DroidStar: Callback Typestates for Android Classes

Arjun Radhakrishna[1] Nicholas V. Lewchenko[2]
Shawn Meier[2] Sergio Mover[2]
Krishna Chaitanya Sripada[2] Damien Zufferey[3]
Bor-Yuh Evan Chang[2] Pavol Černý[2]

[1]Microsoft (and University of Pennsylvania)

[2]University of Colorado Boulder

[3]Max Planck Institute for Software Systems

International Conference on Software Engineering, 2018

DroidStar: Callback Typestates for Android Classes

Arjun Radhakrishna[1] Nicholas V. Lewchenko[2]
Shawn Meier[2] Sergio Mover[2]
Krishna Chaitanya Sripada[2] Damien Zufferey[3]
Bor-Yuh Evan Chang[2] Pavol Černý[2]

[1]Microsoft (and University of Pennsylvania)

[2]University of Colorado Boulder

[3]Max Planck Institute for Software Systems

International Conference on Software Engineering, 2018

Everyone is using asynchronous frameworks!

In asynchronous frameworks, control of execution is divided.

Callins and Callbacks

In asynchronous frameworks, control of execution is divided.

Callins

Callbacks

Callins and Callbacks

In asynchronous frameworks, control of execution is divided.

Callins

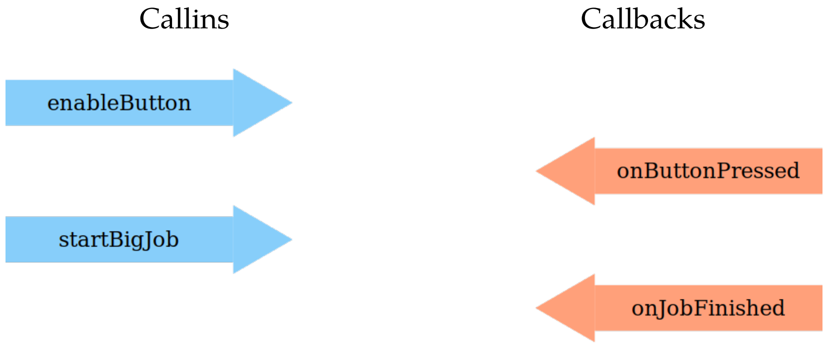
Callbacks

Callins and Callbacks

In asynchronous frameworks, control of execution is divided.

Callins and Callbacks

In asynchronous frameworks, control of execution is divided.



Callins and Callbacks

In asynchronous frameworks, control of execution is divided.

Callins | Callbacks

(Controlled by you) | (Controlled by framework)

The Android Framework

In the Android Framework, Java objects are the asynchronous interfaces.

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.

In the Android Framework, Java objects are the asynchronous interfaces.

Public methods		
final void	cancel() Cancel the count	Callin
abstract void	onFinish() Callback fired wh p.	Callback
abstract void	onTick(long mi hed) Callback fired on	Callback
final CountDownTimer	start() Start the countdo	Callin

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

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.

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

| start | cancel | → | ??? |

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.

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

start cancel → ???

cancel start → ???

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.

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

start	cancel	→	???
cancel	start	→	???
start	start	→	???

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.

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

start	cancel	→	???
cancel	start	→	???
start	start	→	???
start	onFinished	→	???

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.

Stateful Behavior

Android Framework objects are stateful: what we have seen and what we have done changes what can happen next.

But in what way?

start	cancel	→	???
cancel	start	→	???
start	start	→	???
start	onFinished	→	???

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.

Object type is not enough!

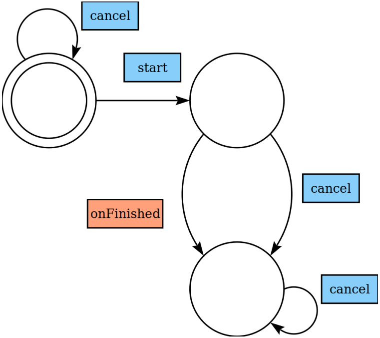
A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

- Paths in the diagram are possible traces

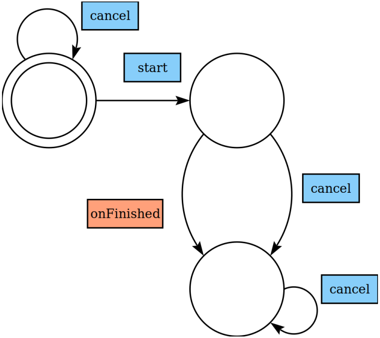


A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

- Paths in the diagram are possible traces

start	cancel	→	???
cancel	start	→	???
start	start	→	???
start	onFinished	→	???

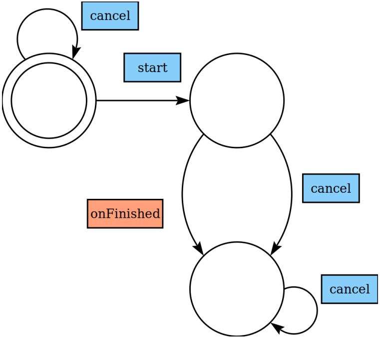


A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

- Paths in the diagram are possible traces

start	cancel	→	(Nothing)
cancel	start	→	onFinished
start	start	→	(Error)
start	onFinished	→	(Nothing)

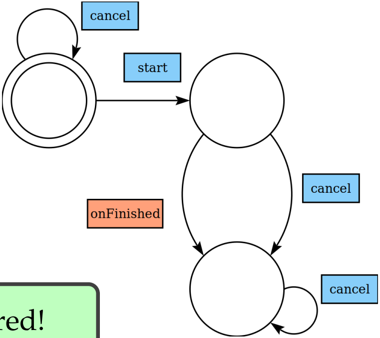


A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

- Paths in the diagram are possible traces

start	cancel	→	(Nothing)
cancel	start	→	onFinished
start	start	→	(Error)
start	onFinished	→	(Nothing)



Questions answered!

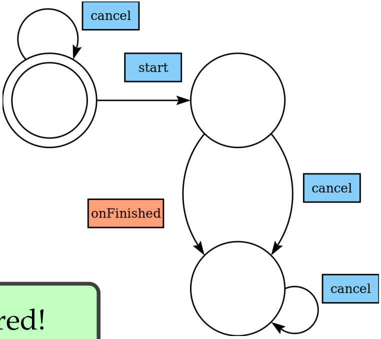
A More Complete Picture

- We need a specification that describes stateful behavior
- Should be formal (machine readable) like types

Callback Typestate

- Paths in the diagram are possible traces

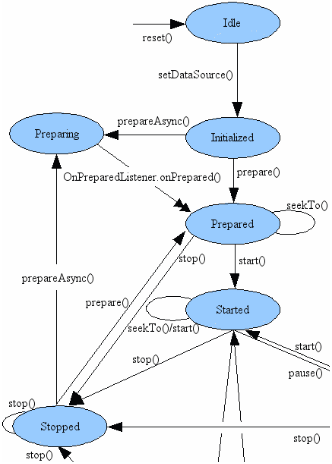
start	cancel	→	(Nothing)
cancel	start	→	onFinished
start	start	→	(Error)
start	onFinished	→	(Nothing)



Questions answered!

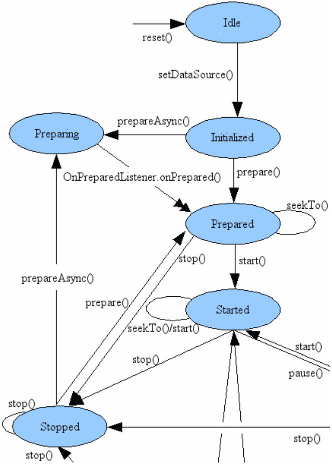
So Where Are They?

- Callback typestates are clearly useful.
- Informal examples exist as documentation…but not many!



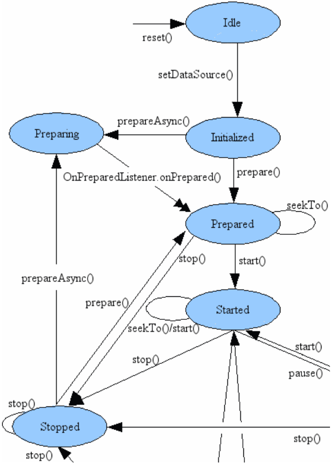
So Where Are They?

- Callback typestates are clearly useful.
- Informal examples exist as documentation…but not many!
- Writing specifications manually is tedious and error-prone.



So Where Are They?

- Callback typestates are clearly useful.
- Informal examples exist as documentation…but not many!
- Writing specifications manually is tedious and error-prone.
- Can these be produced automatically?



Previous Work in Automated Typestate Inference

Static/symbolic analysis

Testing-based

Previous Work in Automated Typestate Inference

Static/symbolic analysis

- Symbolic model of system
- Does not scale to a system like the Android Framework

Testing-based

Previous Work in Automated Typestate Inference

Static/symbolic analysis

- Symbolic model of system
- Does not scale to a system like the Android Framework

Testing-based

- Use active learning to test all of behavior space
- Standard methods require intractable numbers of tests

Previous Work in Automated Typestate Inference

Static/symbolic analysis

- Symbolic model of system
- Does not scale to a system like the Android Framework

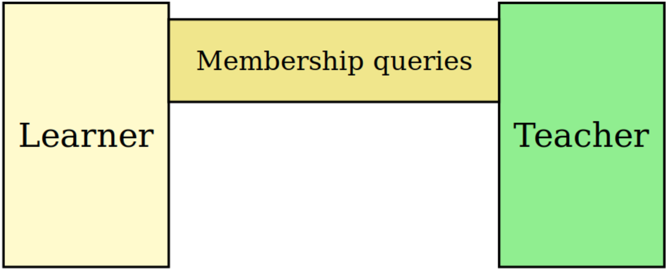
Testing-based

- Use active learning to test all of behavior space
- Standard methods require intractable numbers of tests

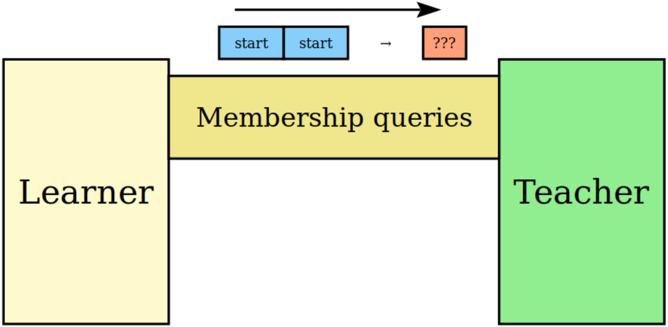
Both approaches have only covered "classical" input-only typestates.

Active Learning Overview

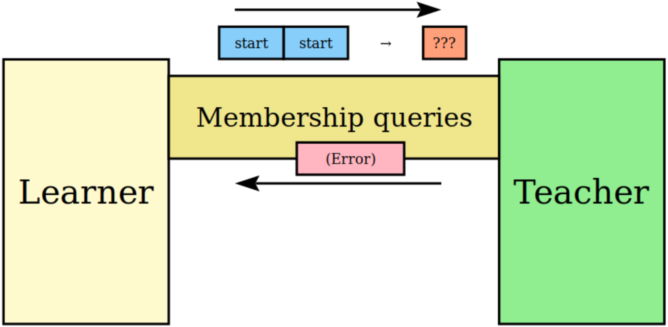
Active Learning Overview



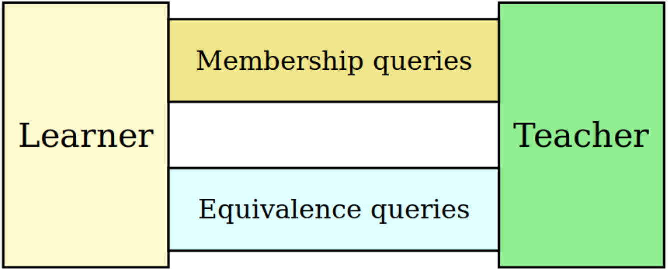
Active Learning Overview



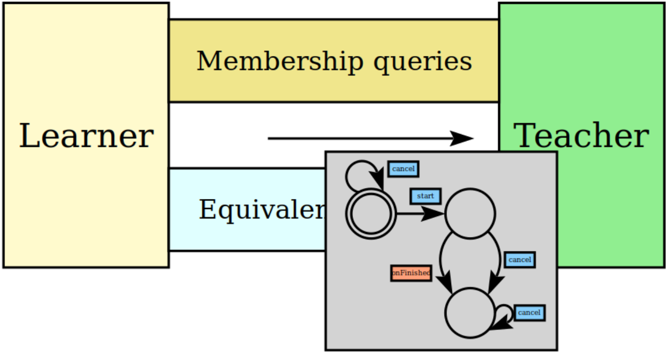
Active Learning Overview



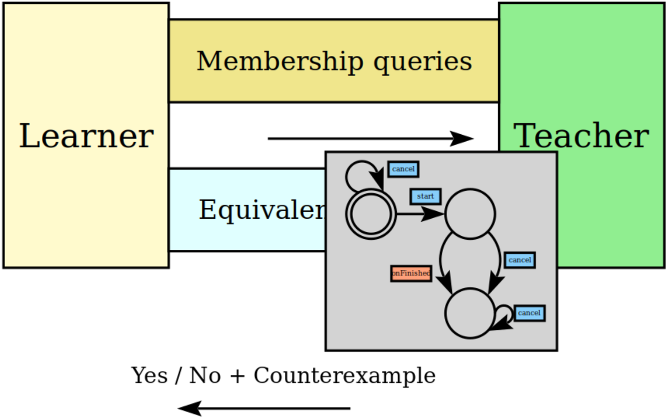
Active Learning Overview



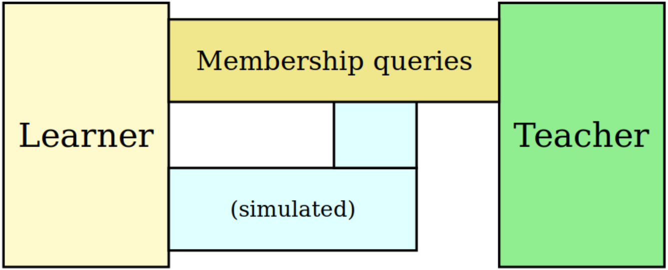
Active Learning Overview



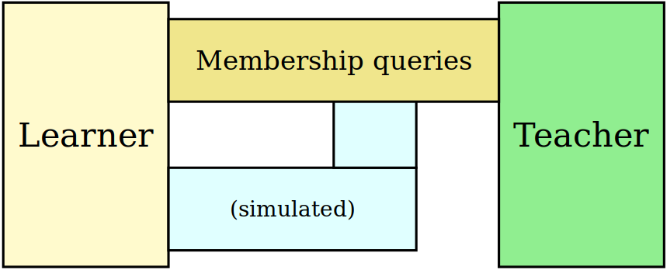
Active Learning Overview



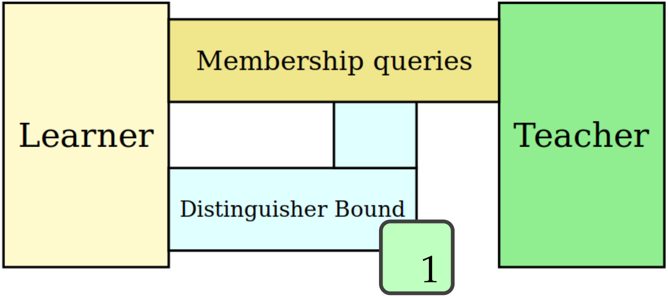
Active Learning Overview



Contributions

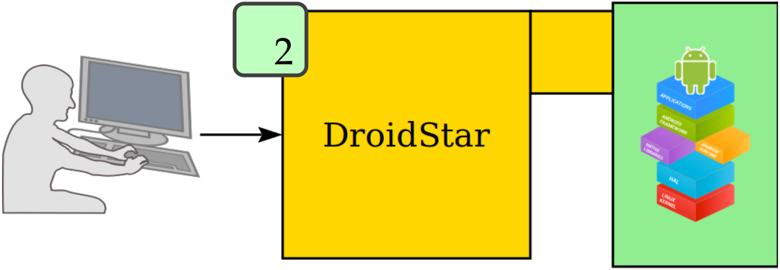


Contributions



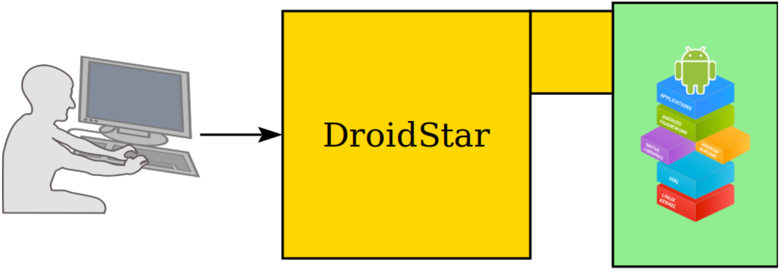
1. Distinguisher Bound: an efficient equivalence algorithm

Contributions



1. Distinguisher Bound: an efficient equivalence algorithm
2. DroidStar: a high-level implementation and interface

Contributions



1. Distinguisher Bound: an efficient equivalence algorithm
2. DroidStar: a high-level implementation and interface
3. Evaluation:
   - RQ 1. Is the Distinguisher Bound equivalence algorithm an improvement?
   - RQ 2. Is the complete DroidStar tool effective?
   - RQ 3. Does callback typestate inference reveal interesting, non-obvious object behavior?

Distinguisher Bound

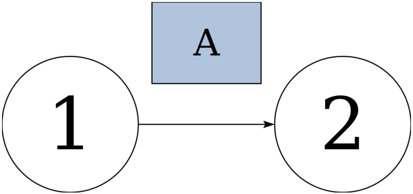
- Standard approach: State Bound (max number of states)
  - Equivalence algorithm: try all membership query sequences up to the number of states

Distinguisher Bound

- Standard approach: State Bound (max number of states)
  - Equivalence algorithm: try all membership query sequences up to the number of states
- Distinguisher Bound is the minimum steps out of any two states that produces a different output

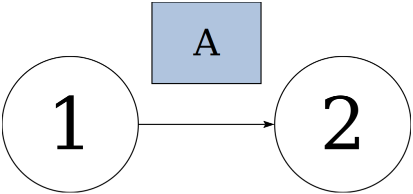
B_{DIST} EQUIVALENCE CHECK

Fidelity check for callin A and state 1:

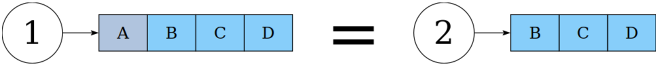


B_{Dist} Equivalence Check

Fidelity check for callin A and state 1:



Test all sequences BCD of inputs up to length B_{Dist}:



Are Distinguisher Bounds Small?

- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$

Are Distinguisher Bounds Small?

- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$
- In theory, $B_{\text{dist}} \leq B_{\text{state}} - 1$

Are Distinguisher Bounds Small?

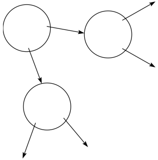
- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$
- In theory, $B_{\text{dist}} \leq B_{\text{state}} - 1$
- In practice, much better!
- 1 or 2, vs. up to 12 states

Class name	states	B_{dist}
AsyncTask	5	1
BluetoothAdapter	12	1
CountDownTimer	3	1
DownloadManager	4	1
FileObserver	6	1
ImageLoader (UIL)	5	1
MediaCodec	8	1
MediaPlayer	10	1
MediaRecorder	8	1
MediaScannerConnection	4	1
OkHttpCall (OkHttp)	6	2
RequestQueue (Volley)	4	1
SpeechRecognizer	7	1
SpellCheckerSession	6	1
SQLiteOpenHelper	8	2
VelocityTracker	2	1

Are Distinguisher Bounds Small?

- Standard state bound approach requires Σ^{B_state} membership queries
- Distinguisher bound requires Σ^{B_dist}
- In theory, $B_\text{dist} \leq B_\text{state} - 1$
- In practice, much better!
- 1 or 2, vs. up to 12 states

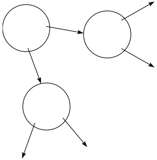
Class name	states	B_dist
AsyncTask	5	1
BluetoothAdapter	12	1
CountDownTimer	3	1
DownloadManager	4	1
FileObserver	6	1
ImageLoader (UIL)	5	1
MediaCodec	8	1
MediaPlayer	10	1
MediaRecorder	8	1
MediaScannerConnection	4	1
OkHttpCall (OkHttp)	6	2
RequestQueue (Volley)	4	1
SpeechRecognizer	7	1
SpellCheckerSession	6	1
SQLiteOpenHelper	8	2
VelocityTracker	2	1

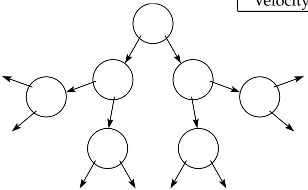


Are Distinguisher Bounds Small?

- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$
- In theory, $B_{\text{dist}} \leq B_{\text{state}} - 1$
- In practice, much better!
- 1 or 2, vs. up to 12 states

Class name	states	B_{dist}
AsyncTask	5	1
BluetoothAdapter	12	1
CountDownTimer	3	1
DownloadManager	4	1
FileObserver	6	1
ImageLoader (UIL)	5	1
MediaCodec	8	1
MediaPlayer	10	1
MediaRecorder	8	1
MediaScannerConnection	4	1
OkHttpCall (OkHttp)	6	2
RequestQueue (Volley)	4	1
SpeechRecognizer	7	1
SpellCheckerSession	6	1
SQLiteOpenHelper	8	2
VelocityTracker	2	1

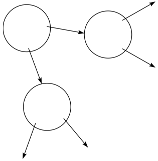


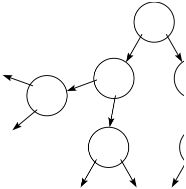


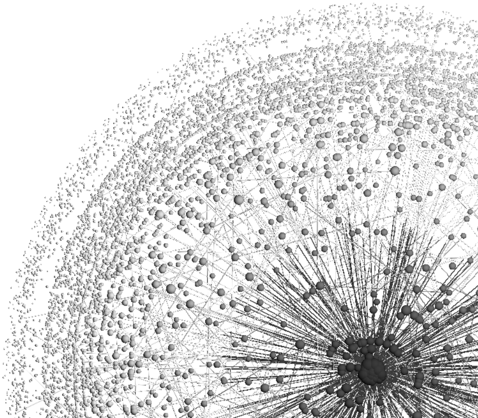
Are Distinguisher Bounds Small?

- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$
- In theory, $B_{\text{dist}} \leq B_{\text{state}} - 1$
- In practice, much better!
- 1 or 2, vs. up to 12 states

Class name	states	B_{dist}
AsyncTask	5	1
BluetoothAdapter	12	1
CountDownTimer	3	1
DownloadManager	4	1
FileObserver	6	1
ImageLoader (UIL)	5	1



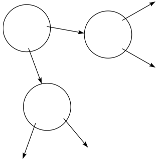


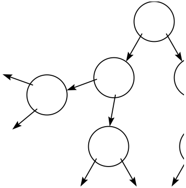


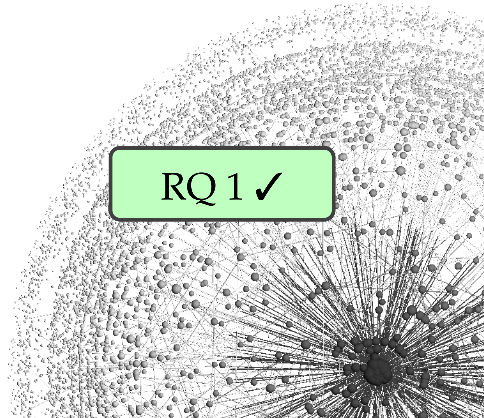
Are Distinguisher Bounds Small?

- Standard state bound approach requires $\Sigma^{B_{\text{state}}}$ membership queries
- Distinguisher bound requires $\Sigma^{B_{\text{dist}}}$
- In theory, $B_{\text{dist}} \leq B_{\text{state}} - 1$
- In practice, much better!
- 1 or 2, vs. up to 12 states

Class name	states	B_{dist}
AsyncTask	5	1
BluetoothAdapter	12	1
CountDownTimer	3	1
DownloadManager	4	1
FileObserver	6	1
ImageLoader (UIL)	5	1







DroidStar [1]

- Java/Scala library for creating and configuring instances of our active learning technique
- User writes a `LearningPurpose`
  1. Callin symbols with associated code snippets
  2. Subclass instrumented with callback reports
  3. Initializer to create fresh objects for tests
  4. Various option settings
- Compiles into an app that runs on an Android device

[1] https://github.com/cuplv/droidstar

DroidStar Example [2]

Defining a `LearningPurpose` for `AsyncTask`.

[2] Tutorial:
`https://github.com/cuplv/droidstar#writing-an-experiment`

Defining a `LearningPurpose` for `AsyncTask`.

1. Define callin symbols with code snippets

```
override def giveInput(i: String, altKey: Int): Unit = i match {
  case `execute` => task.execute(param)
  case `cancel` => task.cancel(false)
}
```

[2]Tutorial:
`https://github.com/cuplv/droidstar#writing-an-experiment`

DroidStar Example [2]

Defining a `LearningPurpose` for `AsyncTask`.

2. Instrument subclass with callback reports

```
class SimpleTask(localCounter: Int) extends AsyncTask[AnyRef,AnyRef,AnyRef] {

  override def onCancelled(s: AnyRef): Unit = {
    if (localCounter == counter) {
      respond(cancelled)
    }
  }

  override def onPostExecute(s: AnyRef): Unit = {
    respond(postexec)
  }
```

[2] Tutorial:
`https://github.com/cuplv/droidstar#writing-an-experiment`

DroidStar Example [2]

Defining a `LearningPurpose` for `AsyncTask`.

3. Define initializer for test isolation

```
override def resetActions(c: Context, b: Callback): String = {
  if (task != null) {
    task.cancel(true)
    counter += 1
  }
  task = new SimpleTask(counter)
  null
}
```

[2] Tutorial:
`https://github.com/cuplv/droidstar#writing-an-experiment`

DroidStar Evaluation

- Learned useful callback typestates for 16 commonly used Android Framework classes
- Process: manually identify significant callins and callbacks from online documentation, write and adjust `LearningPurpose` accordingly

Class name	LP LoC	Time (s)	Mem. Queries
AsyncTask	79	49	372 (94)
BluetoothAdapter	161	1273	839 (157)
CountDownTimer	94	134	232 (61)
DownloadManager	84	136	192 (43)
FileObserver	134	104	743 (189)
ImageLoader (UIL)	80	88	663 (113)
MediaCodec	152	371	1354 (871)
MediaPlayer	171	4262	13553 (2372)
MediaRecorder	131	248	1512 (721)
MediaScannerConnection	72	200	403 (161)
OkHttpCall (OkHttp)	79	463	839 (166)
RequestQueue (Volley)	79	420	475 (117)
SpeechRecognizer	168	3460	1968 (293)
SpellCheckerSession	109	133	798 (213)
SQLiteOpenHelper	140	43	1364 (228)
VelocityTracker	63	98	1204 (403)

DroidStar Evaluation

- Learned useful callback typestates for 16 commonly used Android Framework classes
- Process: manually identify significant callins and callbacks from online documentation, write and adjust `LearningPurpose` accordingly

Class name	LP LoC	Time (s)	Mem. Queries
AsyncTask	79	49	372 (94)
BluetoothAdapter	161	1273	839 (157)
CountDownTimer	94	134	232 (61)
DownloadManager	84	136	192 (43)
FileObserver	134	104	743 (189)
ImageLoader (UIL)	80	88	663 (113)
MediaCodec	152	371	1354 (871)
MediaPlayer	171	4262	13553 (2372)
MediaRecorder	131	248	1512 (721)
MediaScannerConnection	72	200	403 (161)
OkHttpCall (OkHttp)	79	463	839 (166)
RequestQueue (Volley)	79	420	475 (117)
SpeechRecognizer	168	3460	1968 (293)
SpellCheckerSession	109	133	798 (213)
SQLiteOpenHelper	140	43	1364 (228)
VelocityTracker	63	98	1204 (403)

RQ 2 ✓

DroidStar Evaluation

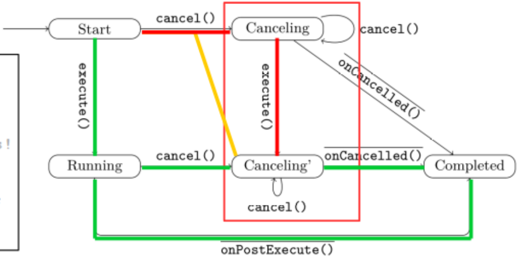
- Learned useful callback typestates for 16 commonly used Android Framework classes
- Process: manually identify significant callins and callbacks from online documentation, write and adjust `LearningPurpose` accordingly

- Discovered 7 cases of behavior that deviates from online documentation

Class name	LP LoC	Time (s)	Mem. Queries
AsyncTask	79	49	372 (94)
BluetoothAdapter	161	1273	839 (157)
CountDownTimer	94	134	232 (61)
DownloadManager	84	136	192 (43)
FileObserver	134	104	743 (189)
ImageLoader (UIL)	80	88	663 (113)
MediaCodec	152	371	1354 (871)
MediaPlayer	171	4262	13553 (2372)
MediaRecorder	131	248	1512 (721)
MediaScannerConnection	72	200	403 (161)
OkHttpCall (OkHttp)	79	463	839 (166)
RequestQueue (Volley)	79	420	475 (117)
SpeechRecognizer	168	3460	1968 (293)
SpellCheckerSession	109	133	798 (213)
SQLiteOpenHelper	140	43	1364 (228)
VelocityTracker	63	98	1204 (403)

- 92 queries, 49 seconds
- Unexpected: it is possible to call an `execute()` that never produces results!

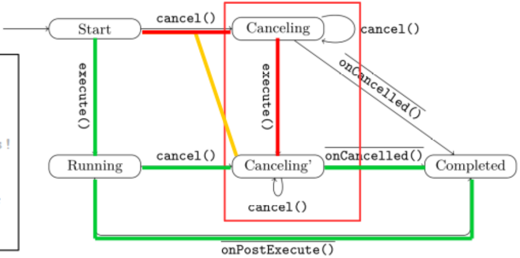
```
void onClick(Button b) {
    try {
        // Very important!
        // User must take
        // action if it fails!
        task.execute();
    } catch e {
        alertUserToFailure();
    }
}
```



Interesting Example: AsyncTask

- 92 queries, 49 seconds
- Unexpected: it is possible to call an `execute()` that never produces results!

```
void onClick(Button b) {
    try {
        // Very important!
        // User must take
        // action if it fails!
        task.execute();
    } catch e {
        alertUserToFailure();
    }
}
```



RQ 3 ✓

Future Work: Tooling

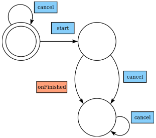
How can DroidStar fit into a software engineering workflow?

- Part of the API

Future Work: Tooling

How can DroidStar fit into a software engineering workflow?

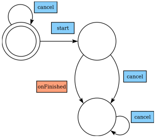
- Part of the API
  - Documentation



Future Work: Tooling

How can DroidStar fit into a software engineering workflow?

- Part of the API
  - Documentation



  - Intelligent IDE assistance

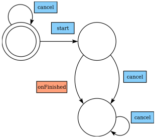
```
// Make sure the timer is started
timer.start();
timer.start();
```

Protocol violation
> start() cannot be called twice

Future Work: Tooling

How can DroidStar fit into a software engineering workflow?

- Part of the API
  - Documentation



  - Intelligent IDE assistance

```
// Make sure the timer is started
timer.start();
timer.start();
```

Protocol violation
> start() cannot be called twice

- Part of the test suite

```
Test Failure.
Latest build c56hk22 deviates from spec
on sequence:
    register → start → unregister

Accept this change as intended? (y/N)
```

Summary

1. Practical equivalence check based on distinguisher bound

Summary

1. Practical equivalence check based on distinguisher bound
2. DroidStar, an implementation of our modified active learning technique for Android, with a high-level interface for developer use

Summary

1. Practical equivalence check based on distinguisher bound
2. DroidStar, an implementation of our modified active learning technique for Android, with a high-level interface for developer use
3. Success in learning useful callback typestates for 16 commonly used Android classes, with some surprises

Summary

1. Practical equivalence check based on distinguisher bound
2. DroidStar, an implementation of our modified active learning technique for Android, with a high-level interface for developer use
3. Success in learning useful callback typestates for 16 commonly used Android classes, with some surprises

Together, a **solution** to the practical automated typestate learning problem.

End

Questions?

Try out our tool: `https://github.com/cuplv/droidstar`