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

A collaborative research project from the University of Colorado Boulder [2]

CUPLV

Callback Typestates for Android Components

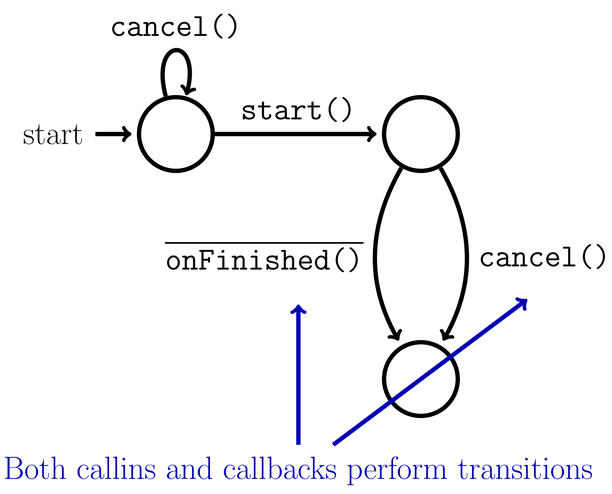
Example: The `SimpleTimer` Class

Public methods	
`final void`	`cancel()` Cancel the countdown.
`abstract void`	`onFinish()` Callback fired when the time is up.
`abstract void`	`onTick(long millisUntilFinished)` Callback fired on regular interval.
`final CountDownTimer`	`start()` Start the countdown.

Questions

- Are we allowed to call `start()` twice?
- Is it possible to recieve `onFinish()` after calling `cancel()`?
- Can we call `start()` again after a `cancel()` or `onFinish()`?

A *callback typestate* provides this missing information.



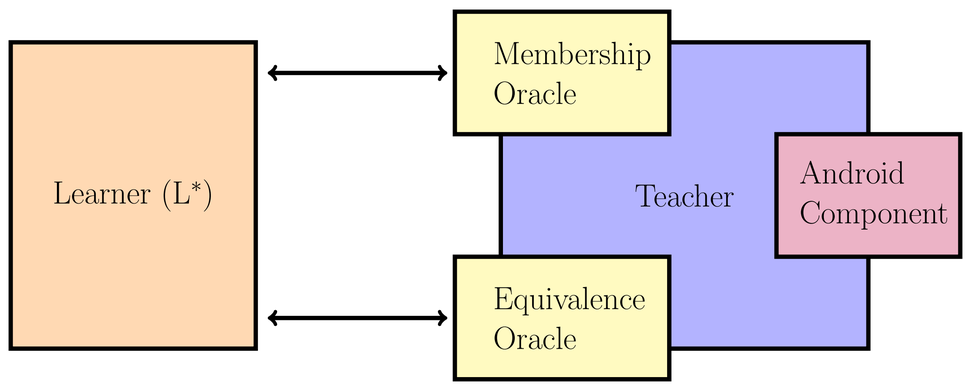
Answers

- We cannot call `start()` twice.
- We never receive $\overline{\texttt{onFinish()}}$ after calling `cancel()`.
- The `SimpleTimer` is a single-use object.

Callback typestates are useful for documentation and verification, but they are rarely provided in Android library documentation, likely because they are difficult to write with precision. **Our solution: generate them automatically!** [2]

Automated Typestate Learning

We use a variation of the L* algorithm [1] to automatically search behavior spaces and generate callback typestates. L* and other *active learning* algorithms perform by performing queries on a "teacher" that wraps the system under study.



Membership Queries

The L* learner builds a prefix-closed set of callin sequences and asks the teacher whether each one is allowed (and if so what callbacks it produces). In our case, the teacher answers by running the sequence as a test and recording any callbacks.

$$
\begin{aligned}
[\Delta, \texttt{cancel}(), \texttt{start}()] &\rightarrow [\delta, +, +] \\
[\Delta, \texttt{cancel}(), \texttt{start}(), \Delta] &\rightarrow [\delta, +, +, \texttt{onFinished}()] \\
[\Delta, \texttt{cancel}(), \texttt{start}(), \texttt{start}()] &\rightarrow \texttt{error}
\end{aligned}
$$

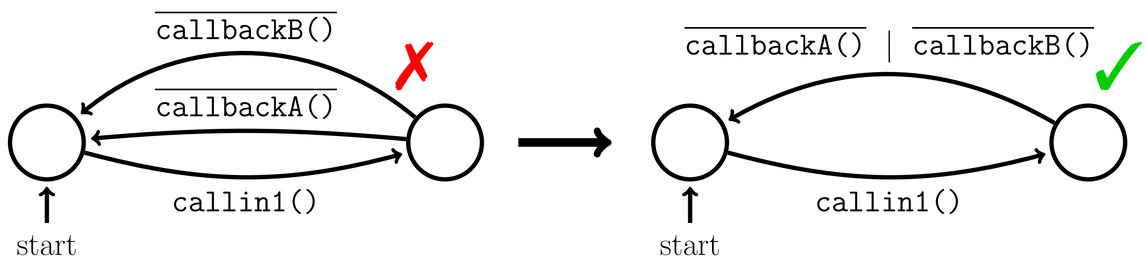
Equivalence Queries

When the table of callin sequences and results meets certain conditions, L* creates an automaton that contains all the sequences in the table and asks the teacher if it is complete. This is impossible for most systems to answer perfectly; we simulate the query with a set of membership queries that distinguishes states according to a selected bound.

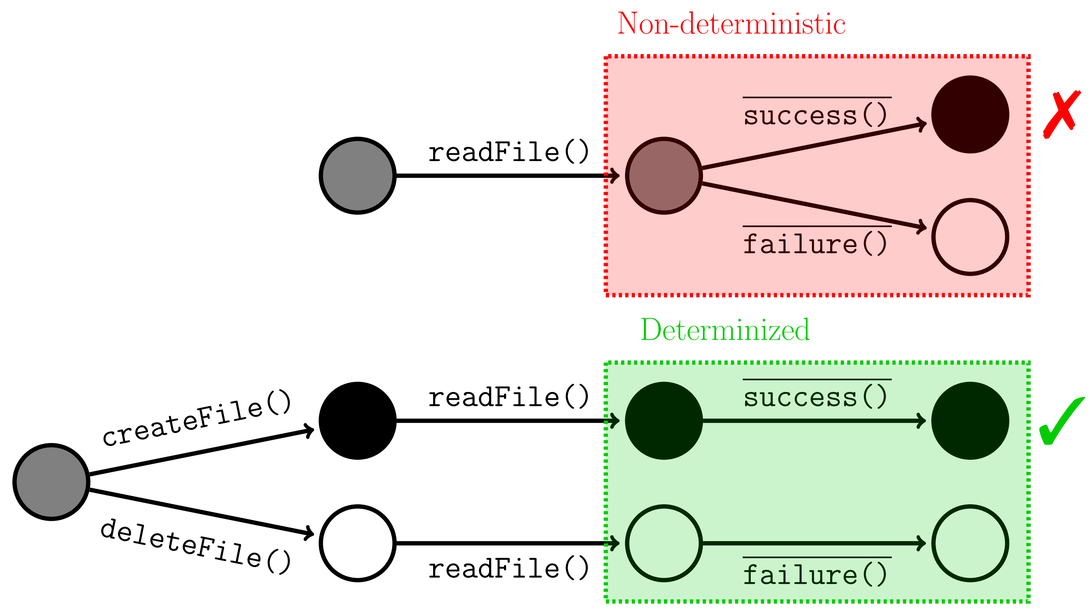
Mixing Automation and User Guidance for Non-deterministic Interfaces

The L* algorithm *cannot learn* interfaces which have non-deterministic callbacks, which are common for Android components that interact with device sensors or network services. We therefore package the learning engine as an interactive tool which prompts the user to bridge over unlearnable behavior when it is encountered. **We have discovered several strategies for isolating non-determinism in this way:**

Output Merging



Environment Lifting



Multi-form Inputs

If a user suspects that an environmental factor will change the system's behavior, now or after a future change, they can perform tentative environment lifting by encoding a single callback with multiple *forms* which will be tested independently.

$$
\begin{aligned}
\mathsf{readFile} = \; & | \; \texttt{createFile}(); \texttt{readFile}() \\
& | \; \texttt{deleteFile}(); \texttt{readFile}()
\end{aligned}
$$

If the behavior of the two forms matches, it will appear as one callin in the specification. If they ever diverge, the user will be prompted to fix the bug or fully split them.

Results

- Specifications generated for 10 commonly used Android Framework classes
- 3 classes required user guidance as described above: `FileObserver`, `SpeechRecognizer`, and `SQLiteOpenHelper`
- Found buggy corner-cases not reasonably discoverable by hand

References

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

[2] 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.