Computer-Assisted Specification of Asynchronous Interfaces with Non-deterministic Behavior

Nicholas V. Lewchenko
University of Colorado Boulder
Colorado, USA

Abstract

The Android Framework is designed around components with *asynchronous interfaces*, in which inputs and outputs are not directly coupled. Precisely specifying behavior of this sort is a slow, error-prone process, and thus documentation and testing for such components is usually incomplete. I have participated over the last year in a collaborative research project seeking to solve this problem by automating the generation and verification of these specifications via active learning on a live Android system. Part of my work has been the extension of our automation technique to Android Framework components with *non-deterministic* behavior that prevents direct application of active learning algorithms. To this end, I have applied our learning engine in the form of a *specification assistant* which mixes automation and manual guidance to learn non-deterministic interfaces with as little user intervention as possible.

CCS Concepts • **Software and its engineering** → **Software verification and validation**; *Interface definition languages*; Documentation; • **Theory of computation** → *Formal languages and automata theory*;

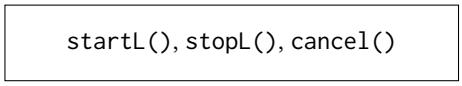
Keywords Active Learning, Typestate, Android

ACM Reference Format:
Nicholas V. Lewchenko. 2017. Computer-Assisted Specification of Asynchronous Interfaces with Non-deterministic Behavior. In *Proceedings of 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3135932.3135944

SPLASH Companion'17, October 22–27, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5514-8/17/10…$15.00
https://doi.org/10.1145/3135932.3135944

Inputs



Outputs

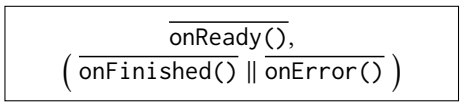


Figure 1. Learning Purpose alphabet for Speech Recognizer

1 Background

The interfaces of asynchronous components—in the form of Java classes—found in the Android Framework are divided into *callin* and *callback* methods. A user of the class gives it inputs by invoking some of its callin methods, which return immediately. The response to those callins comes some time later when the class invokes some of its callback methods, which notify the user of some completed task or provide results. Specifications for asynchronous classes can be encoded by a form of graph known as an *interface automaton* (as seen in Figure 3). *Active learning* algorithms such as L* can automatically learn correct, complete behaviors represented by interface automata by performing selected sequences of inputs to a system and recording their responses. [1]. The core of our project is an active learning engine which implements this technique for class objects in a live Android system.

While powerful, active learning cannot be directly applied to interfaces which are non-deterministic [1]. This unfortunately makes a large portion of class interfaces in the Android Framework impossible to infer in a totally automated manner, because many depend on environmental conditions in ways that make their behavior effectively non-deterministic.

2 Approach

To address this limitation, I have augmented our core active learning engine to perform as an interactive *specification assistant*. The tool, called DroidStar, is used by incrementally building up the set of callin and callback methods to be studied in a Java interface called the `LearningPurpose` (Figure 1). Inference proceeds as a series tests chosen by L*, in the form

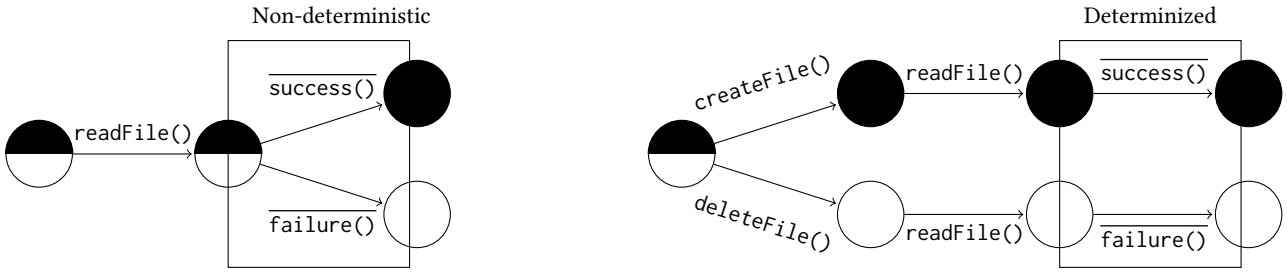


Figure 2. Environment lifting

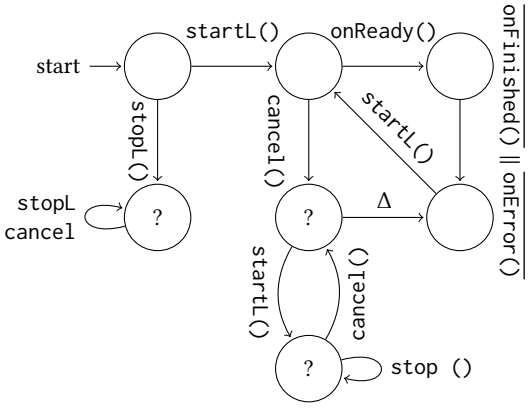


Figure 3. Learned asynchronous typestate for the Speech Recognizer class

of prefix-closed callin sequences during which callbacks are observed and recorded. If the tool observes non-deterministic behavior during this attempt (by caching and comparing the results of prefixes), it terminates immediately and presents the user with the offending query and disagreeing responses. With this information, the user manually "determinizes" the interface in one of two ways:

Output merging If two non-deterministically chosen callbacks take the system to a common state, the user can "contain" the non-determinism by merging the callbacks in the `LearningPurpose`, replacing them with a single output symbol that both callbacks report. If the merge was incorrect, the non-determinism will necessarily propagate to actions after the merged output and the user will be alerted immediately. Figures 1 and 3 provide an example of this method with the `onFinished` and `onError` outputs.

Environment lifting If the non-deterministic choice depends on an environmental condition that can be controlled by the active learning system, the relevant manipulation of that condition can be encoded as an additional input. For example, a choice of callbacks that depends on the existence of a file in the Android device's file-system can be determinized by creating inputs that create and delete that file. Figure 2 demonstrates this lifting; The environmental condition of a file existing or not existing is represented by black and white circles. Non-deterministic states (the half-filled circles) are determinized by explicit input actions so that sensitive inputs (such as `readFile()` in the figure) are performed on determined states only.

Both of these strategies are enabled by the flexibility of input and output symbols defined in the `LearningPurpose` interface.

The inference of partially non-deterministic interfaces via guided user interaction is a unique approach. Prior work that has been able to address non-determinism has been based in static analysis of synchronous components through symbolic evaluation or predicate abstraction, in which all behavioral branches can be enumerated and explored [2, 3]. The learning approach of the project for which DroidStar was produced [4] is based on dynamic testing in order to reasonably target the size of Android Framework code and complexity of asynchronous interfaces, to which symbolic methods are unlikely to scale.

3 Results

Of the 10 commonly used Android Framework classes we have used to evaluate the DroidStar tool [4], 3 required the non-determinism handling approach I've discussed here: `FileObserver`, `SpeechRecognizer`, and `SQLiteOpenHelper`. `FileObserver` and `SQLiteOpenHelper` both interact with the Android device's file-system and thus required lifting of environment actions to fully explore their behaviors.

In the `SpeechRecognizer` case, where the output merging strategy was used, we discovered a bug in the form of three spurious states where the class gets stuck, triggered by a series of inputs which manual testing previously required for non-deterministic classes would be unlikely to find. In Figure 3, these spurious states are marked with question marks. The one reached by `stopL()` from the starting state is inescapable, meaning that calling `stopL()` on a fresh `SpeechRecognizer` object effectively kills it. The Δ transition is a virtual input used to by DroidStar to poll for callbacks, which should not appear in sensible typestates; its presence there indicates a race condition.

References

[1] F. Aarts and F. Vaandrager. Learning I/O automata. In *CONCUR 2010*, pages 71–85, 2010.

[2] Dimitra Giannakopoulou, Zvonimir Rakamarić, and Vishwanath Raman. Symbolic learning of component interfaces. In *Proceedings of the 19th International Conference on Static Analysis*, SAS'12, pages 248–264, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Permissive interfaces. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 31–40, New York, NY, USA, 2005. ACM.

[4] Arjun Radhakrishna, Nicholas Lewchenko, Shawn Meier, Sergio Mover, Krishna Chaitanya Sripada, Damien Zufferey, Bor-Yuh Evan Chang, and Pavol Cerný. Learning asynchronous typestates for android classes. *CoRR*, abs/1701.07842, 2017.