2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 979-8-3503-2996-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASE56229.2023.00071

ICTDroid: Parameter-Aware Combinatorial Testing for Components of Android Apps

Shixin Zhang[1,3], Shanna Li[3], Xi Deng[2,4], Jiwei Yan[1,4,†], Jun Yan[1,2,4]
[1] Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[3] Information Technology Center, Beijing Jiaotong University
[4] University of Chinese Academy of Sciences
Email: {zhangshixin, yanjiwei}@otcaix.iscas.ac.cn, lsn@bjtu.edu.cn, {dengxi, yanjun}@ios.ac.cn

***Abstract*—Components are the fundamental building blocks of Android applications. Different functional modules represented by components often rely on inter-component communication mechanisms to achieve cross-module data transfer and method invocation. It is necessary to conduct robustness testing on components to prevent component launching crashes and privacy leaks caused by unexpected input parameters. However, as the complexity of the input parameter structure and the diversity of possible inputs, developers may overlook specific inputs that result in exceptions. At the same time, the vast input space also brings challenges to efficient component testing. In this paper, we designed an automated test generation and execution tool for Android application components named *ICTDroid*, which combines static parameter extraction and adaptive-strength combinatorial testing generation to detect bugs with a compact test suite. Experiments have shown that the tool triggers 205 unique exceptions in 30 open-source applications with 1,919 test cases in 83 minutes, where the developers have confirmed six defects in three issues we reported.**

***Index Terms*—Android App, Inter-Component Communication, Combinatorial Testing, Automated Testing**

I. Introduction

Components are considered the primary focus in numerous analysis and testing efforts for Android applications. The inter-component communication (ICC) mechanisms [1] they rely on are essential for interaction within and beyond applications. Therefore, the robustness problem in processing ICC parameters requires particular attention. In practice, developers handle and use ICC parameters with complex structures through branches with different conditions to adapt to user scenarios, resulting in multiple execution paths. On the one hand, it increases the possibility of unvalidated data access operations that cause exceptions. On the other hand, it increases the difficulty of triggering errors in specific execution paths. In order to effectively discover the ICC-related hidden bugs, it is necessary to accurately resolve the structure and values of ICC parameters and construct parameter models with high precision to improve the path exploration capability. However, as the size of the parameter model increases, the combinatorial space formed by all parameters and their values will expand dramatically, making it challenging to find the key parameter combinations that can trigger defects in the execution path.

To address the problem, this paper proposes a parameter-aware automated testing approach on Android application components and develops a prototype tool, ***ICTDroid***. To thoroughly explore potential defects behind parameters with complex structures, this approach leverages advanced ICC static analysis techniques [2], [3] for accurate and comprehensive modeling of ICC parameter structures and values. In response to the immense combinatorial space posed by modeled parameters, we perform adaptive-strength combinatorial testing (CT) [4], which uses data-flow analysis to amplify the interaction between used ICC parameters. Furthermore, we also extract constraints based on the semantics of ICC parameters to reduce ineffective parameter value combinations and enhance testing efficiency. With 1,919 test cases generated for 144 exposed activities in 30 open-source Android apps, *ICTDroid* detected 205 unique exceptions in 18 apps through 83 minutes of dynamic testing, with 88 test cases resulting in application crashes. Meanwhile, developers from GitHub confirmed six defects in three issues we reported. More information and demonstration of ICTDroid is publicly available at [5].

II. Background

Intent is the fundamental of the ICC mechanism in the Android system, which describes operation requests, typically containing operation type, target component, and additional data. Due to the complex structure of Intent and its ability to carry structural data, test cases need to meet multiple conditions simultaneously to reach deeper-level program branches.

Intent parameters can be divided into properties (of the Intent object itself) and fields (of the key-value pairs in Bundle type). Consider the Intent processing code with multiple branches shown in Fig. 1, where the execution path depends on various properties of the input Intent object. If the `loginV2()` method in line 7 has a defect, then the Intent object needs to satisfy at least three conditions simultaneously to trigger it:

1) Property $action$ of `Intent` equals to `action.LOGIN_V2`.
2) Field $data$ with type `Bundle` exists in property $extra$ of `Intent`.
3) Value of field key and $token$ with type `String` of field $data$ satisfies the condition (if any) related to the defect.

When preparing ICC test cases for the component containing such code, it is impossible to detect the defect without considering the values and structures of the `Intent` object.

†Corresponding author.

*This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA0320000 and XDA0320100) and the National Natural Science Foundation of China (Grant No. 62102405).

2643-1572/23/$31.00 ©2023 IEEE
DOI 10.1109/ASE56229.2023.00071

```
1  Intent i = getIntent();
2  if ("action.LOGIN_V1".equals(i.getAction())) {
3    loginV1(i.getIntExtra("appId", -1), i.getStringExtra("token"));
4  } else if ("action.LOGIN_V2".equals(i.getAction())) {
5    Bundle data = i.getBundleExtra("data");
6    if (data != null) {
7      // The loginV2() method has a defect
8      loginV2(data.getString("key"), data.getString("token"));
9    } else {
10     reportError(i.getStringExtra("error"));
11   }
12 }
```

Fig. 1: Motivating Example

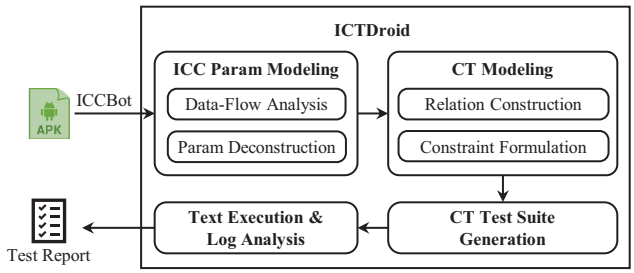


Fig. 2: Framework of *ICTDroid*

Therefore, it is necessary to obtain some guiding information about the parameter structure and its values by ICC resolution techniques [2], which refers to extracting ICC-related information by methods like static analysis, including but not limited to parameters and their structures, sources, targets, and so on.

Another issue is that while the `Intent` object has numerous parameters, not all of them affect the execution path of the program. For example, the only parameters in Fig. 1 that affect the program execution path include *action* and the *data* field in *extra*. Suppose we perform exhaustive testing directly on the Intent parameter structure obtained by ICC resolution. In that case, it requires enormous test cases, while many would be ineffective for defect exploration. In addressing this challenge, CT [4] tackles the problem of vast parameter space by concentrating on pairwise (or higher level) interaction between critical parameters, which significantly reduces the number of required test cases compared with exhaustive testing, resulting in cost and time efficiency while preserving a high defect detection rate.

CT requires the relation specification among several parameter groups, where the strength t reflects the degree of interaction coverage among the system parameters under test. The CT with constant strength among all parameters is also known as *t-way* CT. Oppositely, variable-strength refers to the scenario where the combinatorial strength between different parameters is non-constant. In applying CT theory, we should pay careful attention to the setting of the strength. High strength will lead to an exponential increase in the number of combinations required to be covered, resulting in a significant growth in the number of test cases needed. Conversely, low strength could mask latent faults caused by parameter interactions, which would only be triggered when the values of multiple parameters meet specific requirements.

III. ICTDroid: Parameter-Aware Combinatorial Testing for Components of Android Apps

This chapter gives an overview of the *ICTDroid* tool and explains complex-structural parameter modeling and combinatorial test case generation.

A. Overview of ICTDroid

Fig. 2 shows the overall workflow of our automated testing tool *ICTDroid*, which consists of four modules: 1) ICC Parameter Modeling, 2) CT modeling, 3) CT Test Suite Generation, and 4) Test Execution & Log Analysis.

For an application package (APK), we perform static analysis pre-processing through the state-of-the-art ICC resolution tool *ICCBot* [3] at the beginning to get the structure of Intent objects sent or received by each application component. The **ICC Parameter Modeling** module takes the result of ICC resolution as input, extracts data-flow-related information, and integrates it with the result to form the original parameter model. The **CT Modeling** module will deconstruct the properties with structures in the original parameter model, thus forming a regular and comprehensive CT model. After performing **CT Test Suite Generation**, the **Test Execution & Log Analysis** module will automatically install target apps, execute test cases and inspect system logs to export the final test report.

B. The Model for Complex-Structural Parameter

Some parameters of the Intent objects used by the ICC mechanism are not simple value but with some specific data structures. These structured parameters and their values can not be directly represented in test generation. To fill this gap, we need to transform the structures of these parameters to illustrate their value range as a finite discrete point set.

Our approach deconstructs a complex structured parameter into several value-controlling and structure-controlling parameters with their combinations. Value-controlling parameters represent the actual values of corresponding properties (or fields) in the original data structure, typically basic data types or specific objects. Structure-controlling parameters denote the structural states of corresponding complex-structural properties (or fields), which correspond to the following eight types of the structure-controlling value:

a) **Unset**: Do not set the property or field, cause methods like `hasExtra()` and `containsKey()` return `false`.
b) **Null**: Set the current property or field itself to `null`.
c) **MinVal/MaxVal**: Upper or lower bound of numeric types.
d) **Empty**: Empty object created by the parameterless constructor of non-primitive types.
e) **NotEmpty**: Non-null value or object preset for different types.
f) $\mathbf{Arr_{MinElem}}$/$\mathbf{Arr_{MaxElem}}$: Array or `ArrayList` containing single **MinVal** or **MaxVal**.
g) $\mathbf{Arr_{EmptyElem}}$/$\mathbf{Arr_{NotEmptyElem}}$: Array or `ArrayList` containing single **Empty** or **NotEmpty** object.
h) $\mathbf{Arr_{NullElem}}$: Array or `ArrayList` containing single **Null** value.

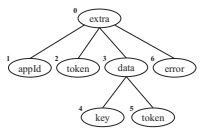
Consider the *category* property of Intent. It is a multi-value attribute of type `Set<String>`. To deconstruct it for fine control over the combination of each *category* value, we can deconstruct it into the structure-controlling parameter p_{cat_0} and value-controlling parameters p_{cat_1}, p_{cat_2}, ..., p_{cat_k}. Here, p_{cat_0} denotes the structural state of the *category* property, with possible values being **Empty** and **NotEmpty**. The value **Empty** indicates not set any value of the *category* property, while **NotEmpty** implies setting at least one. $p_{cat_i}(i \in [1,k])$ represents whether the *category* includes the ith value.

Assuming the values of *category* include "DEFAULT" and "NORMAL," the parameter model after applying the approach mentioned above can be represented as Fig. 3.

```
1: category: [Empty, NotEmpty]
2: category_DEFAULT: [true, false]
3: category_NORMAL: [true, false]
```

Fig. 3: Parameter Model of Category Data

Similarly, we can deconstruct and model the *extra* property in the code shown in Fig. 1. The *extra* property is essentially based on the Bundle data structure provided by the Android system, which is a container composed of key-value pairs. The *extra* property of the Intent in the code shown in Fig. 1 can be correspondingly represented as the tree structure shown in Fig. 4a, and the result shown in Fig. 4b can be obtained by further preserving the tree node position information based on the deconstruction and modeling approach mentioned above.



(a) Tree of Extra

```
1: extra: [Null, Empty, NotEmpty]
2: extra_0_1_int_appId: [Unset, -1]
3: extra_0_2_String_token: [Unset, Null, Empty, NotEmpty]
4: extra_0_3_Bundle_data: [Unset, Null, Empty, NotEmpty]
5: extra_0_6_String_error: [Unset, Null, Empty, NotEmpty]
6: extra_3_4_String_token: [Unset, Null, Empty, NotEmpty]
7: extra_3_5_String_key: [Unset, Null, Empty, NotEmpty]
```

(b) Parameter Model of Extra

Fig. 4: Tree and Parameter Model of Extra in Fig. 1

C. The Model for Combinatorial Testing

After modeling the parameters of an Intent object, we can obtain numerous parameters corresponding to original properties and fields. These parameters and their values can represent the complex-structural parameters to a certain degree, but all their combinations would form a vast space of test cases. To reduce test costs while maintaining error detection capability, we introduce the CT theory to alleviate the issue of exploding combinations of parameters and their values.

Relation construction. To properly set the combination strength among ICC parameters, we isolate the subsets of Intent parameters that appeared in each program path of the components. With a default combinatorial strength, we set inter-parameter relations with higher combinatorial strength for each parameter that have inter-parameter relations. Generally, for standard testing configurations, the default strength is set to one and the higher strength to three. Users can adjust these values through tool configuration.

Constraint formulation. According to the semantics and structures of the ICC parameters, some values are mutually exclusive, which form invalid combinations that can be reduced by formulating constraints. For example, for the structure-controlling parameter p_{cat_0} and the value-controlling parameters $p_{cat_1}, p_{cat_2}, \ldots, p_{cat_k}$ obtained after modeling the *category* property, if p_{cat_0} has a value of **Empty**, which means do not set any value for the category property, then the value of p_{cat_i} can only be *false*, i.e., $p_{cat_0} = Unset \rightarrow \forall p_{cat_i} = false(i \in [1,k])$; if p_{cat_0} has a value of **NotEmpty**, it implies that at least one p_{cat_i} should be *true*, i.e., $p_{cat_0} = NotEmpty \rightarrow \exists p_{cat_i} = true(i \in [1,k])$.

IV. Usage

For a user-given set of APKs, *ICTDroid* will first call *ICCBot* [3] to perform ICC static resolution, extract parameters based on the above process, and construct variable-strength CT model for ICC. Subsequently, *ICTDroid* invokes *ACTS* [6], a highly efficient tool to produce CT test suites, by using the auto-generated CT model like Fig. 5 as input to generate CT test suites that satisfy the strength and constraint requirements.

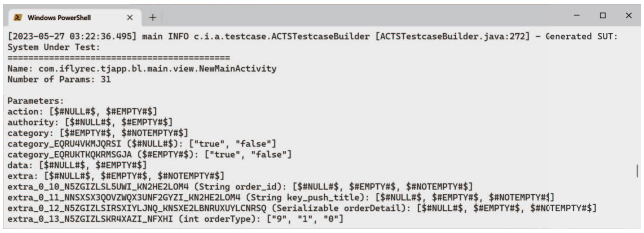


Fig. 5: Example of Auto-Generated CT Model

ICTDroid provide a Test Bridge application running on Android device for dynamic ICC testing, which converts the generated static representation of test cases into well-formed ICC message objects. After installing and launching the Test Bridge app, *ICTDroid* will automatically detect the device, install the target application, grant necessary permissions, and sequentially send each test case to the target application via the Test Bridge. Throughout this process, *ICTDroid* continuously monitors the application's runtime status using Android system services such as the Activity Manager and collects execution logs through Logcat.

After test execution, *ICTDroid* will analyze the log files generated during runtime and output the test report, which provides detailed stack traces for any exceptions in the application, along with the corresponding Intent data for one or more test cases.

V. Evaluation

To validate the effectiveness of our approach, we reuse a dataset of 30 real-world apps on F-droid according to recent work for evaluation of ICC resolution techniques [2]. In the adaptive combination strength strategy, we set the default strength to one and the increased strength to three. The evaluation metrics primarily include:

Execution efficiency. It refers to the ratio of the number of unique exceptions detected by the tests to the execution time in minutes. A higher value indicates a better exception detection capability within a unit of execution time.

TABLE I: Evaluation Result of Dynamic Testing on 30 Apps with Different Combi. Strength

Strategy	UniqueExc Count*	Case Count	Test Case Efficiency	Execution Time	Execution Efficiency
Adaptive	205	1,919	10.68	4,924s	2.50
1-way	153 (-25.4%)	552 (-71.2%)	27.72 (+159.5%)	1,631s (-66.9%)	5.63 (+125.3%)
2-way	217 (+5.9%)	2,080 (+8.4%)	10.43 (-2.3%)	5,735s (+16.5%)	2.27 (-9.1%)
3-way	246 (+20.0%)	6,699 (+249.1%)	3.67 (-65.6%)	18,112s (+267.8%)	0.81 (-67.4%)
4-way	—	19,488 (+915.5%)	—	—	—

Case efficiency. It refers to the ratio of the number of unique exceptions detected by the tests to the number of test cases in hundreds. A higher value indicates a better exception detection capability within a unit quantity of generated test cases.

Table I compares the evaluation results of automatic dynamic testing under different combination strengths. Since the 4-way CT resulted in excessive test cases and high testing costs, only the number of test cases is listed here for illustration purposes. According to the result, we can draw the following conclusions:

- Under the 1-way CT, it only takes about 30% of the test cases under the adaptive strength strategy to detect approximately 75% of unique exceptions. Moreover, it exhibits relatively high execution and case efficiency, demonstrating the rationality and effectiveness of our modeling strategies.
- The adaptive strength strategy contributes another 25% of unique exceptions compared to 1-way CT. Although there is a slight decrease in efficiency, there is still a marginal improvement compared to 2-way CT. The overall number of test cases and execution time remain acceptable.
- When the strength reaches three or even higher, the number of test cases and execution time rapidly increases. Despite an increase in detected unique exceptions, there is a significant decrease in execution and case efficiency, making the overall testing cost unacceptable.

TABLE II: Comparison on Detected Unique Crashes

Tool	Apps Can be Tested	Avg. Analysis / TestExec Time (minutes)	Unique Crashes
ICTDroid	30 / 30	16.4 / 2.7	88
Fax	26 / 30	3.8 / 60.0	58
IntentFuzzer	30 / 30	- / -	24

Table II compares *ICTDroid* with two available related tools *Fax* [7] and *IntentFuzzer* [8] in terms of the number of unique crashes detected. Note that the time of *IntentFuzzer* is ignored because it only performs two test cases per activity. *ICTDroid* found the highest number of unique crashes on all 30 apps within an acceptable analysis and test execution time.

Meanwhile, we reported six defects on three apps detected only by *ICTDroid* with three issues (check README at [5] for details). Developers from GitHub confirmed all of them.

*For the strategies of the combination strength, we compare the *Unique Exception* instead of *Unique Crash*, since one *Unique Crash* may correspond to multiple *Unique Exceptions*.

VI. Related Work

Regarding Android application testing, the literature review by Kong et al. [1] indicates that there have been numerous GUI-based approaches and tools in recent years, like Monkey [9] and Appium [10]. However, the publicly accessible tools for automated testing based on ICC are limited. IntentFuzzer [8] can test all components of an application using empty Intent or Intent with all null values but without Intent with valid values and structured data. Hwacha [11] allows fuzzing tests through ADB using only manually-specified Intent specifications, while the command line cannot handle extra data with complex structures. Fax [7] introduces simple static analysis strategies to extract ICC parameter structures to aid multi-entry GUI testing. However, its analysis precision is restricted, and it needs to consider the interactions between parameters sufficiently. The previous version of ICTDroid, known as AACT, was utilized by Deng et al. [12], serving as a foundation upon which enhancements were made to both test generation methodologies and test execution capabilities.

VII. Conclusion

This paper presents *ICTDroid*, a precise and efficient automated testing tool for Android application components. *ICTDroid* overcomes the challenges in automatic parameter analysis and complex-structural data representation using state-of-the-art static analysis techniques for ICC. It employs an adaptive-strength CT method to reduce the test set, striking a balance between testing cost and error detection capability. Furthermore, *ICTDroid* constitutes a viable solution for cost-sensitive scenarios, such as feature testing after porting of Android apps onto emerging architectures like RISC-V.

References

[1] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated Testing of Android Apps: A Systematic Literature Review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2019.

[2] J. Yan, S. Zhang, Y. Liu, X. Deng, J. Yan, and J. Zhang, "A Comprehensive Evaluation of Android ICC Resolution Techniques," in *ASE 2022*, 2022, Conference Proceedings, p. Article 1.

[3] J. Yan, S. Zhang, Y. Liu, J. Yan, and J. Zhang, "ICCBot: Fragment-Aware and Context-Sensitive ICC Resolution for Android Applications," in *ICSE-Companion 2022*, 2022, Conference Proceedings, pp. 105–109.

[4] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "Comparative Analysis of Constraint Handling Techniques for Constrained Combinatorial Testing," *TSE 2021*, vol. 47, no. 11, pp. 2549–2562, 2021.

[5] LightningRS, "ICTDroid," 2023. [Online]. Available: https://github.com/LightningRS/ICTDroid

[6] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "ACTS: A Combinatorial Test Generation Tool," *ICST 2013*, pp. 370–375, 2013.

[7] J. Yan, H. Liu, L. Pan, J. Yan, J. Zhang, and B. Liang, "Multiple-Entry Testing of Android Applications by Constructing Activity Launching Contexts," in *ICSE 2020*, 2020, Conference Proceedings, p. 457–468.

[8] MindMac, "IntentFuzzer," 2015. [Online]. Available: https://github.com/MindMac/IntentFuzzer

[9] Google, "UI/Application Exerciser Monkey," 2018. [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey

[10] M. Hans, *Appium essentials*. Packt Publishing Ltd, 2015.

[11] K. Choi, M. Ko, and B.-m. Chang, "A Practical Intent Fuzzing Tool for Robustness of Inter-Component Communication in Android Apps," *KSII TIIS*, vol. 12, pp. 4248–4270, 2018.

[12] X. Deng, J. Yan, S. Zhang, J. Yan, and J. Zhang, "Variable-strength Combinatorial Testing of Exported Activities Based on Misexposure Prediction," *Journal of Systems and Software*, vol. 204, p. 111773, 2023.