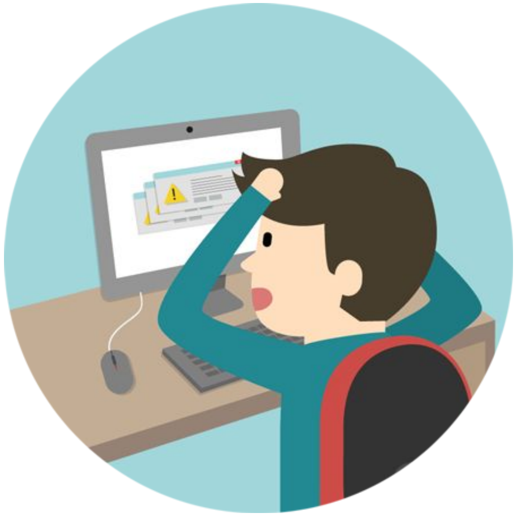
Software Testing and Analysis



Introduction

All software have bugs, we know that…

Number of bugs per kLOC:
Between 57.02 bugs/kLOC and 10.09 bugs/kLOC



Time to Fix:
Between 5 and 340 days

… and also the smallest vulnerability may trigger a domino effect!

- Aljedaani, Wajdi, and Yasir Javed. "Bug Reports Evolution in Open Source Systems."
- Xia, Xin, et al. "An empirical study of bugs in software build system."

Goal of GrapPa

Design and implement a generic bug finder that uses machine learning to learn from buggy examples

- Static analysis
  - from source code to graph
- Train graph-based classifier
- Classify graphs of previously unseen code

What is “buggy”?

```
void example(User user){
    String name = user.getName();
    if(user != null){
        int id = user.getId();
    }
}
```

Buggy example

```
void example(User user){
    if(user != null){
        String name = user.getName();
        int id = user.getId();
    }
}
```

Non-Buggy example

What is "buggy"?

```
void example(User user){
    String name = user.getName();
    if(user != null){
        int id = user.getId();
    }
}
```

Buggy example

```
void example(User user){
    if(user != null){
        String name = user.getName();
        int id = user.getId();
    }
}
```

Non-Buggy example

Background

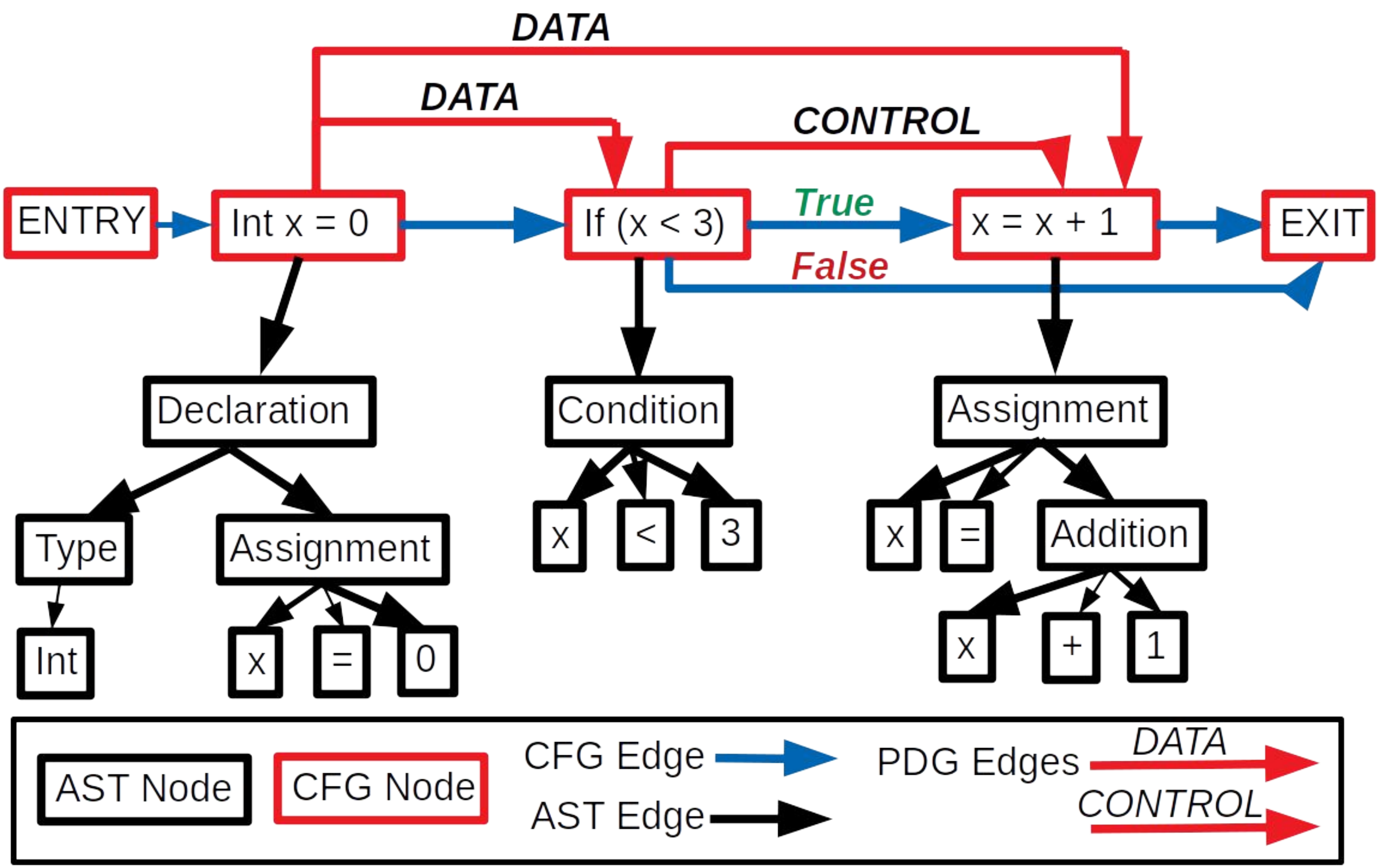
- **Code Property graph (CPG)**
  - ➢ Merges classical graph representation into one data structure
- **Contextual Graph Markov Model (CGMM)**
  - ➢ Neural network approach for processing graph data
- **Multilayer Perceptron (MLP)**
  - ➢ Classical neural network model

Background - CPG

```
void Example(){
    int x = 0;
    if (x < 3){
        x = x + 1;
    }
}
```

Code example

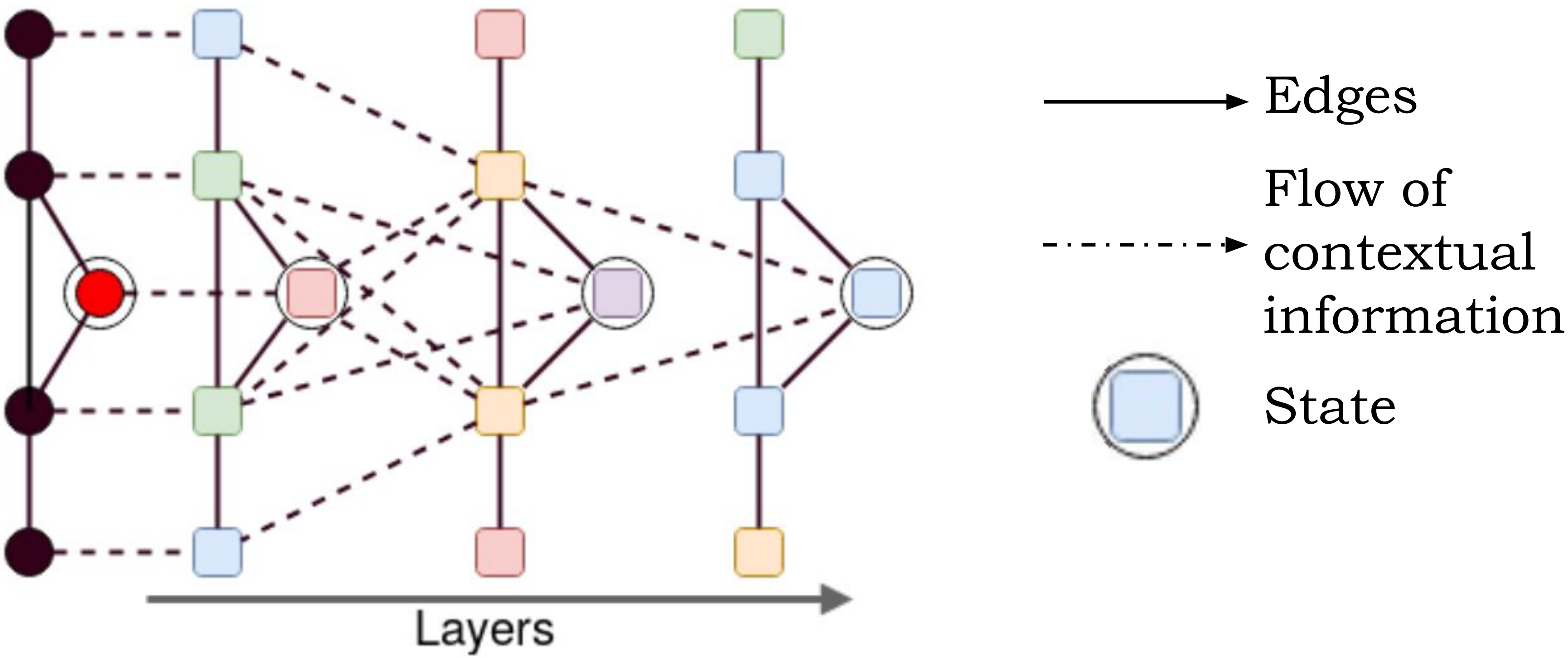
Background - CPG



- Yamaguchi, Fabian, et al. "Modeling and discovering vulnerabilities with code property graphs." (2014).

Background - CGMM

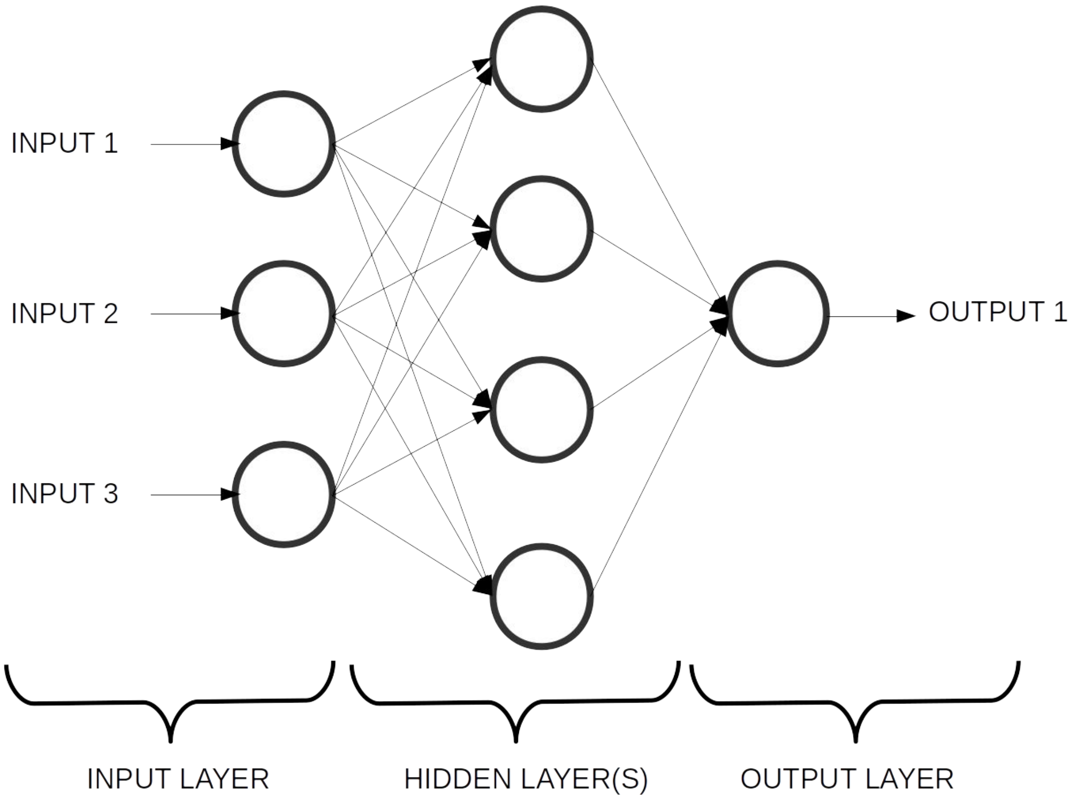
An unsupervised model able to encode graphs of varying size and topology to a fixed dimension vector



- Bacciu Davide, Federico Errica, and Alessio Micheli. "Contextual Graph Markov Model: A Deep and Generative Approach to Graph Processing." (2018).

Background - MLP

Feedforward artificial neural network.



Dropout

The dropout layer randomly selects a fraction rate of input neurons that are ignored during training

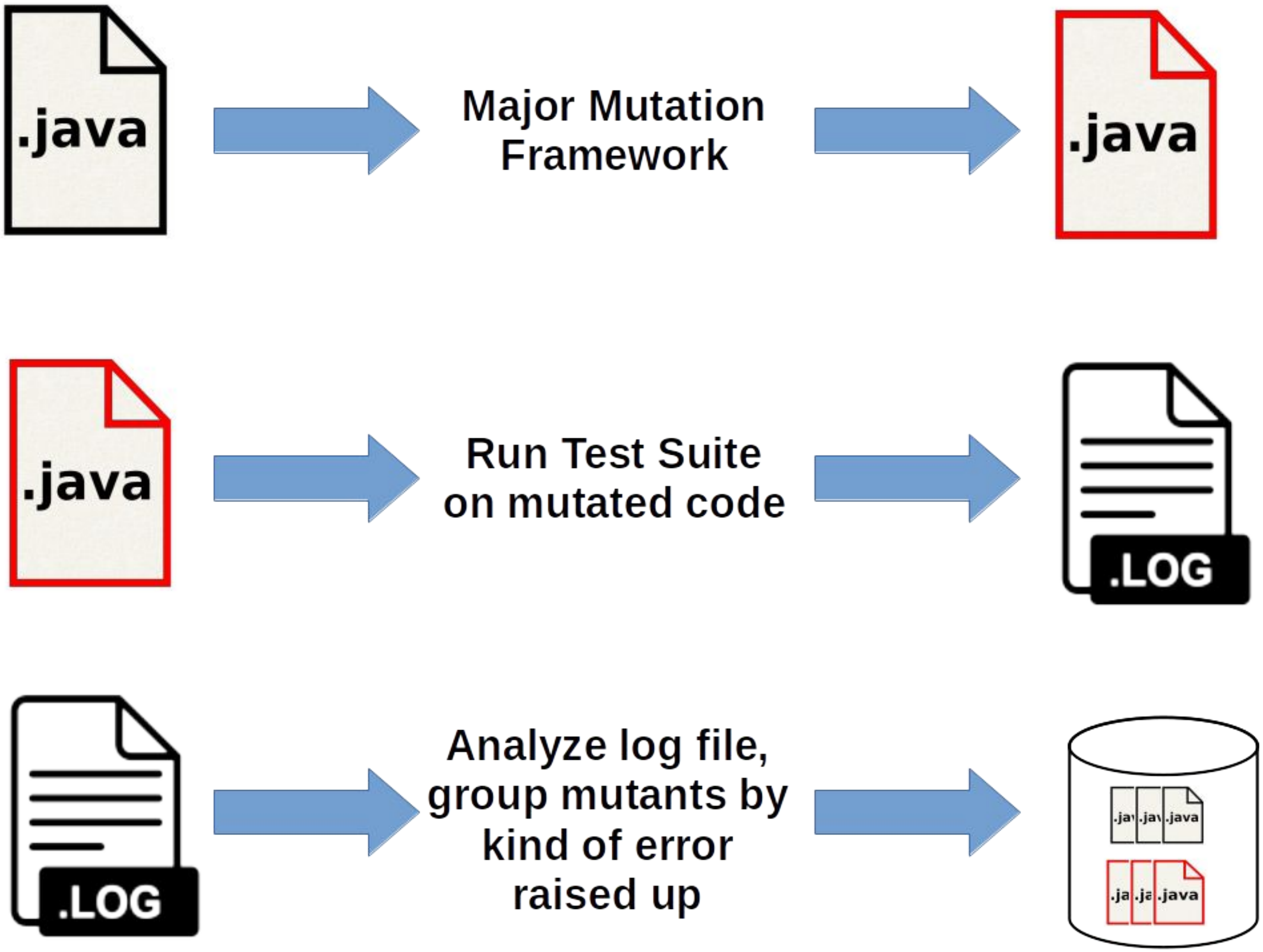
Methodology

Approach steps

- **Database of source code samples**
- **Static analysis and graph generation**
- **Graph vectorization**
- **Classification**

Approach - The Dataset

Approach - The Dataset



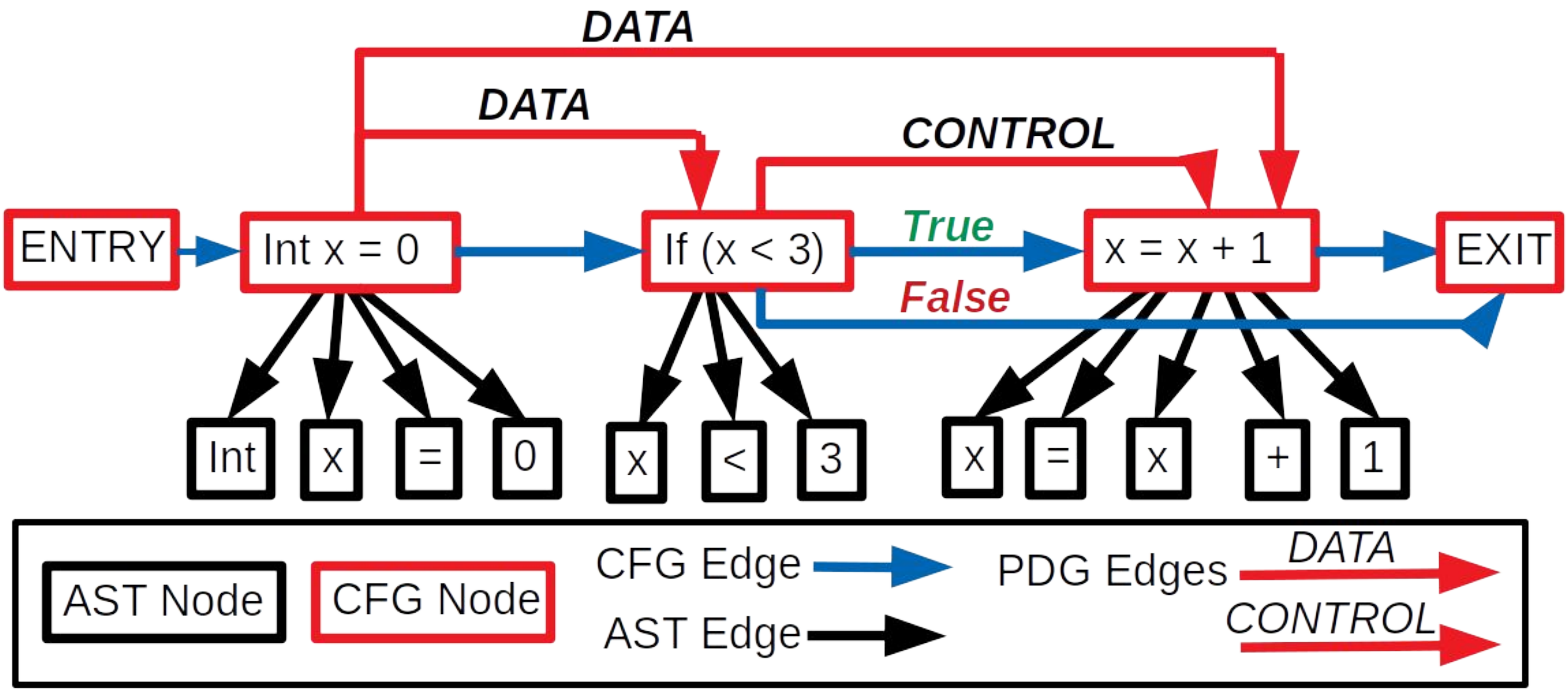
Approach - The Dataset

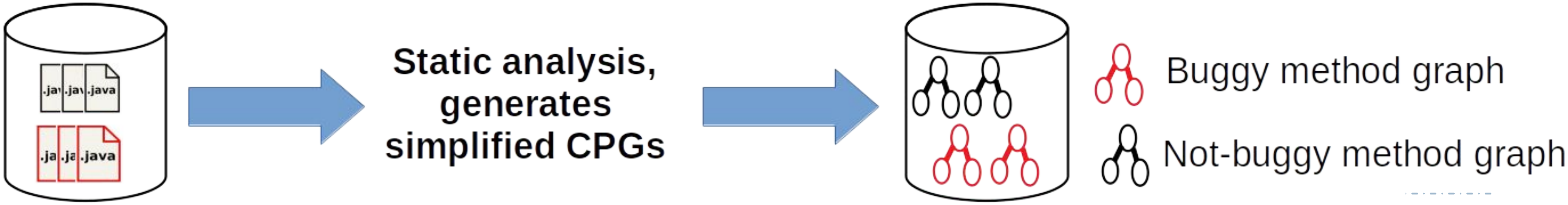
Arithmetic Operator Replacement	`a + b`	⟼ `a - b`
Logical Operator Replacement	`a ^ b`	⟼ `a | b`
Conditional Operator Replacement	`a || b`	⟼ `a && b`
Relational Operator Replacement	`a == b`	⟼ `a >= b`
Shift Operator Replacement	`a >> b`	⟼ `a << b`
Expression Value Replacement		
Replaces an expression (in an otherwise unmutated statement) with a default value.	`return a`	⟼ `return 0`
	`int a = b`	⟼ `int a = 0`
Literal Value Replacement	`0`	⟼ `1`
	`1`	⟼ `-1`
	`1`	⟼ `0`
	`true`	⟼ `false`
	`false`	⟼ `true`
	`"Hello"`	⟼ `""`
STatement Deletion		
Deletes (omits) a single statement:		

List of applied mutations

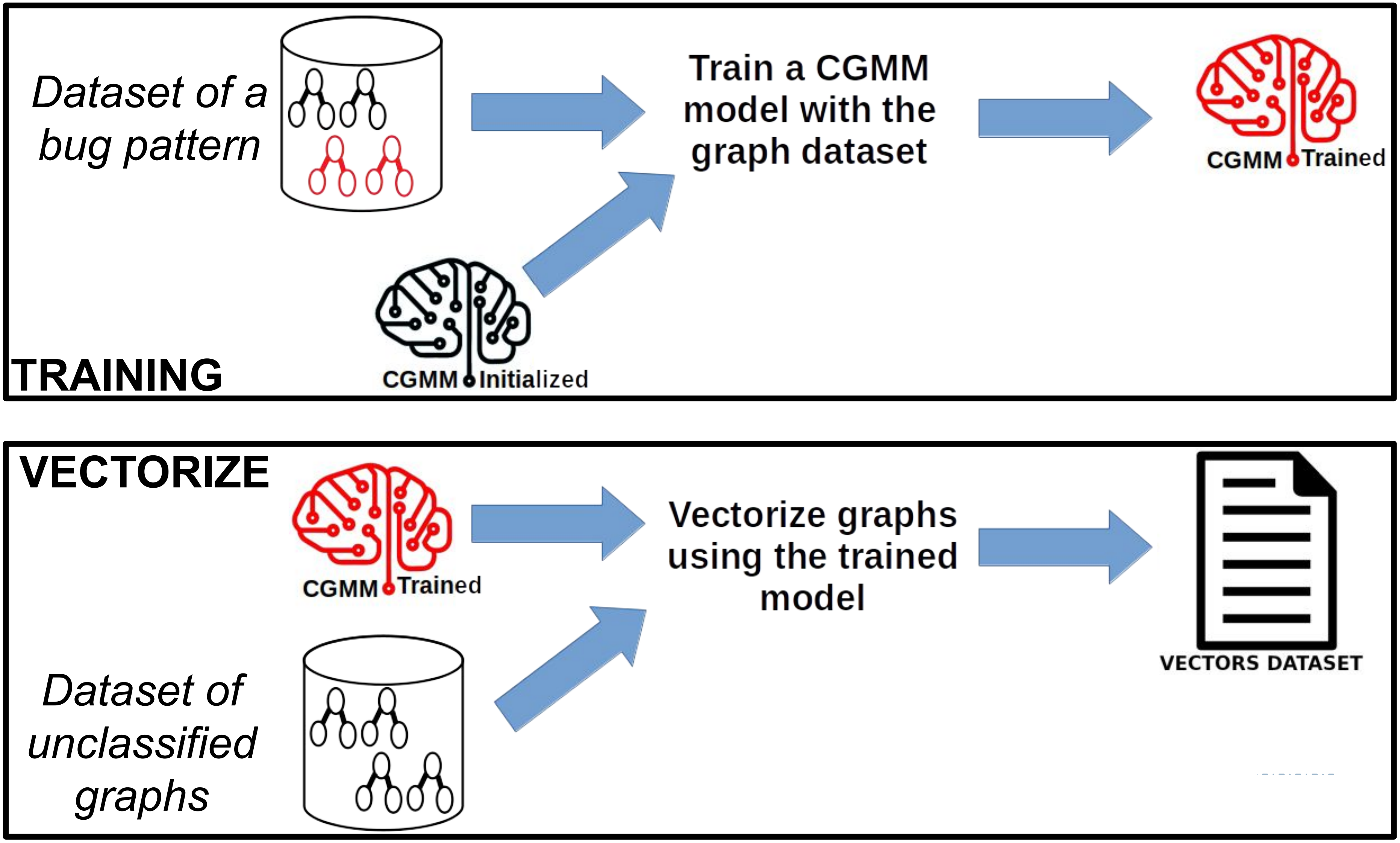
The major mutation framework - documentation. http://mutation-testing.org/

Approach - Generate CPGs



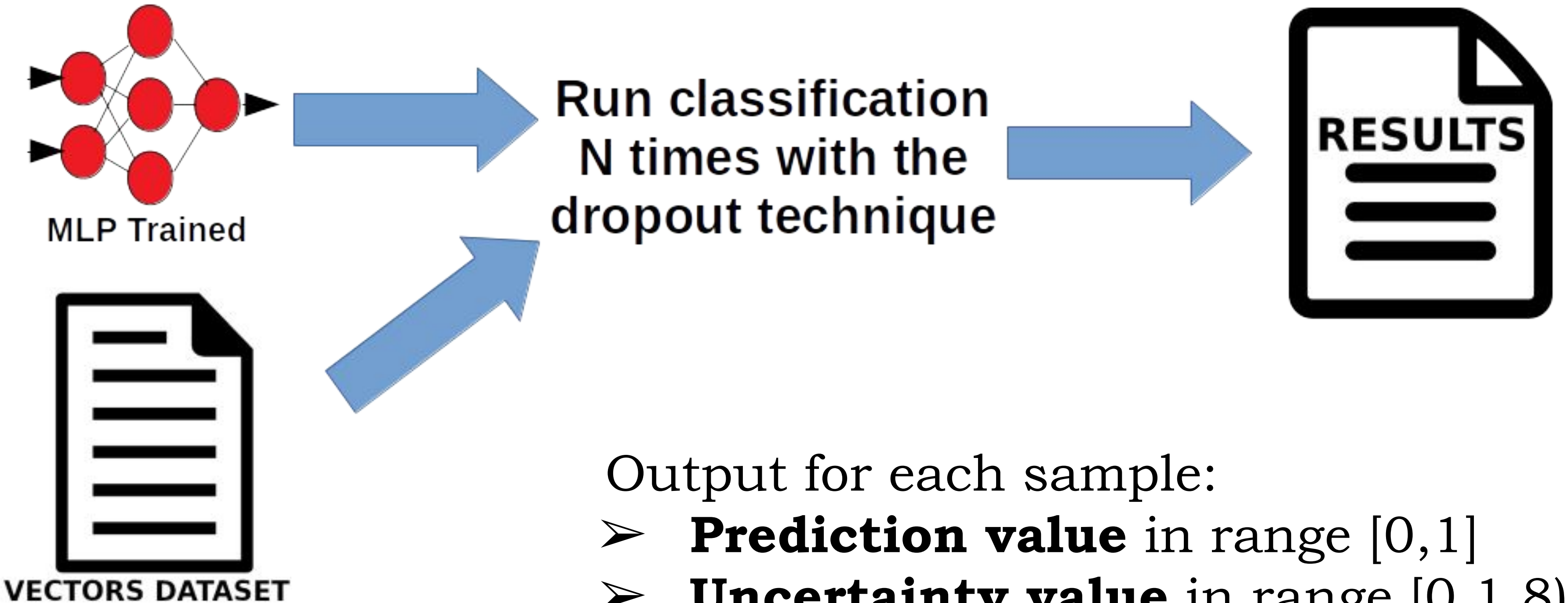


Approach - Graphs vectorization



Approach - Classification

Approach presented by Gal Y. e Ghahramani Z. to calculate the uncertainty of the model predictions.



Output for each sample:

- ➢ **Prediction value** in range [0,1]
- ➢ **Uncertainty value** in range [0,1.8)

- Gal, Yarin, and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning." (2016).

Approach - Classification

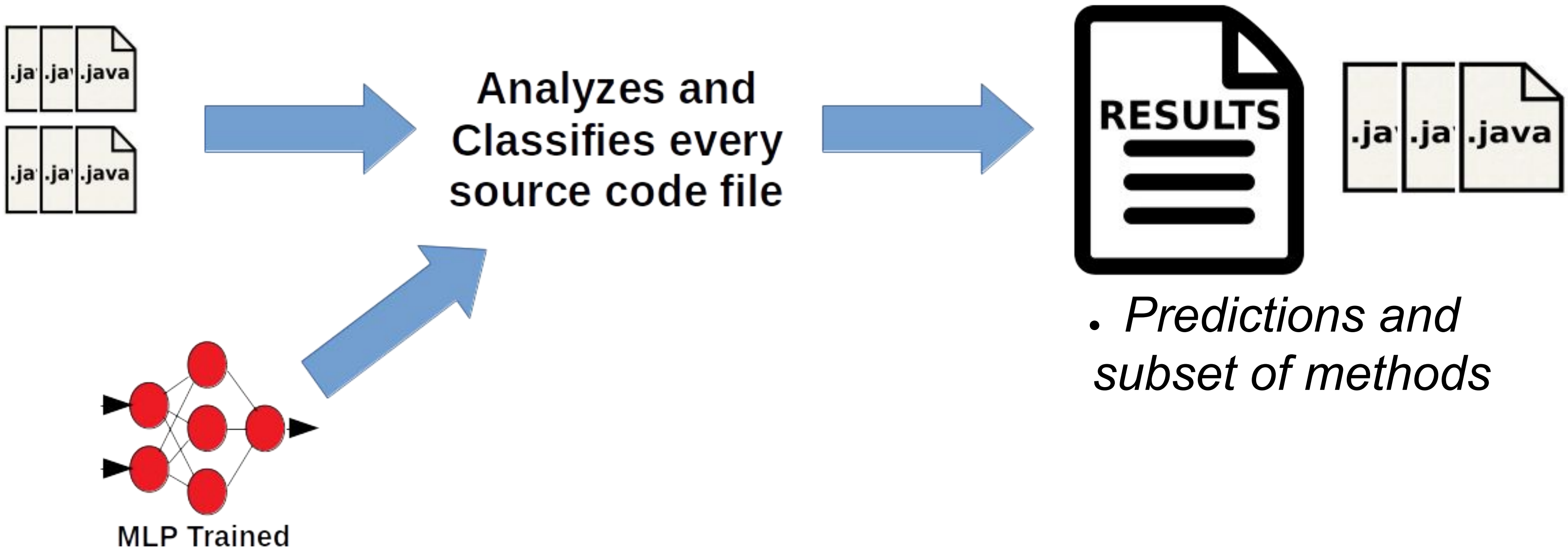
We define uncertainty as:

$$uncertainty(vect) = |pred(vect) - avg\_pred\_dropout(vect)| + std\_pred\_drop(vect)$$

Final step: removing graphs/vector:

$$filter\_vectors(vect) = \begin{cases} remove & if\ uncertainty(vect) > T \\ store & otherwise \end{cases}$$

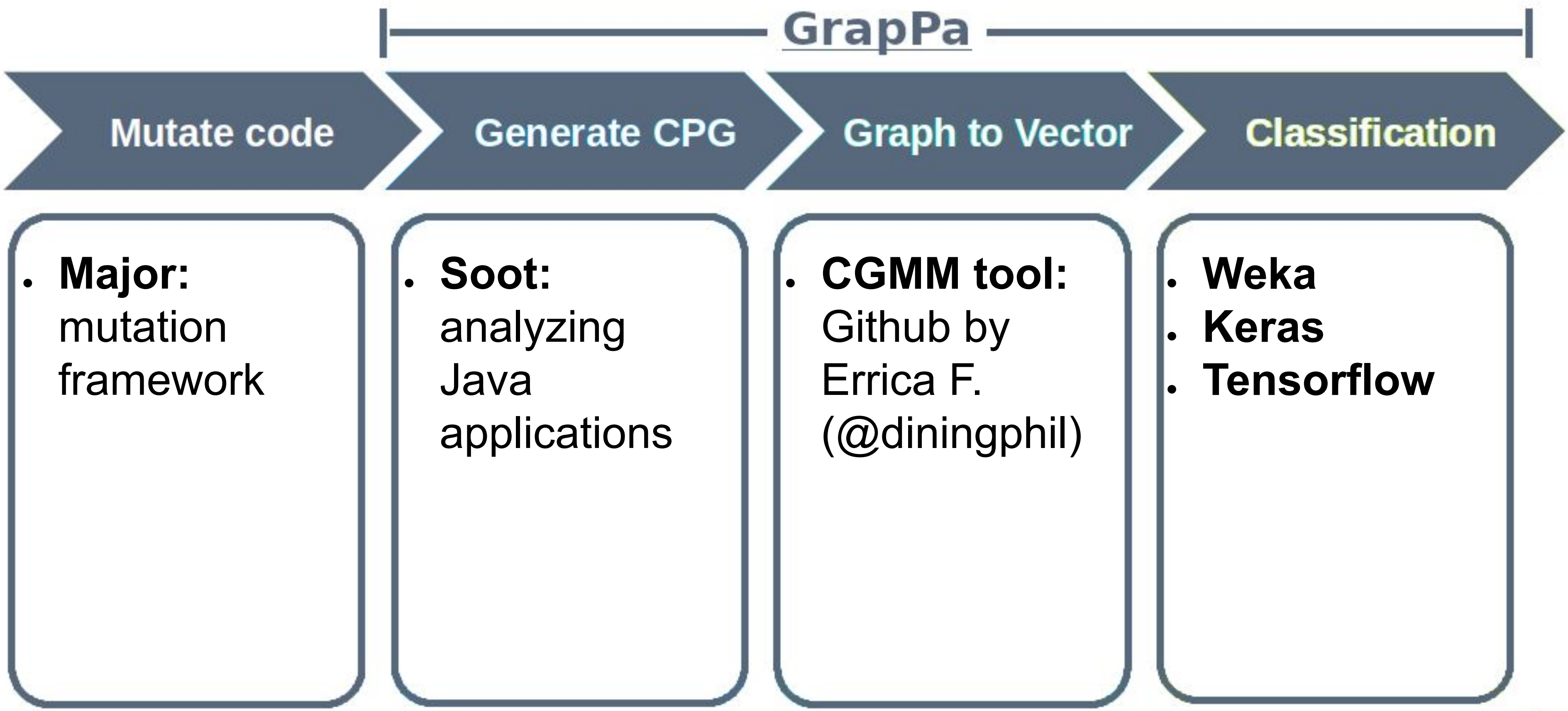
Approach - Classification



- *Predictions and subset of methods*

Model trained on a specific bug pattern

Implementation - GrapPa



Results - NPE Example #1

- Classified by the model as: **BUGGY**
- Manual check classified as: **BUGGY**

```java
public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
                    ValueAxis domainAxis, ValueAxis rangeAxis,
                    int rendererIndex,
                    PlotRenderingInfo info) {

    PlotOrientation orientation = plot.getOrientation();
    AxisLocation xAxisLocation = plot.getDomainAxisLocation();
    AxisLocation yAxisLocation = plot.getRangeAxisLocation();
    RectangleEdge xEdge = Plot.resolveDomainAxisLocation(xAxisLocation,
        orientation);
    RectangleEdge yEdge = Plot.resolveRangeAxisLocation(yAxisLocation,
        orientation);
    ...
}
```

Results - NPE Example #2

- Classified by the model as: **BUGGY**
- Manual check classified as: **NON-BUGGY**

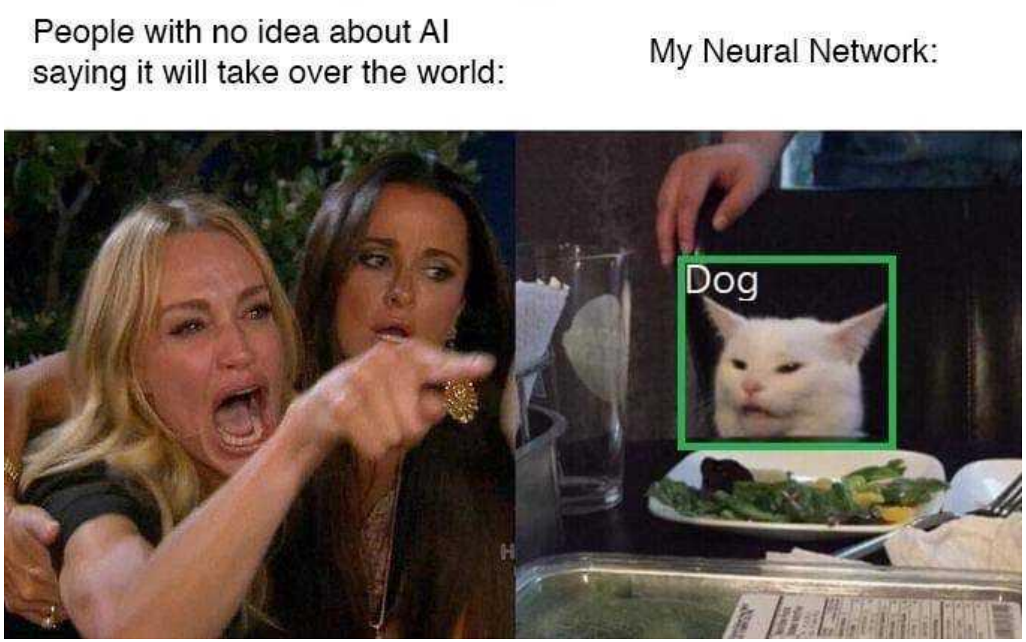
```
public void add(Block block, Object key) {
    // since the flow layout is relatively straightforward,
    // no information needs to be recorded here
}

```

Results - NPE Example #2

- Classified by the model as: **BUGGY**
- Manual check classified as: **NON-BUGGY**

```
public void add(Block block, Object key) {
    // since the flow layout is relatively straightforward,
    // no information needs to be recorded here
}
```



Take-home points for GrapPa

Novel and general approach

- ➢ Use of recent works
- ➢ Useful for developers in improving code security
- ➢ Not need prior-knowledge on code (neither on the bug pattern)

The tool GrapPa (https://github.com/Djack1010/GrapPa)

- ➢ Three trained models available
- ➢ Easy to include more bug patterns

Simplified version of the **CPG**

Three datasets of **syntetich bugs** available online

- ➢ https://github.com/Djack1010/BUG_DB

Mobile Security

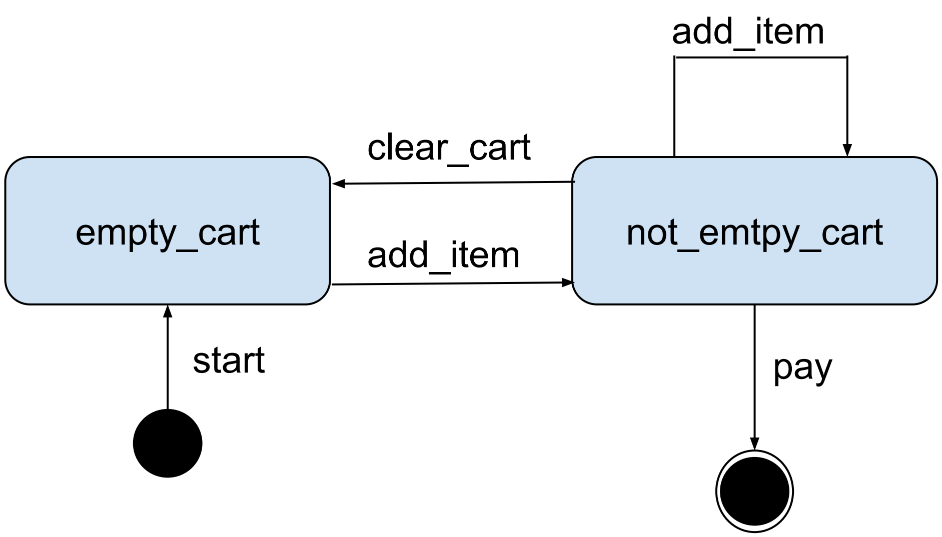
Motivation

- Mobile devices handle huge amount of sensitive data
  - ➢ really lucrative and attractive for attackers
- Mobile malware abuse of the “weakest link” of security
  - ➢ malware detection techniques to mitigate
- Banking malware are critical
  - ➢ significant exposure to every infected device

Formal methods in a nutshell

➢ Formal Model & Temporal Logics

Calculus of
Communicating Systems of Milner (CCS)



Modal mu-calculus (extended form)

doing_shopping =
init ⋀ empty_cart ⋀ not_empty_cart

init = init.<start>empty_cart

empty_cart =
empty_cart.<add_item>not_empty_cart

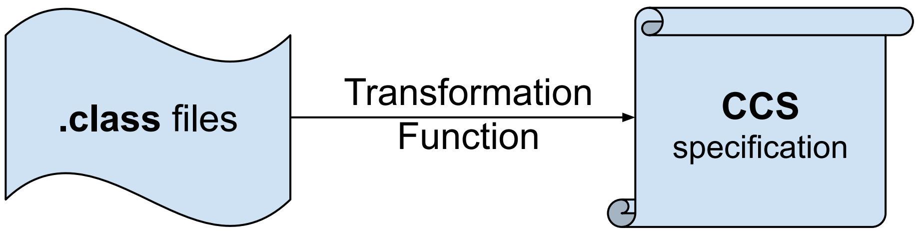
not_empty_cart =
not_empty_cart.<add_item>not_empty_cart
⋁ not_empty_cart.<pay>true

The Method

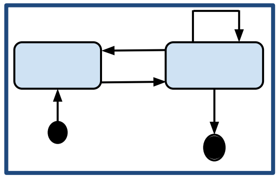
➢ Formal Model & Temporal Logics

- Java Bytecode-to-CCS transformation
  - ➢ defined for each instruction

App under analysis



Labelled Transition System



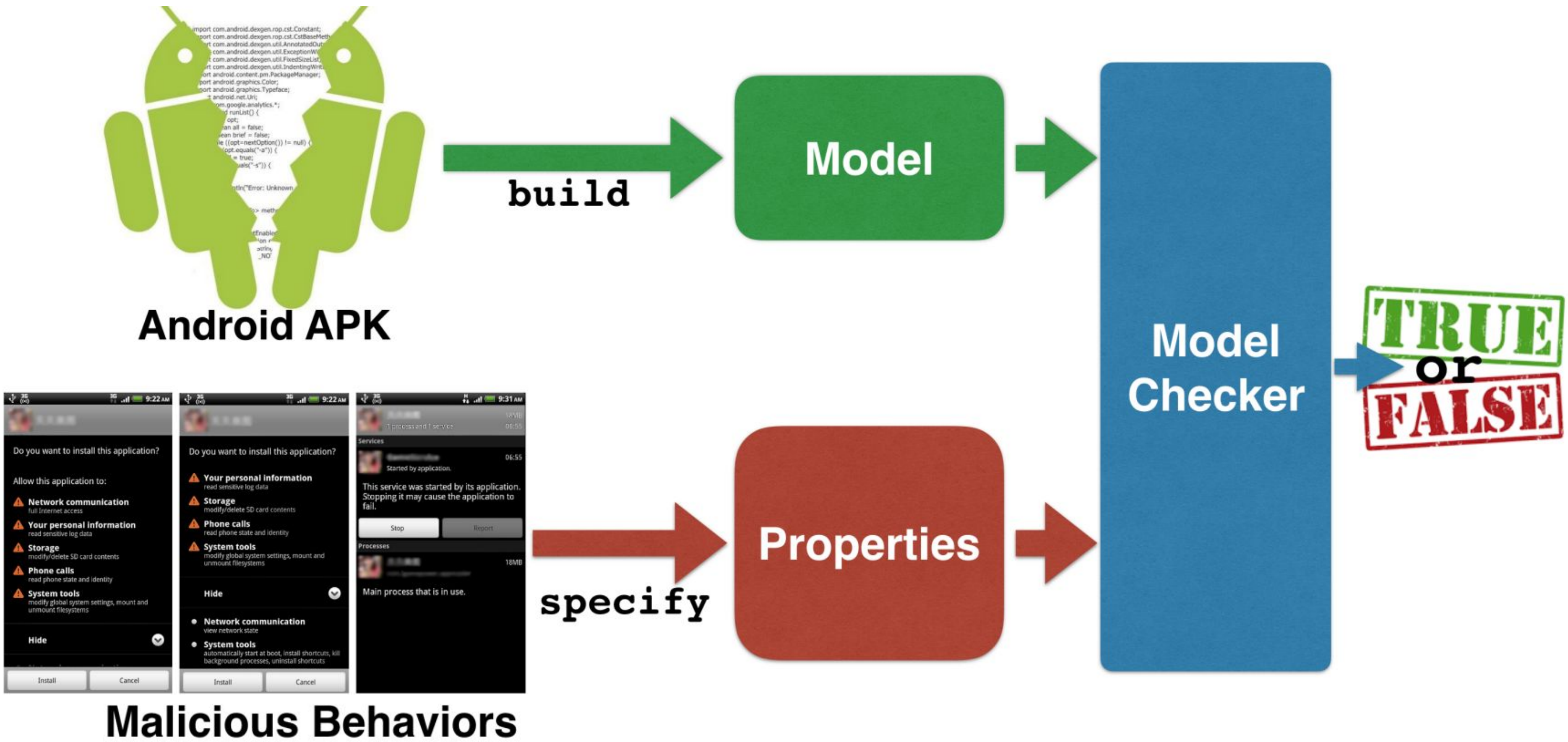
- Specify set of properties
  - ➢ describing malware behaviours

Manual inspection and current literature

Properties

$$
\begin{aligned}
\varphi_{TIGERBOT} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_1} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_1} &= \mu X.\langle pushpackName\rangle\, \varphi_{1_2} \vee \langle -pushpackName\rangle\, X\\
\varphi_{1_2} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_3} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_3} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_4} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_4} &= \mu X.\langle pushprodName\rangle\, \varphi_{1_5} \vee \langle -pushprodName\rangle\, X\\
\varphi_{1_5} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_6} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_6} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_7} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_7} &= \mu X.\langle pushcoorType\rangle\, \varphi_{1_8} \vee \langle -pushcoorType\rangle\, X\\
\varphi_{1_8} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_9} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_9} &= \ldots
\end{aligned}
$$

The Method

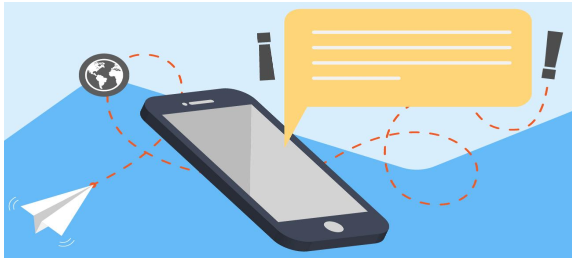

Android APK
build
Model
Model
Checker
TRUE
or
FALSE
Malicious Behaviors
specify
Properties

Features and Pros of the Method

- Use of formal methods
- Inspection directly on Java Bytecode
- Capture of malicious behaviours at finer granularity
- Method independent of source programming language
- Identification payload without decompilation

The Experiment on the Overlay family

1. Intercepting SMS messages



2. Stealing money in background

3. Password resetting

[1] Wei, Fengguo, et al. "Deep ground truth analysis of current android malware." *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Cham, 2017.
[2] Han, Qian, et al. "DBank: Predictive Behavioral Analysis of Recent Android Banking Trojans." *IEEE Transactions on Dependable and Secure Computing* (2019).
[3] Wazid, Mohammad, Sherali Zeadally, and Ashok Kumar Das. "Mobile banking: evolution and threats: malware threats and security solutions." *IEEE Consumer Electronics Magazine* 8.2 (2019)
[4] Pan, Jordan "Fake Bank App Ramps Up Defensive Measures" Available at: http://tiny.cc/xz209y [Accessed: Oct '19]

The Experiment on the Overlay family

```
public void onReceive(Context var1, Intent var2){
  Bundle var5=var2.getExtras();
  int var3=var5.getInt('const_id_send_sms');
  int var4=var5.getInt('const_task_id_send_sms');
  M.d('receiverStatusSms', 'smsId: ' + var3 \
    + "; smsTaskId: " + var4 \
    + "; getResultCode(): " \
    + this.getResultCode());
  switch(this.getResultCode()) {
  case -1:
  if (var4 >= 0) {
    SocketService.access$000(this.this$0, var4);
  }
  SocketService.access$100(this.this$0, var3, \
    Const._SMS_STATUS_SEND);
```

Malicious Behaviour in Java Code

$$
\begin{aligned}
\psi &= \mu X.\langle pushconstidsendsms\rangle\ \psi_1 \vee \langle -pushconstidsendsms\rangle X\\
\psi_1 &= \mu X.\langle invokegetInt\rangle\ \psi_2 \vee \langle -invokegetInt\rangle X\\
\psi_2 &= \mu X.\langle store\rangle\ \psi_3 \vee \langle -store\rangle X\\
\psi_3 &= \mu X.\langle load\rangle\ \psi_4 \vee \langle -load\rangle X\\
\psi_4 &= \mu X.\langle pushconsttaskidsendsms\rangle\ \psi_5 \vee \langle -pushconsttaskidsendsms\rangle X\\
\psi_5 &= \mu X.\langle invokegetInt\rangle\ \psi_6 \vee \langle -invokegetInt\rangle X\\
\psi_6 &= \mu X.\langle store\rangle\ \psi_7 \vee \langle -store\rangle X\\
\psi_7 &= \mu X.\langle pushreceiverStatusSms\rangle\ \psi_8 \vee \langle -pushreceiverStatusSms\rangle X\\
\psi_8 &= \mu X.\langle pushsmsId\rangle\ \psi_9 \vee \langle -pushsmsId\rangle X\\
\psi_9 &= \mu X.\langle pushsmsTaskId\rangle\ \psi_{10} \vee \langle -pushsmsTaskId\rangle X\\
\psi_{10} &= \mu X.\langle pushgetResultCode\rangle\ \mathtt{tt} \vee \langle -pushgetResultCode\rangle X
\end{aligned}
$$

Malicious Behaviour in mu-calculus formulae

The Experiment on the Overlay family

```
public void onReceive(Context var1, Intent var2){
  Bundle var5=var2.getExtras();
  int var3=var5.getInt('const_id_send_sms');
  int var4=var5.getInt('const_task_id_send_sms');
 M.d('receiverStatusSms', 'smsId: ' + var3 \
    + "; smsTaskId: " + var4 \
    + "; getResultCode(): " \
    + this.getResultCode());
  switch(this.getResultCode()) {
  case -1:
  if (var4 >= 0) {
    SocketService.access$000(this.this$0, var4);
  }
  SocketService.access$100(this.this$0, var3, \
    Const._SMS_STATUS_SEND);
```

Collecting User Info

Send Info to attackers

Malicious Behaviour in Java Code

$$
\begin{aligned}
\psi &= \mu X.\langle pushconstidsendsms\rangle\ \psi_1 \vee \langle -pushconstidsendsms\rangle X\\
\psi_1 &= \mu X.\langle invokegetInt\rangle\ \psi_2 \vee \langle -invokegetInt\rangle X\\
\psi_2 &= \mu X.\langle store\rangle\ \psi_3 \vee \langle -store\rangle X\\
\psi_3 &= \mu X.\langle load\rangle\ \psi_4 \vee \langle -load\rangle X\\
\psi_4 &= \mu X.\langle pushconsttaskidsendsms\rangle\ \psi_5 \vee \langle -pushconsttaskidsendsms\rangle X\\
\psi_5 &= \mu X.\langle invokegetInt\rangle\ \psi_6 \vee \langle -invokegetInt\rangle X\\
\psi_6 &= \mu X.\langle store\rangle\ \psi_7 \vee \langle -store\rangle X\\
\psi_7 &= \mu X.\langle pushreceiverStatusSms\rangle\ \psi_8 \vee \langle -pushreceiverStatusSms\rangle X\\
\psi_8 &= \mu X.\langle pushsmsId\rangle\ \psi_9 \vee \langle -pushsmsId\rangle X\\
\psi_9 &= \mu X.\langle pushsmsTaskId\rangle\ \psi_{10} \vee \langle -pushsmsTaskId\rangle X\\
\psi_{10} &= \mu X.\langle pushgetResultCode\rangle\ \mathtt{tt} \vee \langle -pushgetResultCode\rangle X
\end{aligned}
$$

Malicious Behaviour in mu-calculus formulae

The Experiment on the Overlay family

```
public void onReceive(Context var1, Intent var2){
  Bundle var5=var2.getExtras();
  int var3=var5.getInt('const_id_send_sms');
  int var4=var5.getInt('const_task_id_send_sms');
 M.d('receiverStatusSms', 'smsId: ' + var3 \
    + "; smsTaskId: " + var4 \
    + "; getResultCode(): " \
    + this.getResultCode());
  switch(this.getResultCode()) {
  case -1:
  if (var4 >= 0) {
    SocketService.access$000(this.this$0, var4);
  }
  SocketService.access$100(this.this$0, var3, \
    Const._SMS_STATUS_SEND);
```

Collecting User Info

Send Info to attackers

Malicious Behaviour in Java Code

$$
\begin{aligned}
\psi &= \mu X.\langle pushconstidsendsms\rangle\ \psi_1 \vee \langle -pushconstidsendsms\rangle X\\
\psi_1 &= \mu X.\langle invokegetInt\rangle\ \psi_2 \vee \langle -invokegetInt\rangle X\\
\psi_2 &= \mu X.\langle store\rangle\ \psi_3 \vee \langle -store\rangle X\\
\psi_3 &= \mu X.\langle load\rangle\ \psi_4 \vee \langle -load\rangle X\\
\psi_4 &= \mu X.\langle pushconsttaskidsendsms\rangle\ \psi_5 \vee \langle -pushconsttaskidsendsms\rangle X\\
\psi_5 &= \mu X.\langle invokegetInt\rangle\ \psi_6 \vee \langle -invokegetInt\rangle X\\
\psi_6 &= \mu X.\langle store\rangle\ \psi_7 \vee \langle -store\rangle X\\
\psi_7 &= \mu X.\langle pushreceiverStatusSms\rangle\ \psi_8 \vee \langle -pushreceiverStatusSms\rangle X\\
\psi_8 &= \mu X.\langle pushsmsId\rangle\ \psi_9 \vee \langle -pushsmsId\rangle X\\
\psi_9 &= \mu X.\langle pushsmsTaskId\rangle\ \psi_{10} \vee \langle -pushsmsTaskId\rangle X\\
\psi_{10} &= \mu X.\langle pushgetResultCode\rangle\ \mathtt{tt} \vee \langle -pushgetResultCode\rangle X
\end{aligned}
$$

Collecting User Info (ψ – ψ_7)

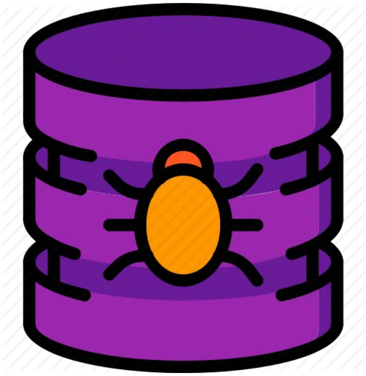
Send Info to attackers (ψ_8 – ψ_{10})

Malicious Behaviour in mu-calculus formulae

The Dataset

+ 75 malware Overlay family
+ 250 malware from Drebin [1]*
+ 50 trusted samples

= 375 real world samples



*25 randomly selected samples from each of the top 10 Drebin Malware Families

[1] ARP, Daniel, et al. Drebin: Effective and explainable detection of android malware in your pocket. In: Ndss. 2014.

Evaluation Result

#Malware $\in$ Overlay	#Malware $\notin$ Overlay	#Trusted
75	250	50

True Positive	*False Positive*	*False Negative*	*True Negative*
75	**0**	**0**	**300**

Take-home points

Short experimental paper: applied known technique[1,2] on a specific malware classification problem

- Methodology:
  - ➢ model checking to detect Overlay malware
- Database:
  - ➢ 350 real world applications
- Experiment result:
  - ➢ achieved precision and recall values equal to 1

[1] Canfora, Gerardo, et al. "Leila: formal tool for identifying mobile malicious behaviour." *IEEE Transactions on Software Engineering* (2018)
[2] Cimitile, Aniello, et al. "Talos: no more ransomware victims with formal methods." *International Journal of Information Security* 17.6 (2018)

Limitations and Future Works

- Extend analysis to more malware (families)
  - ➢ Image classification and Deep Learning

- Take into account obfuscation
  - ➢ Check robustness model

- Using preliminary static analysis to automatize malicious behaviour extraction (GrapPa)

Research topics and publications

- **Software Testing and Analysis**
  - *Graph-based classification for detecting instances of bug patterns* → Master's degree thesis TU Darmstadt

- **Mobile Security (Android OS)**
  - *Improving robustness and efficiency in malware classification* → Work in progress with F. Mercaldo
  - *Formal Methods for Android Banking Malware Analysis and Detection* → Published IOTSMS19

- **Machine Learning** (towards Adversarial Learning)
  - *Image-based Malware Family Detection: An Assessment between Feature Extraction and Classification Techniques* → submitted IoTBDS20

Thanks for the attention

Questions?

References

Literature for specifying malware behaviours as logic property:

[1] Wei, Fengguo, et al. "Deep ground truth analysis of current android malware." *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Cham, 2017.

[2] Han, Qian, et al. "DBank: Predictive Behavioral Analysis of Recent Android Banking Trojans." *IEEE Transactions on Dependable and Secure Computing* (2019).

[3] Wazid, Mohammad, Sherali Zeadally, and Ashok Kumar Das. "Mobile banking: evolution and threats: malware threats and security solutions." *IEEE Consumer Electronics Magazine* 8.2 (2019)

[4] Pan, Jordan “Fake Bank App Ramps Up Defensive Measures“ http://tiny.cc/xz209y

Applied techniques based on Formal Methods:

[5] Canfora, Gerardo, et al. "Leila: formal tool for identifying mobile malicious behaviour." *IEEE Transactions on Software Engineering* (2018)

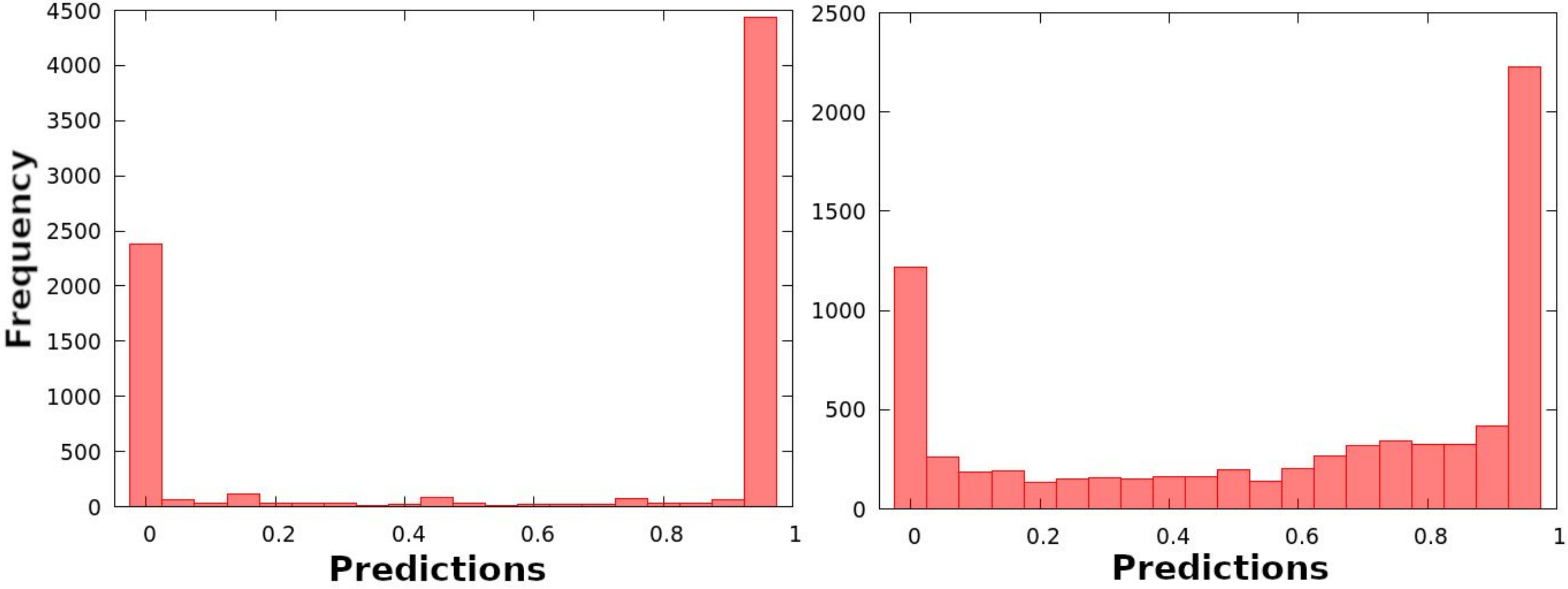
[6] Cimitile, Aniello, et al. "Talos: no more ransomware victims with formal methods." *International Journal of Information Security* 17.6 (2018)

Database:

[7] ARP, Daniel, et al. Drebin: Effective and explainable detection of android malware in your pocket. In: Ndss. 2014. p. 23-26.

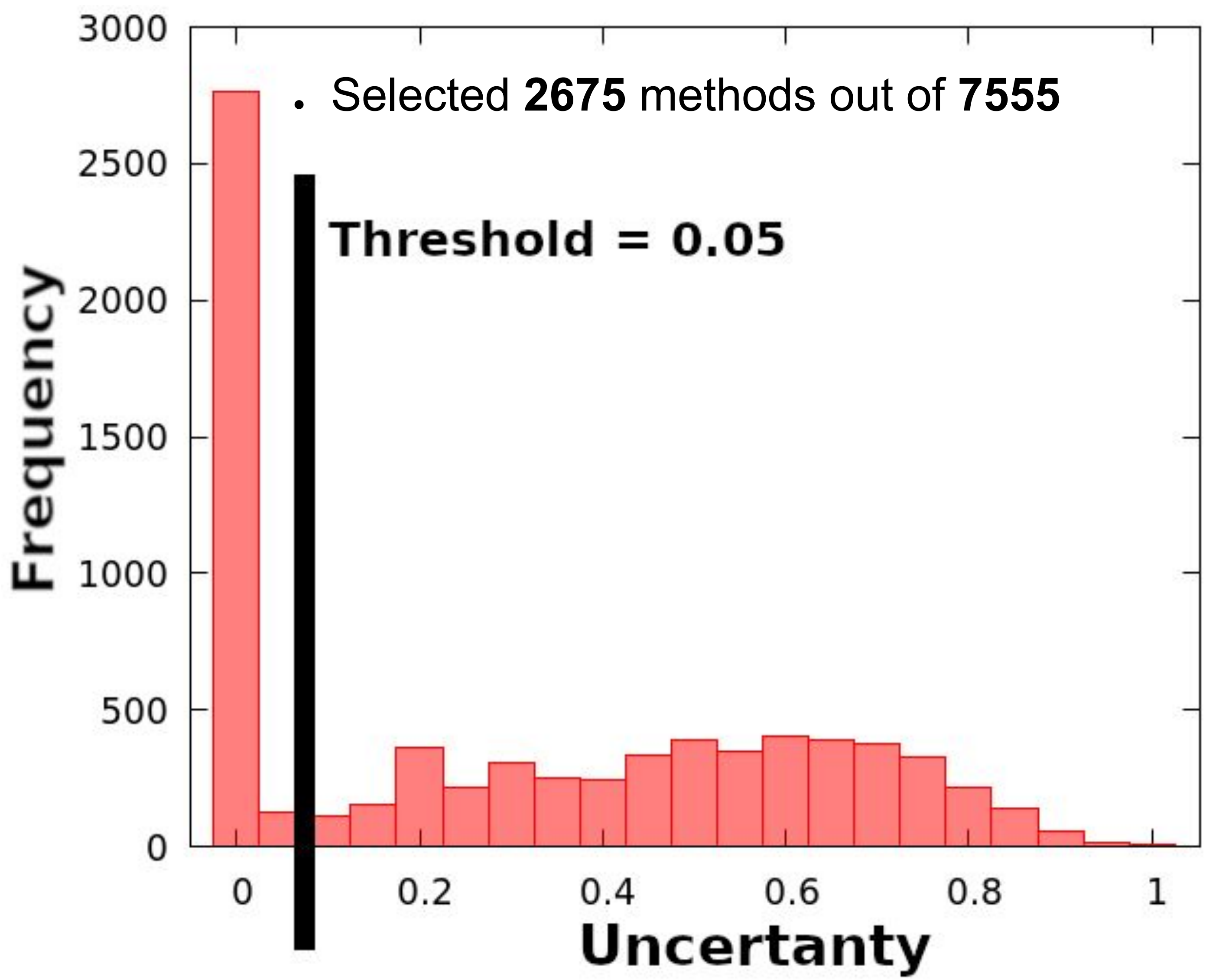
Results GrapPa

JfreeChart project as test dataset (7555 methods)



Frequency of predictions without dropout (on the left) and the average of predictions with dropout (on the right).

Results GrapPa



Results GrapPa

Manually checked **80** methods of the **2675** selected by the tool

- **40** buggy predictions
- **40** non-buggy predictions

We agreed with the tool predictions in **70%** of the cases.

PREDICTION	**AGREED** with the model	**NOT AGREED** with the model
(**1**) Possible NPE	**60%** **23 cases**	**40%** **17 cases**
(**0**) NPE not-possible	**80%** **32 cases**	**20%** **8 cases**

Result manual check

Intercepting SMS messages behaviour

```
public void onReceive(Context var1,Intent var2){
  String var3 = var2.getAction();
  this.app = \
    (Application)var1.getApplicationContext();
  if(var3.equals( \
    'android.intent.action.BOOT_COMPLETED')){
   M.d('Receiver', 'ACTION_BOOT_COMPLETED');
    this.app.startSocket();
  }else if(var3.equals("alarm_check_connected")){
    this.cardActivity(var1);
    this.app.startSocket();
  }else if(var3.equals( \
    'android.provider.Telephony.SMS_RECEIVED')){
   M.d('Receiver', \
      'android.provider.Telephony.SMS_RECEIVED');
    (new Thread(new 1(this, var2))).start();
    try {
      this.abortBroadcast();
    } catch (Exception var4){
      var4.printStackTrace();
    }
  }
}
```

$$\chi = \mu X.\langle pushandroidintentactionBOOTCOMPLETED\rangle\ \chi_1 \vee \langle -pushandroidintentactionBOOTCOMPLETED\rangle\ X$$

$$\chi_1 = \mu X.\langle pushandroidproviderTelephonySMSRECEIVED\rangle\ \chi_2 \vee \langle -pushandroidproviderTelephonySMSRECEIVED\rangle\ X$$

$$\chi_2 = \mu X.\langle invokeequals\rangle\ \chi_3 \vee \langle -invokeequals\rangle\ X$$

$$\chi_3 = \mu X.\langle pushReceiver\rangle\ \chi_4 \vee \langle -pushReceiver\rangle\ X$$

$$\chi_4 = \mu X.\langle newjavalangThread\rangle\ \chi_5 \vee \langle -newjavalangThread\rangle\ X$$

$$\chi_5 = \mu X.\langle load\rangle\ \chi_6 \vee \langle -load\rangle\ X$$

$$\chi_7 = \mu X.\langle invokeinit\rangle\ \chi_8 \vee \langle -invokeinit\rangle\ X$$

$$\chi_8 = \mu X.\langle invokestart\rangle\ \chi_9 \vee \langle -invokestart\rangle\ X$$

$$\chi_9 = \mu X.\langle invokeabortBroadcast\rangle\ \mathtt{tt} \vee \langle -invokeabortBroadcast\rangle\ X$$

Password resetting behaviour

```
void showReq() {
  this.deviceManger = (DevicePolicyManager)
    this.getSystemService('device_policy');
  this.deviceAdmin = new ComponentName(this,
   MyAdmin.class);
  Intent var1 = new Intent( \
    'android.app.action.ADD_DEVICE_ADMIN');
  var1.putExtra('android.app.extra.' + \
    'DEVICE_ADMIN', this.deviceAdmin);
  var1.putExtra('android.app.extra.' + \
    'ADD_EXPLANATION','Explanation Message';
  this.startActivityForResult(var1, 1);
  if (this.deviceManger.isAdminActive( \
    this.compName)){
    this.app.setAdmin(1);
  }
}
```

$$\zeta = \mu X.\langle pushdevicepolicy\rangle\ \zeta_1 \vee \langle -pushdevicepolicy\rangle X$$

$$\zeta_1 = \mu X.\langle invokegetSystemService\rangle\ \zeta_2 \vee \langle -invokegetSystemService\rangle X$$

$$\zeta_2 = \mu X.\langle checkcastandroidappadminDevicePolicyManager\rangle\ \zeta_3 \vee \langle -checkcastandroidappadminDevicePolicyManager\rangle X$$

$$\zeta_3 = \mu X.\langle pushandroidappactionADDDEVICEADMIN\rangle\ \zeta_4 \vee \langle -pushandroidappactionADDDEVICEADMIN\rangle X$$

$$\zeta_4 = \mu X.\langle pushandroidappextraDEVICEADMIN\rangle\ \zeta_5 \vee \langle -pushandroidappextraDEVICEADMIN\rangle X$$

$$\zeta_5 = \mu X.\langle invokeputExtra\rangle\ \mathtt{tt} \vee \langle -invokeputExtra\rangle X$$

Why Formal Methods?

- The checking process is automatic, there is no need to construct a correctness proof
- The possibility of using the diagnostic counterexamples
- Temporal logic can easily and correctly express the behaviour of a malware
- Formal verification allows evaluating all possible scenarios, the entire state space all at once