SYSMODIS

SYSTEMATIC MODEL DISCOVERY APPROACH

10th International Workshop on Combinatorial Testing (IWCT 2021)

Omer Korkmaz and Cemal Yilmaz

AGENDA

01 Introduction

02 Approach

03 Experiments

04 Conclusion & Future Works

1

INTRODUCTION

In today's technology, mobile devices and applications have increasingly become smarter and more powerful.

Used by millions of people

In daily life, people are using mobile applications from various categories

- education,
- health,
- economy,
- or management

Started to be more complex

While providing solutions for the demands of the users, the mobile apps started to have huge, complex and interactive logics and functionalities.

2.560.000 Android Apps

https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

OPS, FAILURES!

- The complexity may cause **more failures** in the applications.
  - lack of business knowledge, wrong implementation, etc.
- Users are exposed to those failures **many times** using the mobile apps.

So, these applications need to be tested thoroughly.

The model of an application plays a significant role to represent the test strategies while the system is under test (SUT).

CONTRIBUTION

COVERING ARRAYS → SYSTEMATIC SAMPLING → MODEL DISCOVERY

MODEL DISCOVERY

Crawl the screens by interacting with UI, generate test values for the inputs, discover the model and execute the tests **systematically** and **automatically**.

PREDICTION

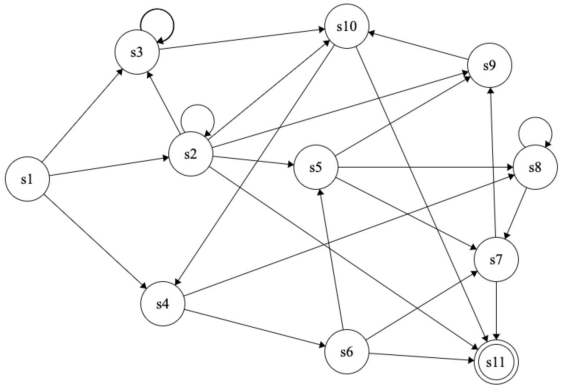
Discover likely guard-conditions **systematically** with the UI interactions, leveraging machine learning approaches for predicting guard-conditions

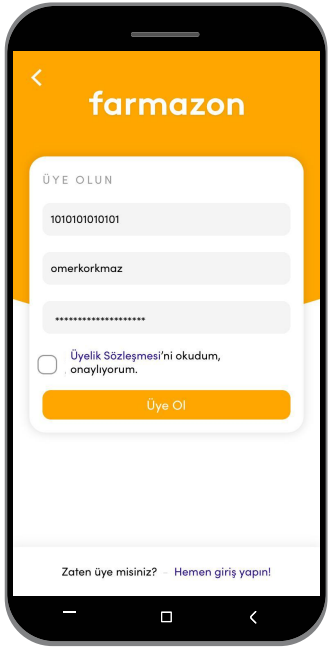
RESULTS

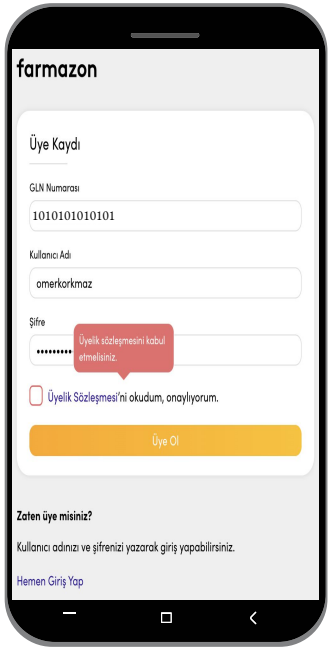
Achieved **%30** more code coverage when compared to other approaches. Achieved high accuracy on predictions when compared to random testing.

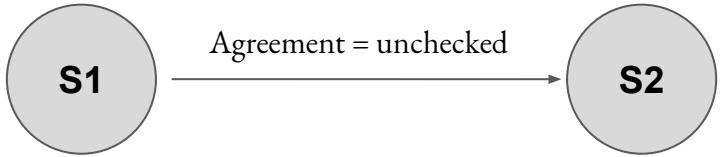
WHAT IS MODEL?

A state machine





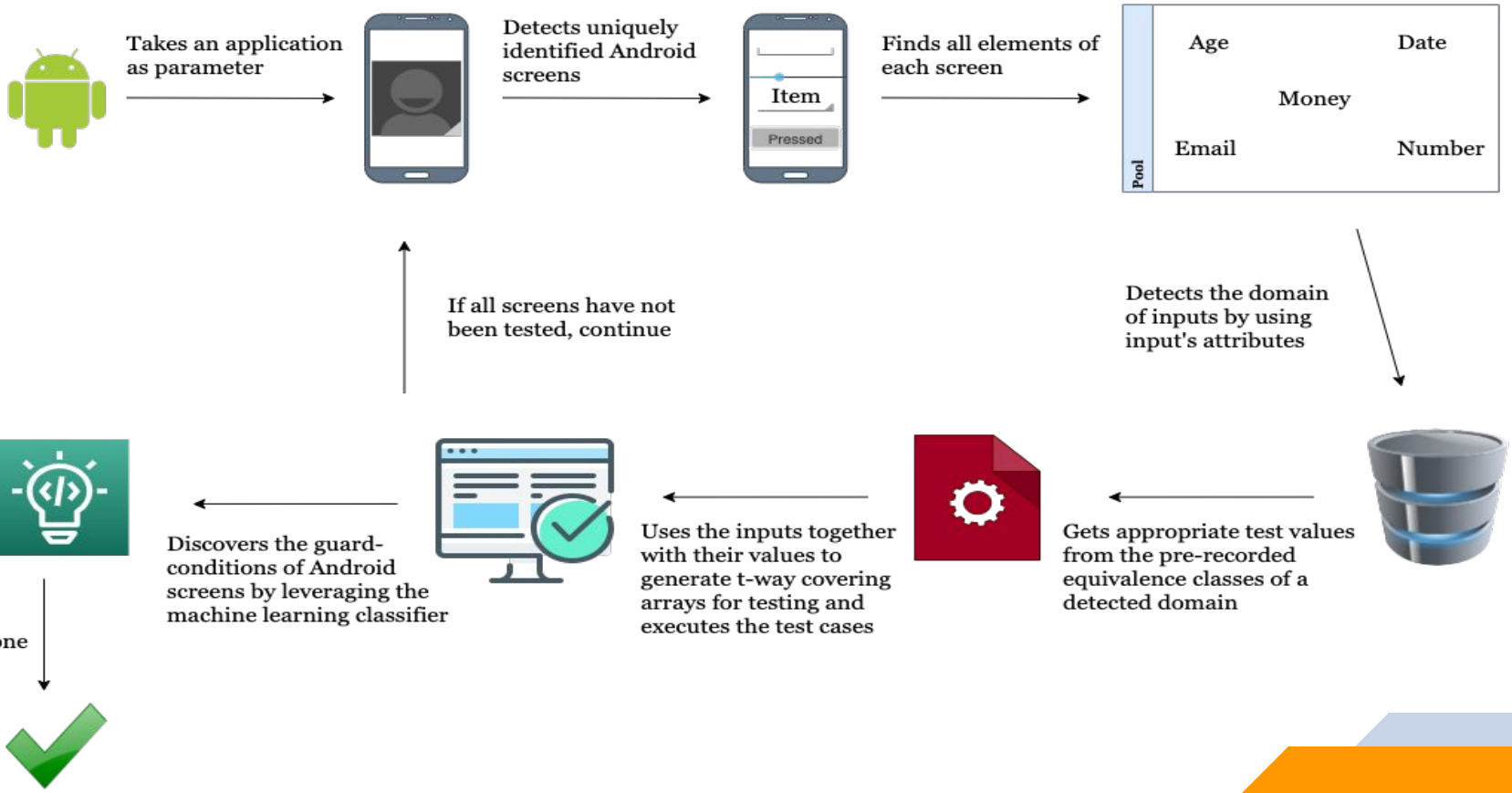




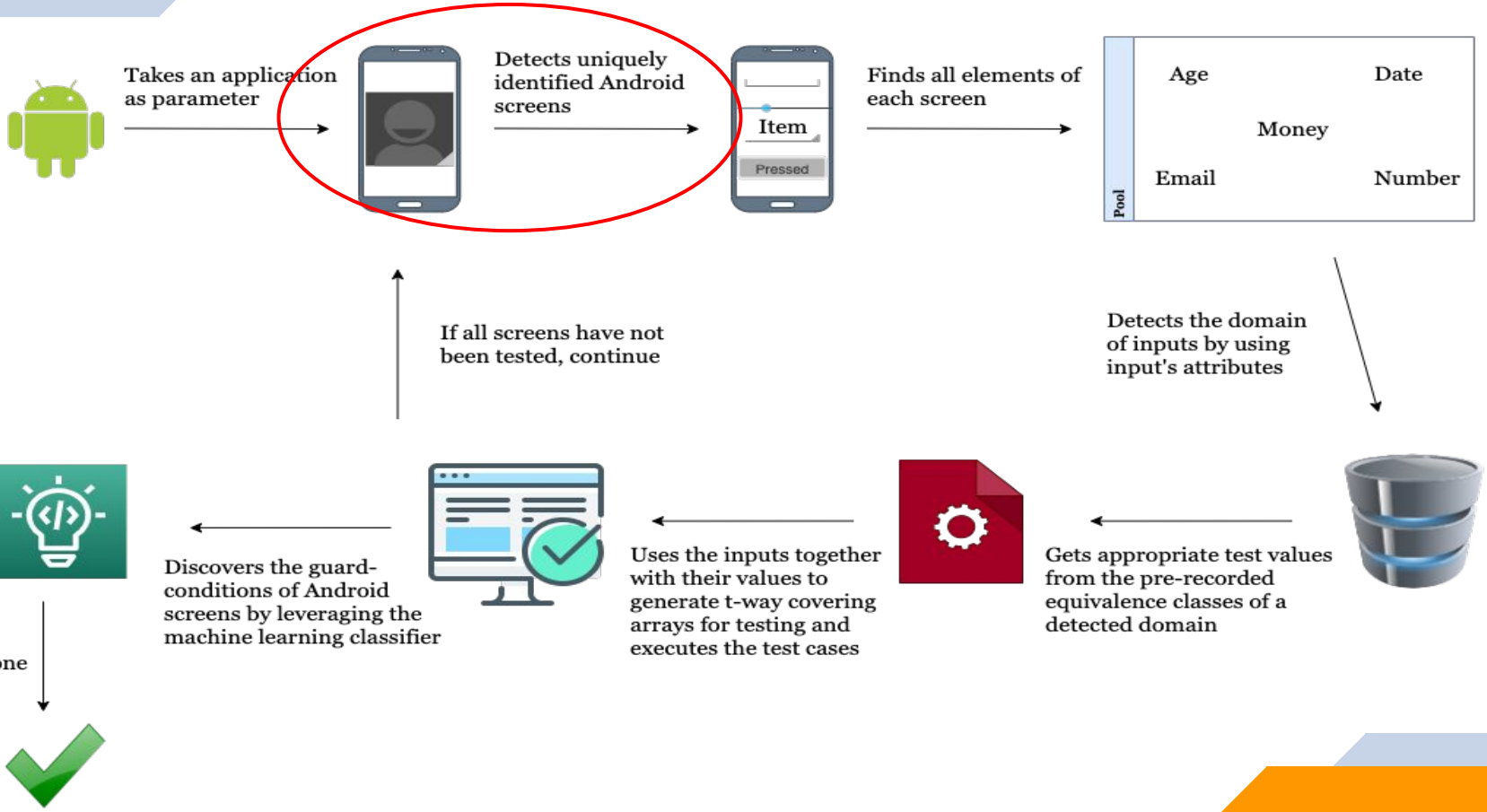
2

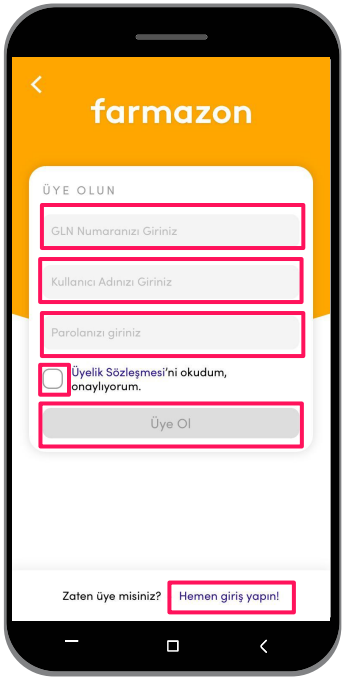
APPROACH

SYSMODIS OVERVIEW



SCREEN DETECTION

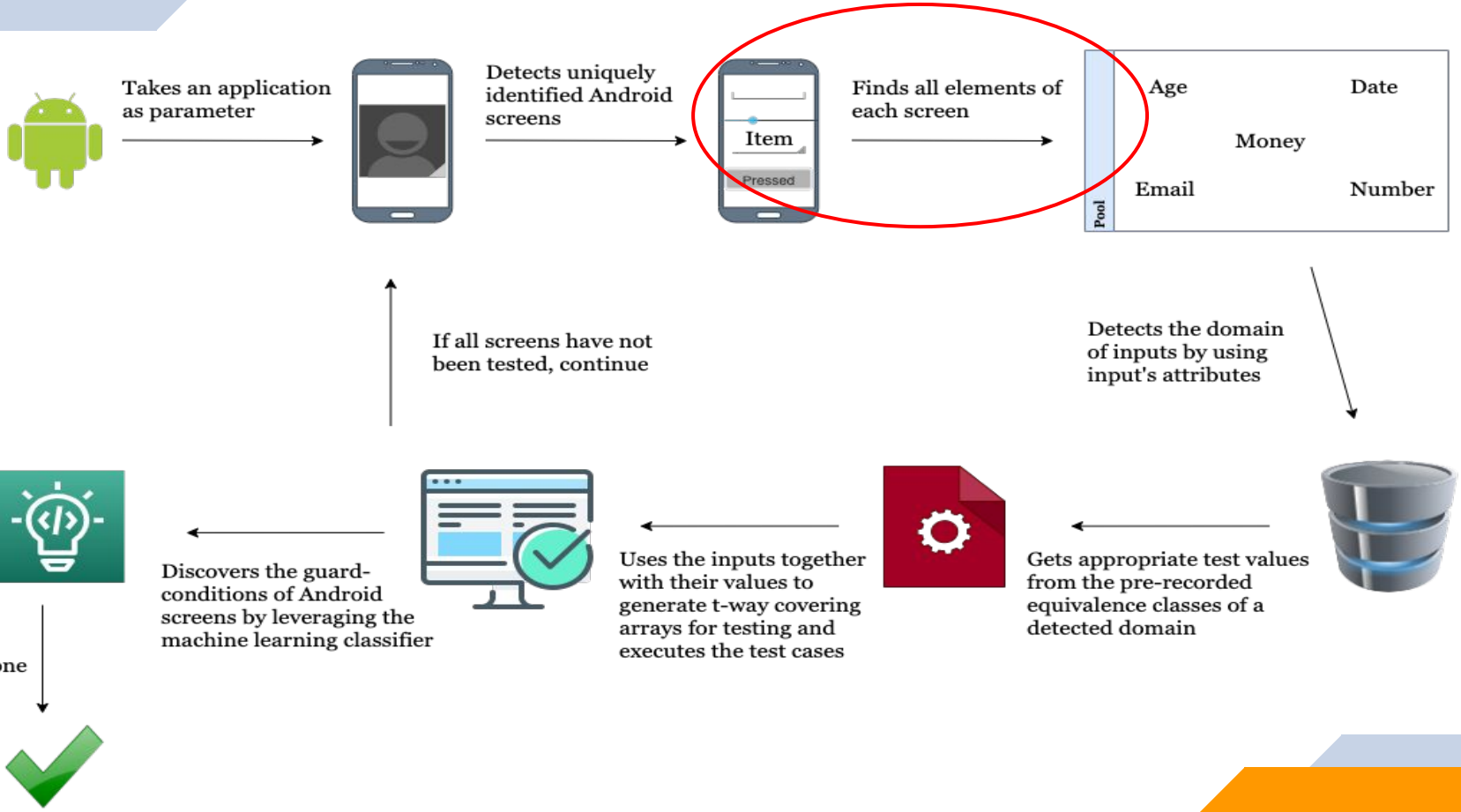




Screen Detection

- Get the XML file of current Android screen.
- Catch the Android elements from the XML.
- Take the attributes of each element.
- Hash the screen elements in an ordered agnostic way
- Check the distinctness comparing hash values!
- Store all information in the database.

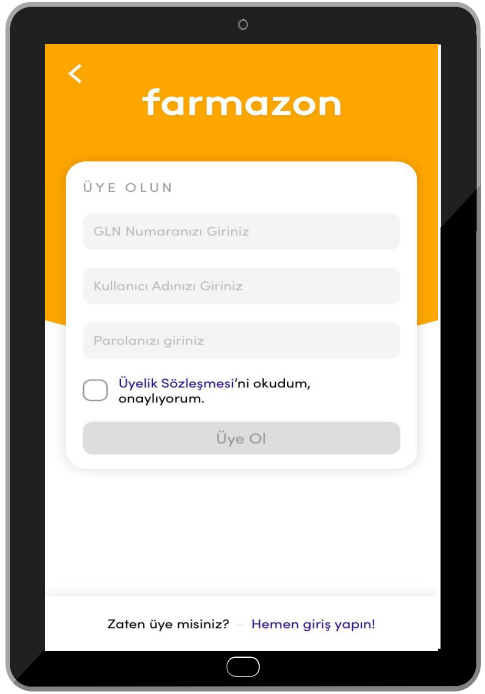
INPUT DETECTION



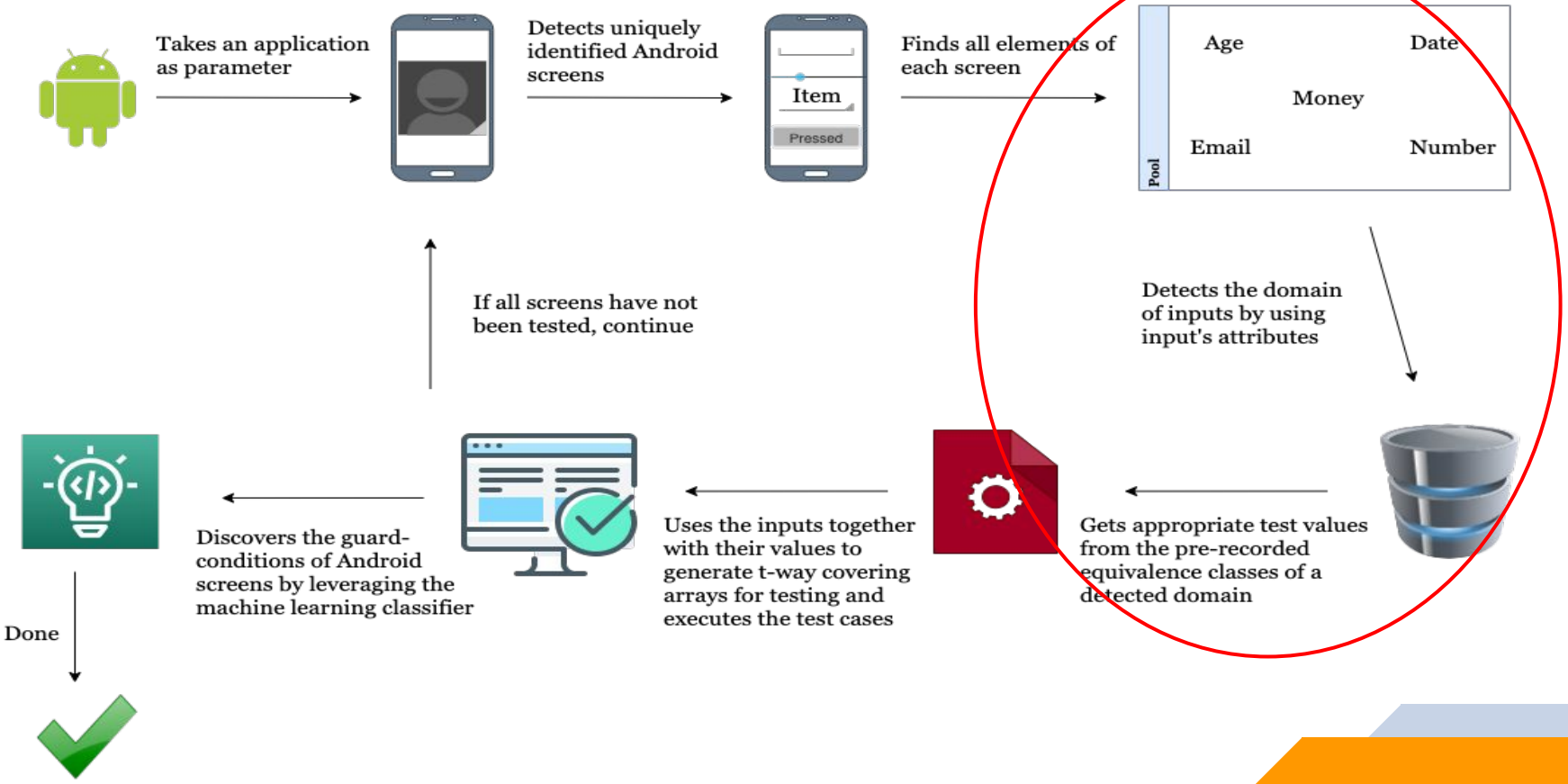
Attributes	Sample Attribute Values
Class Name	*android.widget.Button*
Resource-id	*com.sample.android:id/LoginButton*
Text	*Login*
Content-desc	*Login the system*
Clickable	*True*
Long-Clickable	*True*
Checkable	*False*
Scrollable	*False*
Bounds	*[10,360][172,426]*

Input Detection

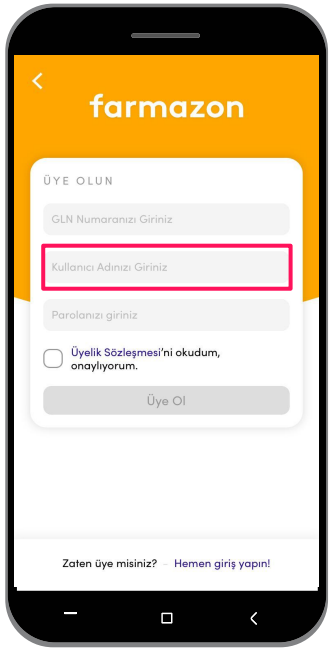
- Get each element from the current screen.
- Get the **actionable attributes** of each element.
- Check the attributes and determine the input type of each element.
- For instance;
  - GLN, Username, Password > Editable
  - Agreement > Checkable
  - Register, Login > Clickable



DOMAIN DETECTION



DOMAIN DETECTION



ATTRIBUTES

content-desc
"type your email"
text
"please enter your email"
resource-id
com.farmazon.id:/emailText

EMAIL DOMAIN

keywords
mail
e-mail
username
...

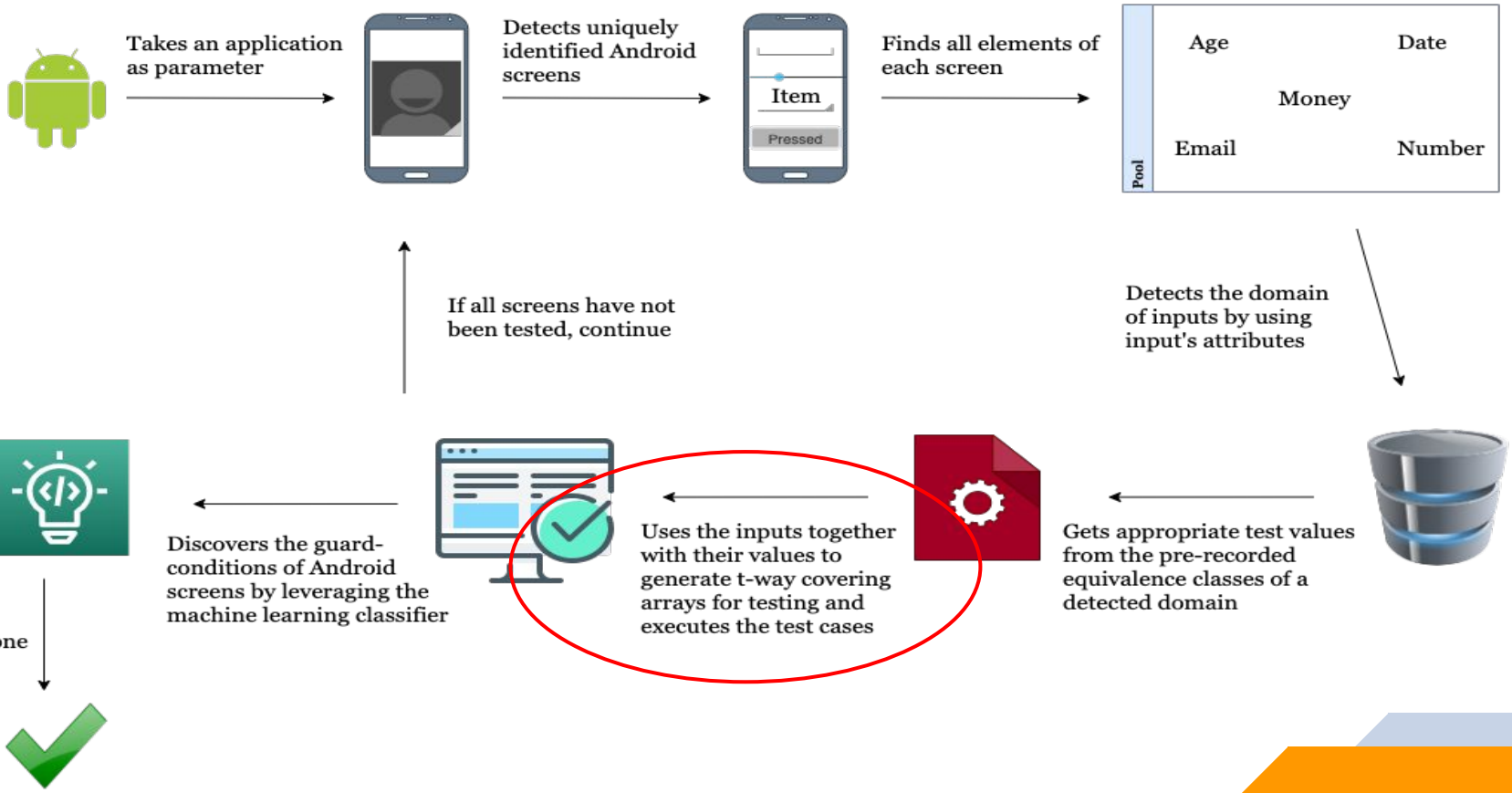
EQUIVALENCE CLASSES

Valid Email
omer@gmail.com

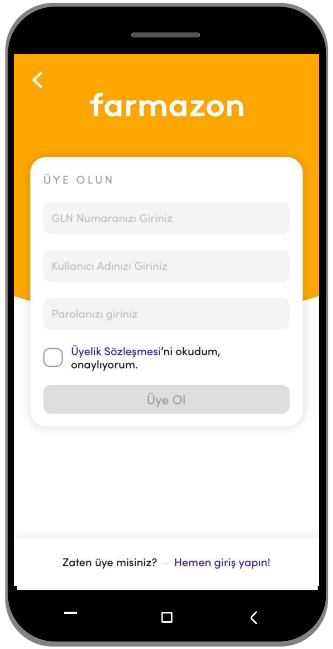
Invalid Email
qy@1?1.com

Semantic Similarity?

CA GENERATION

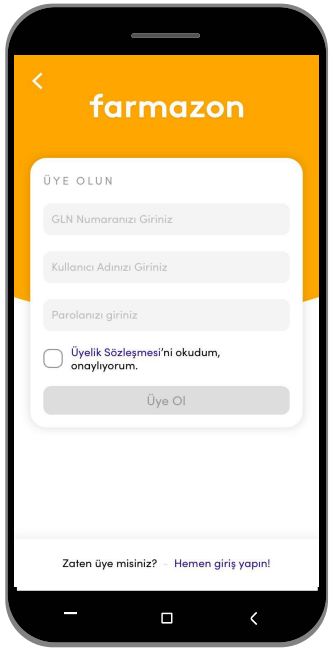


CA GENERATION



Options	Settings
O_1 (GLN)	$< 100716, 000000 >$
O_2 (Username)	$< omer@hotmail.com, qy@11.com >$
O_3 (Password)	$< Passw0rd!, ???? >$
O_4 (Agreement)	$< Checked, Unchecked >$
O_5 (Actions)	$< LoginButton, RegisterButton >$

CA GENERATION



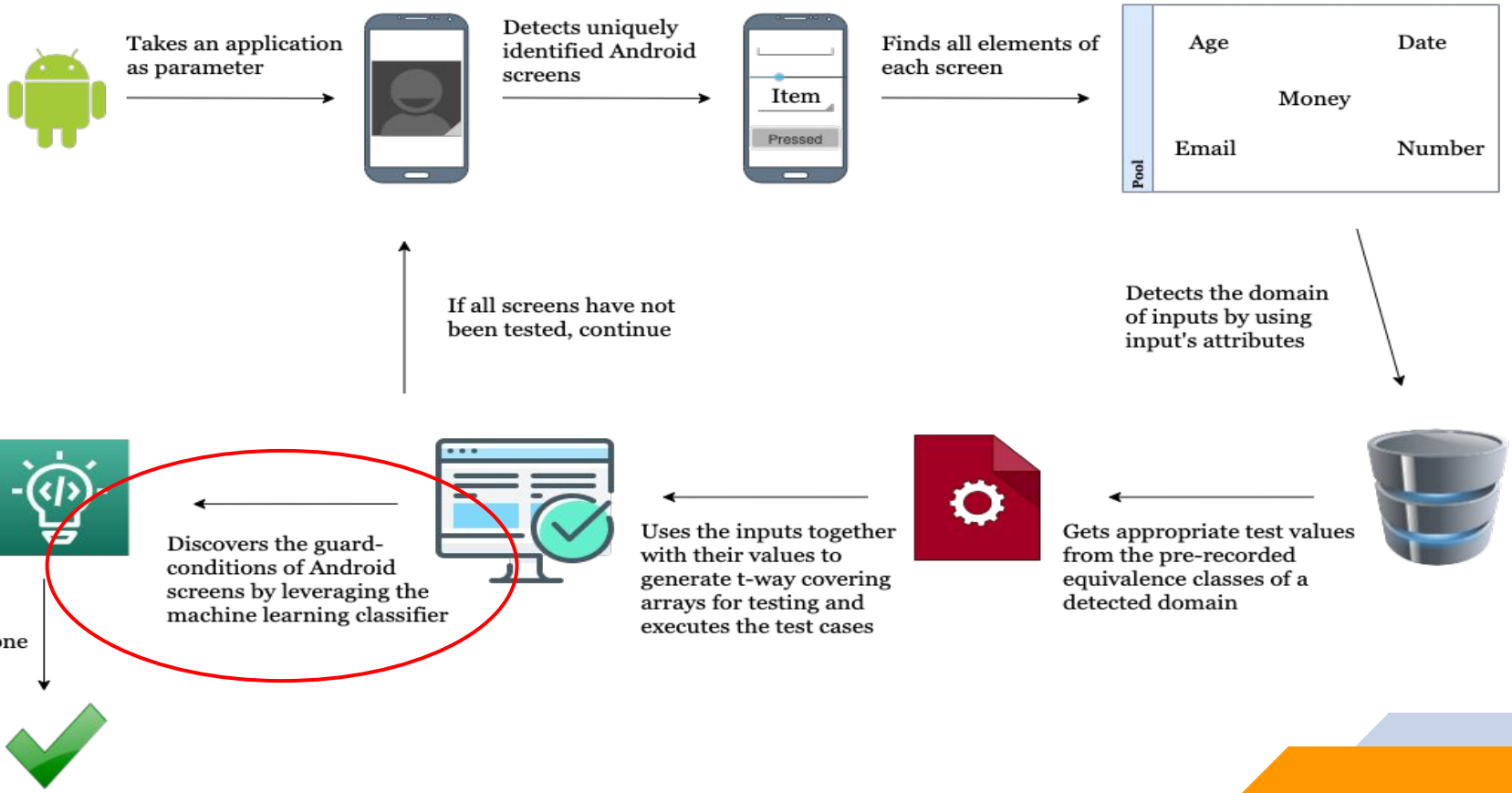
GLN	Username	Password	Agreement	Action
SET 1007716	SET omer@hotmail.com	SET Passw0rd!	SET checked	CLICK Register Button
SET 1007716	SET omer@hotmail.com	SET ????	SET unchecked	CLICK Register Button
SET 1007717	SET qy@11.com	SET Passw0rd!	SET unchecked	CLICK Register Button
SET 1007718	SET qy@11.com	SET ????	SET checked	CLICK Register Button
SET 000000	SET omer@hotmail.com	SET Passw0rd!	SET unchecked	CLICK Register Button
SET 000001	SET omer@hotmail.com	SET ????	SET checked	CLICK Register Button
SET 000002	SET qy@11.com	SET Passw0rd!	SET checked	CLICK Register Button
SET 000003	SET qy@11.com	SET ????	SET unchecked	CLICK Register Button
SET 1007716	SET omer@hotmail.com	SET Passw0rd!	SET checked	CLICK Login Button
SET 1007716	SET omer@hotmail.com	SET ????	SET unchecked	CLICK Login Button
SET 1007717	SET qy@11.com	SET Passw0rd!	SET unchecked	CLICK Login Button
SET 1007718	SET qy@11.com	SET ????	SET checked	CLICK Login Button
SET 000000	SET omer@hotmail.com	SET Passw0rd!	SET unchecked	CLICK Login Button
SET 000001	SET omer@hotmail.com	SET ????	SET checked	CLICK Login Button
SET 000002	SET qy@11.com	SET Passw0rd!	SET checked	CLICK Login Button
SET 000003	SET qy@11.com	SET ????	SET unchecked	CLICK Login Button

Covering Array (***t*:3**)
Each row => Test case
Each cell => Test action

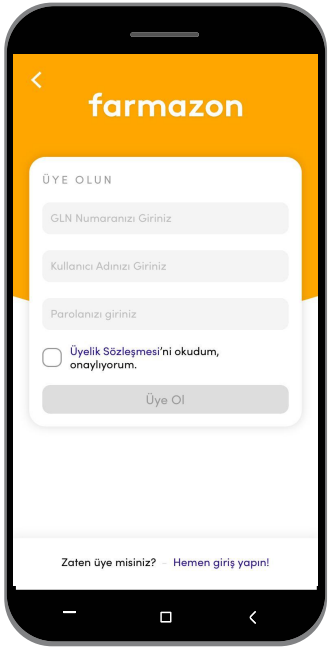
CRAWLING

- Opportunistic crawling
  - Screen is visited ⟶ previously untested test case executed
  - Test suite executed ⟶ move nearest state with untested test cases and execute
- Continues to execute tests where it left off
  - In case of any failure, system exception, etc.
- Configured to restart the system under test
  - Preventing crawling from getting stuck as much as possible

GC DISCOVERY



S2: Agreement Error Screen, **S3:** Login Screen, **S4:** Other Errors Screen

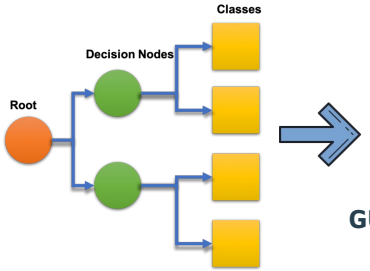


GLN	Username	Password	Agreement	Action	Target
SET 1007716	SET omer@hotmail.com	SET Passw0rd!	SET checked	CLICK Register Button	S3
SET 1007716	SET omer@hotmail.com	SET ????	SET unchecked	CLICK Register Button	S2
SET 1007716	SET qy@11.com	SET Passw0rd!	SET unchecked	CLICK Register Button	S2
SET 1007716	SET qy@11.com	SET ????	SET checked	CLICK Register Button	S4
SET 000000	SET omer@hotmail.com	SET Passw0rd!	SET unchecked	CLICK Register Button	S2
SET 000000	SET omer@hotmail.com	SET ????	SET checked	CLICK Register Button	S4
SET 000000	SET qy@11.com	SET Passw0rd!	SET checked	CLICK Register Button	S4
SET 000000	SET qy@11.com	SET ????	SET unchecked	CLICK Register Button	S2
SET 000003	SET qy@11.com	SET ????	SET unchecked	CLICK Register Button	S2

1 FOR EACH TARGET STATE → ITSELF

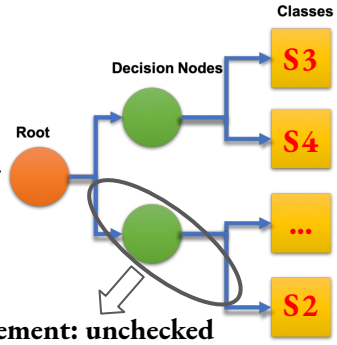
0 → OTHERS

BINARY CLASSIFICATION



PREDICTED GUARD-CONDITION

EXAMPLE



Agreement: unchecked

3

EXPERIMENTS

EXPERIMENTS

Study 1

Evaluating sensitivity to model parameters

Evaluated the sensitivity of the approach to various model parameters, including the number of states, density, the level of determinism, and the complexity of the guard conditions.

To this end, used **simulations** as it was not possible to systematically vary these parameters on real subject apps.

Study 2

Evaluations on Subject Applications

Evaluated the proposed approach by conducting comparative studies using **real subject applications**.

Also, applied **random-testing** with the proposed approach and compared it with other approaches.

- **states:** the number of states in the model.
- **density:** the density of the model which is used to compute the number of transitions in the model.
- **parameters:** the number of parameters defined in a state, i.e., the number of input fields on a screen.
- **settings:** the number of equivalence classes for a parameter.
- **guard-complexity:** the number of distinct parameters involved in a guard condition associated with a transition.
- **t:** the coverage strength of the covering arrays used for sampling.
- **determinism:** the level of determinism in the model, depicting the probability of taking a transition given that the guard condition of the transition is satisfied. When determinism = 1.0, all the transitions are deterministic given a transition, when the system is currently in the source state and the guard condition of the transition is satisfied, the transition is guaranteed to be taken and the system moves to the target state.

Table 4.1 Model parameters manipulated in the experiments.

Parameter	Values
number of states	$\{10, 20, 50\}$
density	$\{0.4, 0.6\}$
number of parameters per state	$\{[5, 10], [16, 20]\}$
number of equivalence classes per input	$\{[3, 6]\}$
number of distinct parameters involved in guard conditions	$\{1, 2, 3, 4, [1, 5]\}$
covering array strength	$\{2, 3, 4\}$
level of determinism	$\{0, 0.01, 0.05, 0.1\}$

FOR EACH CONFIG

100

STATE MACHINES

STUDY 1: EVALUATION

1

State Coverage

percentage of the states visited

2

Transition Coverage

percentage of the transitions satisfied

3

Accuracy

accuracy of the guard conditions predicted

4

Machine

I7 6700HQ, 16 GB RAM, Windows 10

5

Decision Tree Classifier

The classification models were trained and the classifier was taken from **scikit-learn**

6

ACTS

Covering arrays were generated.

STUDY 1: ANALYSIS

TABLE II
AVERAGE RESULTS OBTAINED FROM ALL THE MODELS USED IN THE EXPERIMENTS.

t	Screen Coverage	Transition Coverage	Accuracy
2	83.10%	82.28%	68.07%
3	91.90%	91.48%	73.12%
4	100.00%	100.00%	77.20%

TABLE III
EFFECT OF NON-DETERMINISM WHEN $t >$ guard $-$ condition.

Non-determinism	State Cov.	Transition Cov.	Accuracy
0.00	100.00%	100.00%	81.89%
0.01	98.42%	98.51%	76.02%
0.05	96.04%	96.07%	72.63%
0.10	93.56%	92.23%	68.81%

TABLE IV
COMPARING THE PROPOSED APPROACH TO RANDOM TESTING WHEN non $-$ determinism $= 0$ AND $t >$ guard $-$ complexity.

Approach	State Coverage	Transition Coverage	Accuracy
SYSMODIS	100.00%	100.00%	81.89%
Random	71.34%	70.89%	73.78%

Fig. 2. State coverage statistics obtained when non-determinism = 0.

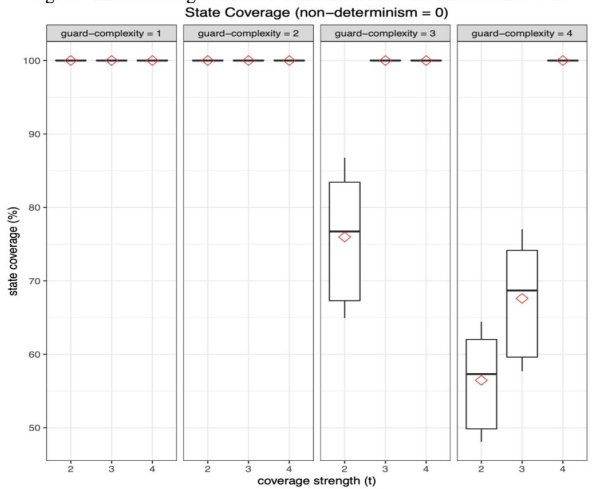


Fig. 3. Transition coverage statistics obtained when non-determinism = 0.

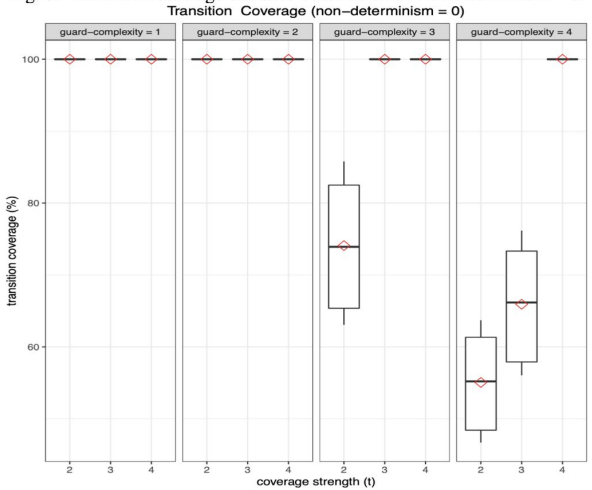
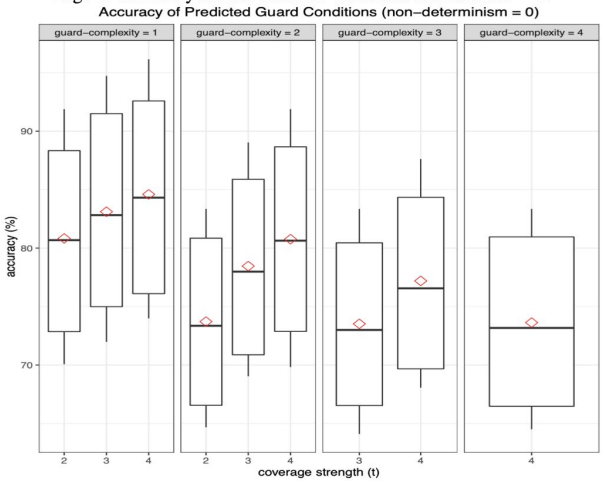


Fig. 4. Accuracy results obtained when non-determinism = 0.



STUDY 2: EVALUATION

- **screen:** page of Android mobile applications. It might be an **Android activity** or a **different page in the same activity** (e.g., pop-up, modal). Each page which consists of UI elements is called as screen.
- **test action:** one of the executable tests in test suites. For example, if we have a test suite that includes 3 executable tests, each of them is called as test action.

1. **Screen Coverage**
   percentage of the screens visited
2. **Code/Line Coverage**
   percentage of the source code statements visited
3. **Decision-Tree Classifier**
   classification models were trained
4. **ACTS**
   covering arrays were generated
5. **ACVTOOL**
   code coverage was measured

TABLE V
INFORMATION ABOUT THE SUBJECT APPLICATIONS USED IN THE STUDY.

SUT	Description	Total Screens
Tureng	Word translator	11
Tippy Tipper	Tip calculator	18
Munchlife	Counter/bookkeeper	5
Contact Manager	Contact manager	2
To Do Manager	To-do manager	7
Alarm Klock	Alarm clock	8
Habit App	Habit tracker	3
Any Memo	Foreign language learning aid	40
Weight Tracker	Weight tracker	15
BBC News	News channel	10

TABLE VII
ITEMIZED RESULTS OBTAINED IN THE EXPERIMENTS.

SUT	$t = 2$					$t = 3$					$t = 4$				
	# CA Size	# Test Cases	# Test Steps	Exec. Time	Screen Cov.	# CA Size	# Test Cases	# Test Steps	Exec. Time	Screen Cov.	# CA Size	# Test Cases	# Test Steps	Exec. Time	Screen Cov.
Tureng	70	95	224	25 min	78.25%	185	210	433	38 min	82.33%	297	330	600	54 min	84.88%
To Do Manager	33	52	111	34 min	100.00%	97	126	288	42 min	100.00%	182	215	410	63 min	100.00%
Tippy Tipper	21	38	54	15 min	100.00%	75	90	188	18 min	100.00%	121	150	310	31 min	100.00%
Munchlife	9	12	36	5 min	100.00%	19	25	67	11 min	100.00%	35	44	119	18 min	100.00%
Alarm Klock	28	33	75	28 min	100.00%	79	87	192	35 min	100.00%	160	198	352	43 min	100.00%
Habit App	34	41	123	37 min	77.88%	90	102	287	44 min	82.33%	176	194	389	56 min	85.33%
Weight Tracker	25	42	116	27 min	100.00%	67	78	195	32 min	100.00%	136	153	352	40 min	100.00%
BBC News	76	92	226	42 min	65.33%	151	183	334	52 min	70.33%	268	297	467	64 min	75.66%
Any Memo	30	38	105	26 min	100.00%	92	101	232	30 min	100.00%	179	192	311	36 min	100.00%
Contact Manager	37	45	147	30 min	100.00%	125	148	296	42 min	100.00%	222	267	520	55 min	100.00%

TABLE VI
COMPARING THE PROPOSED APPROACH TO RANDOM TESTING.

t	Approach	Screen Coverage	Code Coverage
2	SYSMODIS	93.37%	56.02%
3	SYSMODIS	93.37%	60.86%
4	SYSMODIS	93.37%	65.37%
2	Random	79.37%	44.02%
3	Random	81.33%	56.86%
4	Random	83.21%	58.35%

TABLE VIII
CODE COVERAGE STATISTICS OBTAINED IN THE PAPER.

SUT	$t = 2$				$t = 3$				$t = 4$			
	SYSMODIS	Random	Monkey	Dynodroid	SYSMODIS	Random	Monkey	Dynodroid	SYSMODIS	Random	Monkey	Dynodroid
Tureng	51%	43%	29%	38%	54%	47%	34%	42%	58%	49%	38%	44%
To Do Manager	64%	56%	36%	42%	66%	59%	39%	44%	68%	60%	41%	47%
Tippy Tipper	76%	62%	45%	-	79%	64%	49%	-	81%	67%	53%	-
Munchlife	74%	63%	42%	47%	76%	65%	44%	49%	80%	68%	49%	51%
Alarm Klock	63%	55%	40%	45%	65%	57%	43%	48%	69%	59%	47%	50%
Habit App	57%	46%	36%	28%	62%	48%	38%	31%	65%	50%	39%	33%
Weight Tracker	63%	57%	44%	46%	65%	59%	47%	48%	66%	60%	48%	50%
BBC News	49%	41%	26%	28%	52%	43%	28%	29%	53%	44%	29%	31%
Any Memo	50%	42%	36%	31%	52%	44%	37%	32%	56%	45%	39%	34%
Contact Manager	67%	57%	49%	52%	70%	59%	50%	54%	72%	63%	52%	55%

4

CONCLUSION & FUTURE WORKS

To sum up briefly...

Automated Approach

Detects:
- screens
- inputs
- domains and types of inputs

Systematic Sampling

Generates convenient test values by using covering arrays
- way to build a model
- test all states

Prediction

Predicts guard-conditions
- Decision-tree classifier

Comparison

Higher code coverage than:
- Dynodroid
- Monkey
- Random-testing

Simulations

Showed the relationship between factors that affects the prediction and coverage
- strength (t)
- determinism, etc.

Large Dataset

Apply proposed approach to a large number of Android applications.

Feedback-Driven Crawling

Take into account not only the interactions within a state but also the interactions across the states.

iOS Environment

Enhance the proposed approach to test iOS-based applications

THANKS!

Any questions?

omerkorkmaz@alumni.sabanciuniv.edu
cyilmaz@sabanciuniv.edu

MOTIVATION QUESTIONS

Can we discover the model with systematic sampling?

Is it possible to discover the model automatically by providing **systematic sampling** for UI fields as a **black-box approach** so that we can get higher code coverage?

Can we offer appropriate test values for each input?

Is it possible to determine **discrete settings as test values** that match the given input for systematic sampling?

Can we predict the guard conditions on the model?

Is it possible to **predict the guard conditions** from the discovered model by providing **systematic sampling** and interacting with only UI, since we don't make static analysis and know exact conditions?

CAN WE MAKE THE ENTIRE APPROACH AUTOMATED?

PIPELINE

STEP 01 — General Overview of SYSMODIS

STEP 02 — Screen and Input Detection

STEP 03 — Domain Detection and Pre-recorded Equivalence Classes

STEP 04 — Covering Array Generation

STEP 05 — Guard-Condition Discovery