Using fuzzing/property based testing to find bugs in a React/Redux application

Jarno Rantanen
@jareware
1.6.2018

What’s fuzzing?

What’s fuzzing?

“Fuzzing is feeding a piece of code (function, program, etc.) data from a large corpus, possibly dynamically generated, possibly dependent on the results of execution on previous data, in order to see whether it fails.”

https://hypothesis.works/articles/what-is-property-based-testing/

Feeding random inputs to an application in the hopes of making it crash

“Monkey Testing”

What's fuzzing?

https://twitter.com/sempf/status/514473420277694465

Smart Monkey
vs.
Dumb Monkey

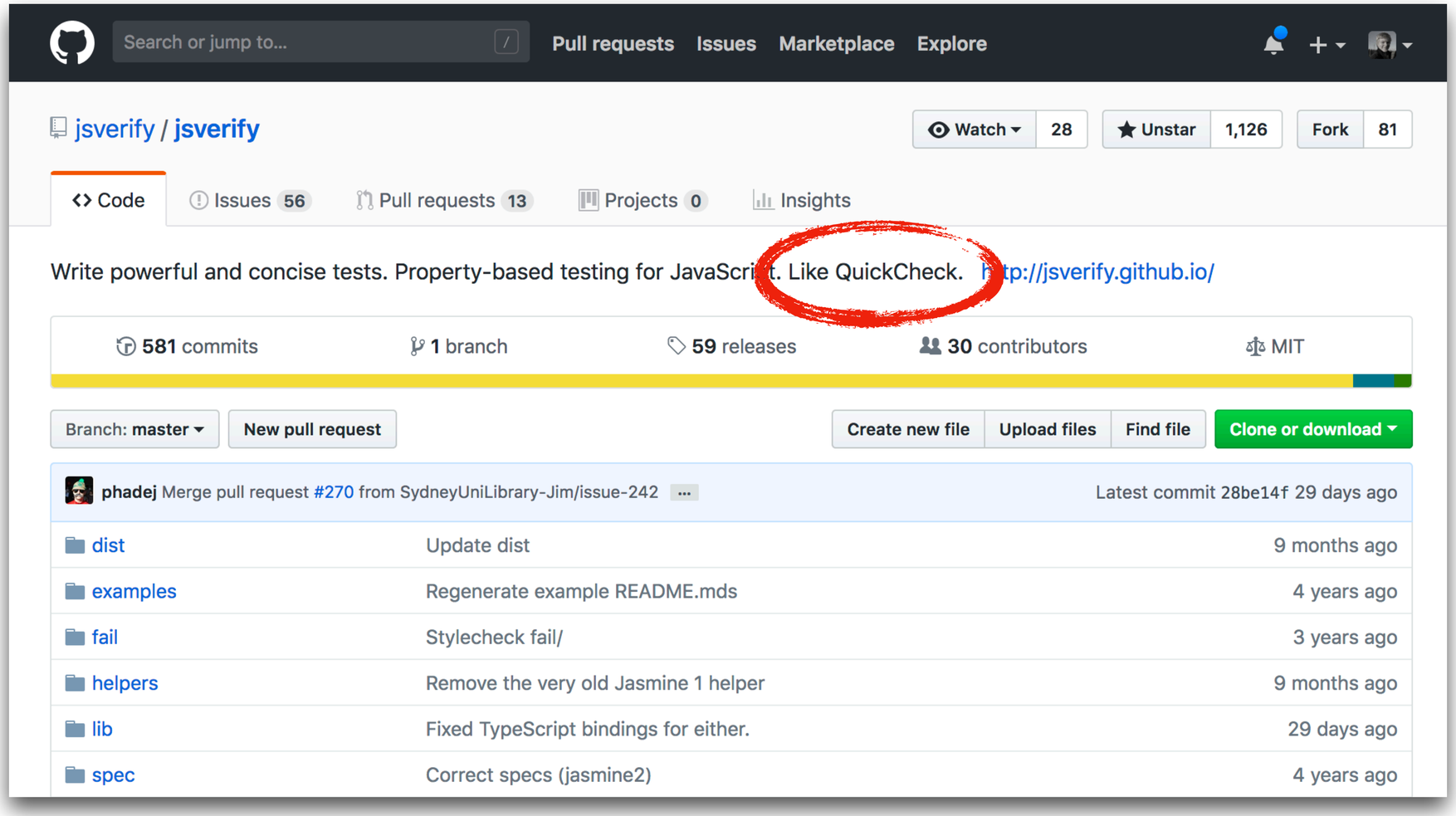
What’s property based testing?

What’s property based testing?

“Property-based tests make statements about the output of your code based on the input, and these statements are verified for many different possible inputs.”

http://blog.jessitron.com/2013/04/property-based-testing-what-is-it.html

What's property based testing?



https://github.com/jsverify/jsverify

Contrast: example based testing

Instead: run the same test 100’s of times, but with different, generated inputs

Test that some

properties

of the system always hold

For function

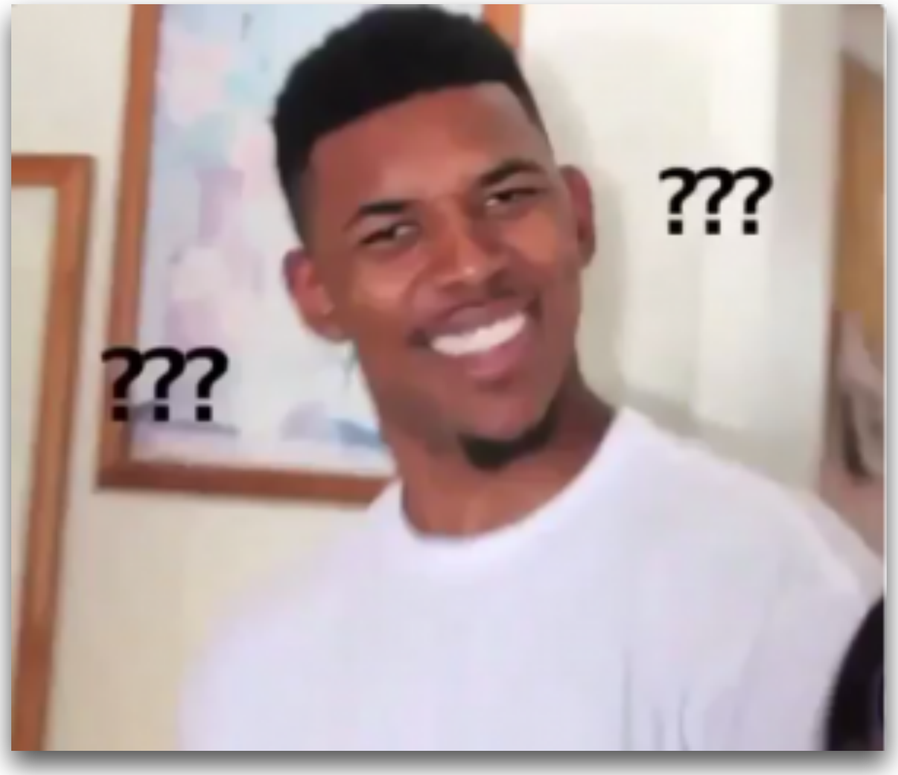
`add(x,y)`

the property

`x+y == add(x,y)`

must always hold

What’s property based testing?



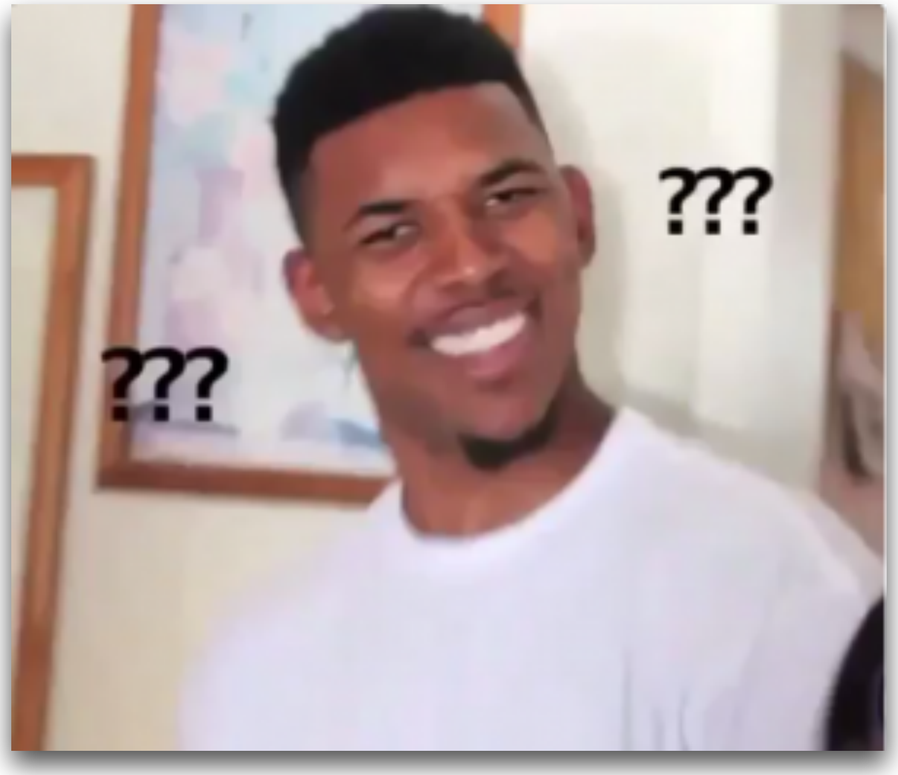
http://knowyourmeme.com/photos/993875-confused-nick-young

What's property based testing?

```
var jsc = require("jsverify");

// forall (f : bool -> bool) (b : bool), f (f (f b)) = f(b).
var boolFnAppliedThrice =
  jsc.forall("bool -> bool", "bool", function (f, b) {
    return f(f(f(b))) === f(b);
  });

jsc.assert(boolFnAppliedThrice);
// OK, passed 100 tests
```

https://github.com/jsverify/jsverify#synopsis

What’s property based testing?



http://knowyourmeme.com/photos/993875-confused-nick-young

Very practical examples soon!

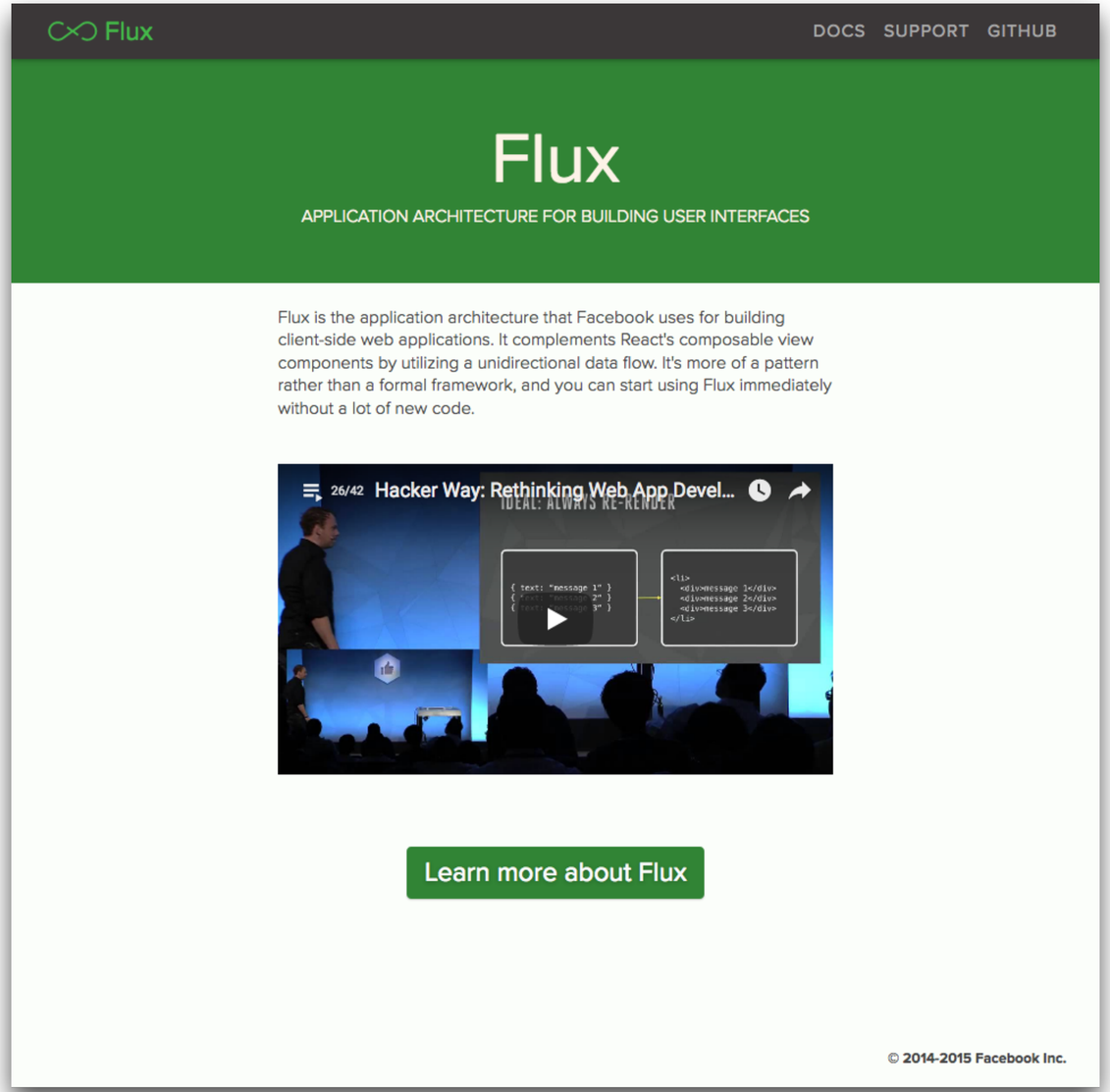
“Using fuzzing/property based testing”

Is this fuzzing?
Is this property based testing?
Both? Neither?

¯_(ツ)_/¯

Quick Flux refresher

Quick Flux refresher



https://facebook.github.io/flux/

Deterministic state machine

```
app(state,action)
  => newState
```

```
state1 + actionA => state2

state2 + actionB => state3

state3 + actionC => state4

...
```

Sounds simple, but easy to break

Project background

Loading event schedule...

Sources of complexity

Complex business rules

External data model

Lots, lots of data

Cross references

Mutual exclusions

Soft deletes

QA checklist

Peer review for every PR

Data normalisation

TypeScript

Test coverage

Unit & UI tests

Lots of assertions

Crash reports

Project background

“Combinatorial explosion is a fundamental problem in computing. It is the problem that the number of combinations that one has to examine grows exponentially, so fast that even the fastest computers will require an intolerable amount of time to examine them.”

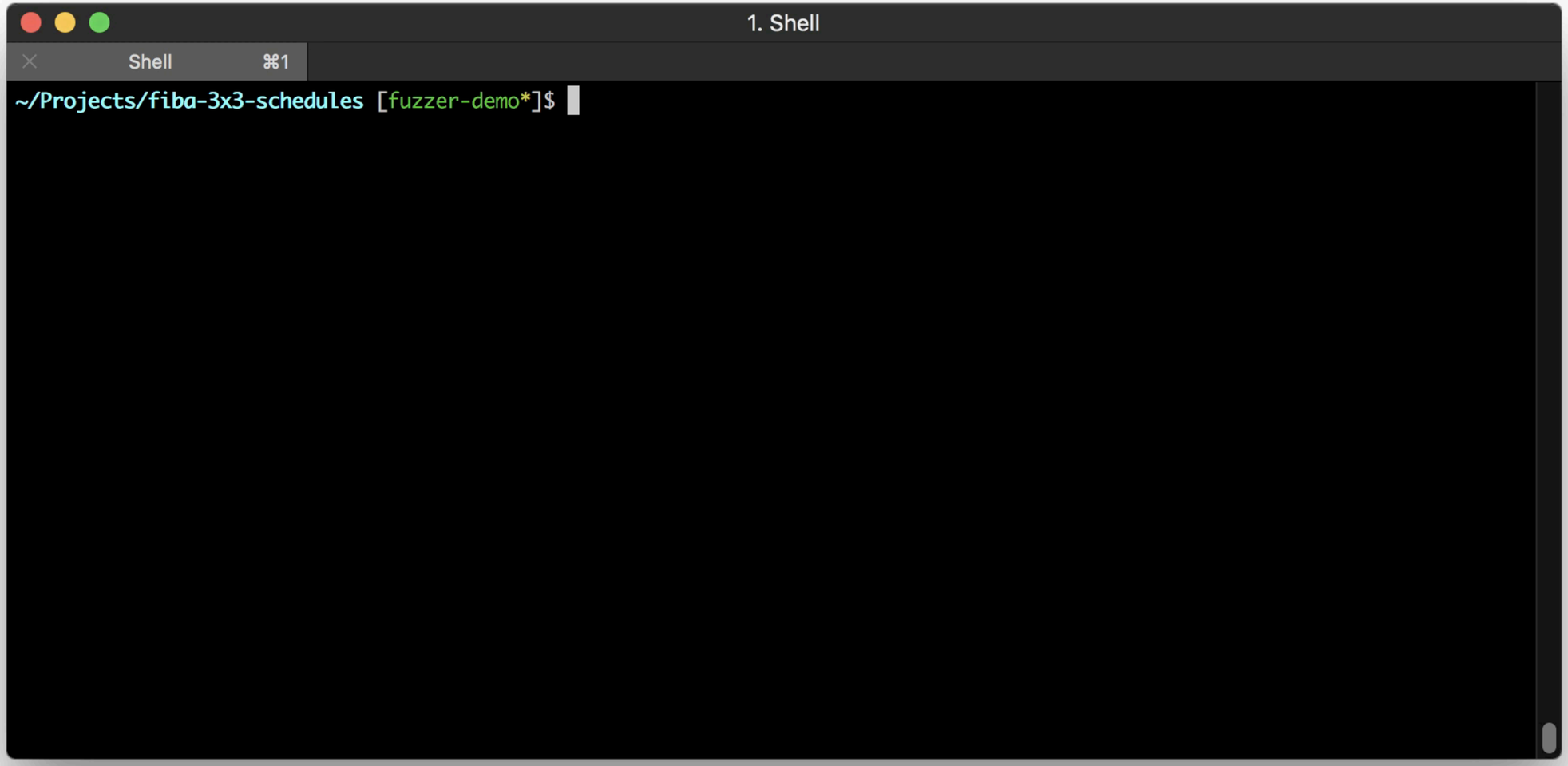
http://cswww.essex.ac.uk/CSP/ComputationalFinanceTeaching/CombinatorialExplosion.html

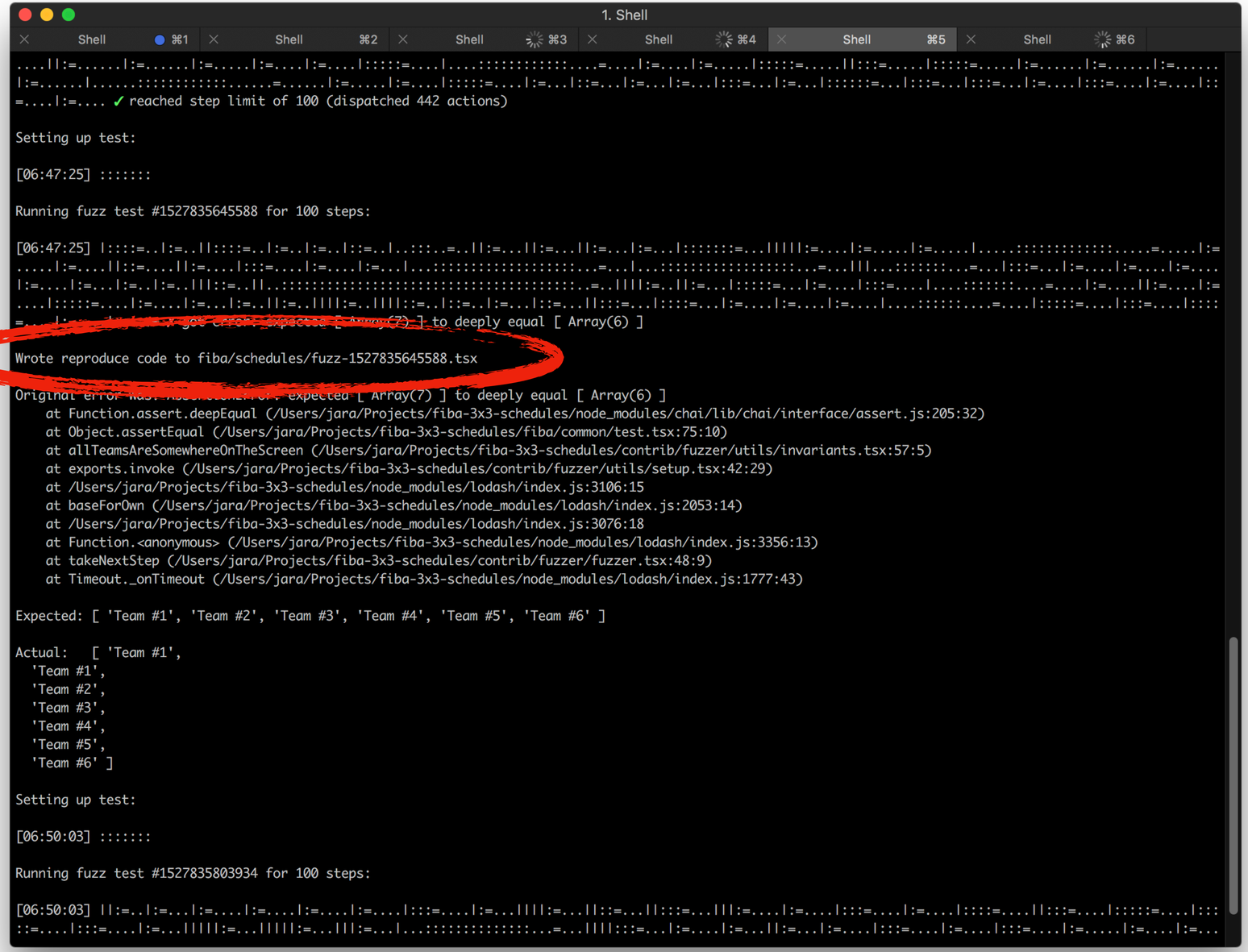
Clicking around manually?

Couldn’t we automate clicking around the UI, and making sure everything makes sense afterwards?

Clicking around = fuzzing

Making sense = property based testing


1. Shell
Shell ⌘1
~/Projects/fiba-3x3-schedules [fuzzer-demo*]$

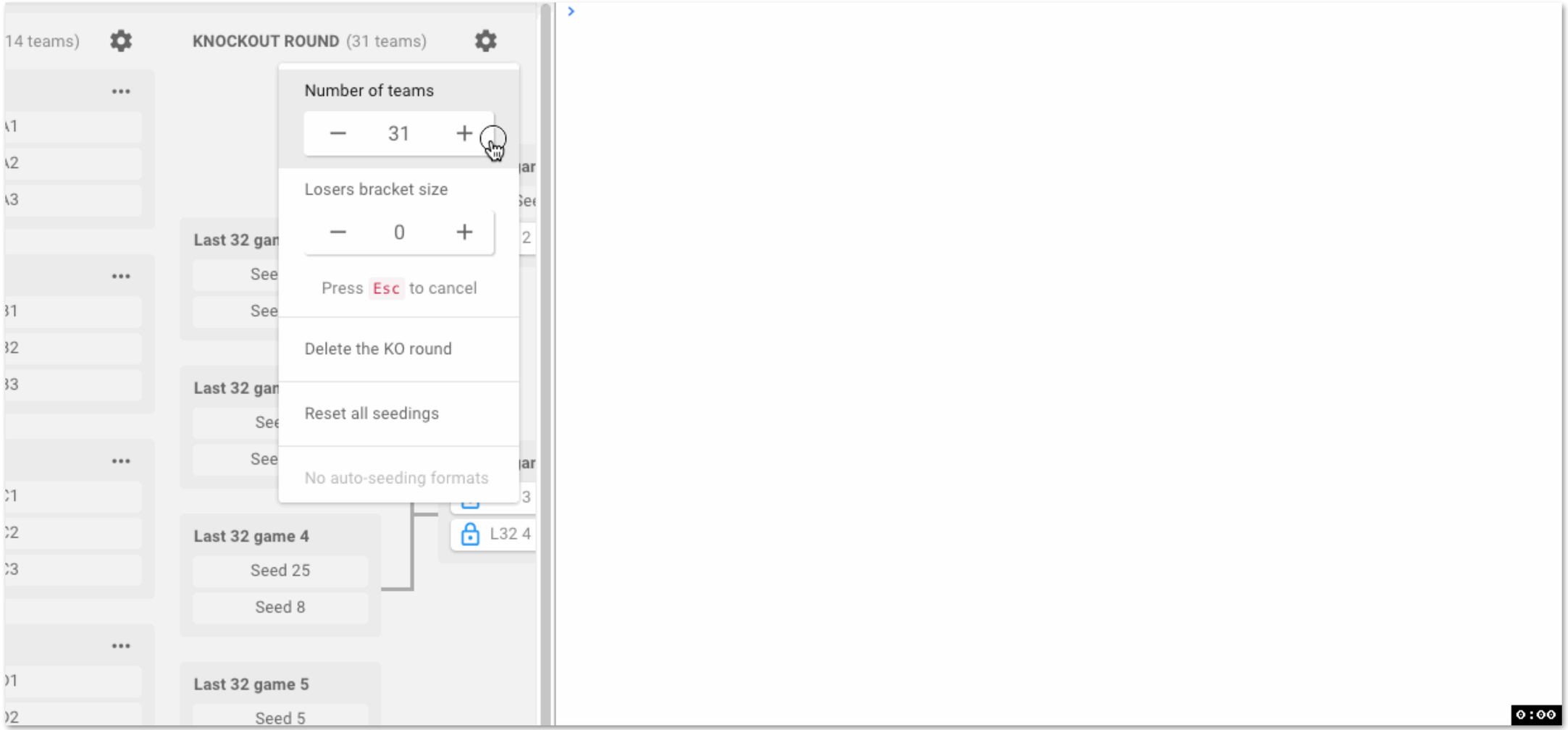

1. Shell
Shell ⌘1
Shell ⌘2
Shell ⌘3
Shell ⌘4
Shell ⌘5
Shell ⌘6
✓ reached step limit of 100 (dispatched 442 actions)
Setting up test:
[06:47:25] :::::::
Running fuzz test #1527835645588 for 100 steps:
to deeply equal [Array(6)]
Wrote reproduce code to fiba/schedules/fuzz-1527835645588.tsx
Original error was: AssertionError: expected [Array(7)] to deeply equal [Array(6)]
at Function.assert.deepEqual (/Users/jara/Projects/fiba-3x3-schedules/node_modules/chai/lib/chai/interface/assert.js:205:32)
at Object.assertEqual (/Users/jara/Projects/fiba-3x3-schedules/fiba/common/test.tsx:75:10)
at allTeamsAreSomewhereOnTheScreen (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/utils/invariants.tsx:57:5)
at exports.invoke (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/utils/setup.tsx:42:29)
at /Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3106:15
at baseForOwn (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:2053:14)
at /Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3076:18
at Function.<anonymous> (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3356:13)
at takeNextStep (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/fuzzer.tsx:48:9)
at Timeout._onTimeout (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:1777:43)
Expected: ['Team #1', 'Team #2', 'Team #3', 'Team #4', 'Team #5', 'Team #6']
Actual: ['Team #1',
'Team #1',
'Team #2',
'Team #3',
'Team #4',
'Team #5',
'Team #6']
Setting up test:
[06:50:03] :::::::
Running fuzz test #1527835803934 for 100 steps:

What kinds of issues did it find?

1. Plain old crashes

(with or without assertions)

What kinds of issues did it find?

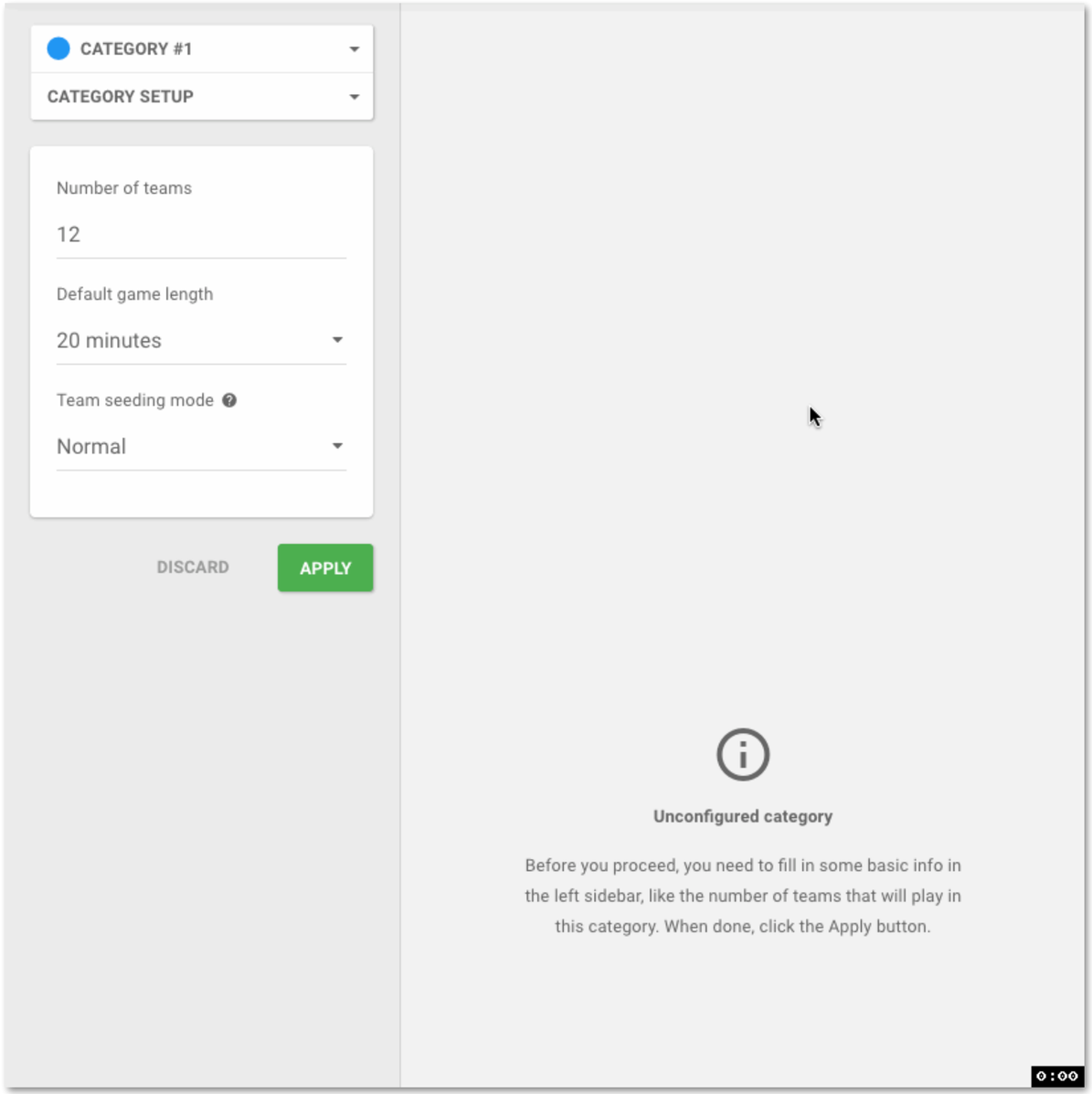


https://github.com/futurice/schedules-project/issues/519

2. Business rule violations

(assertions & invariants)

What kinds of issues did it find?

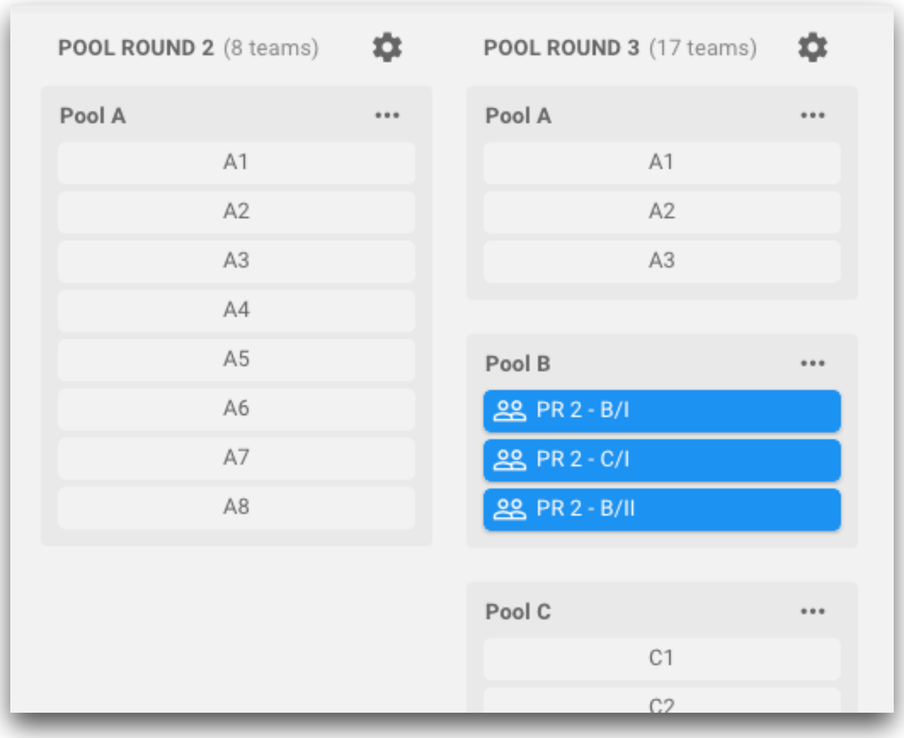


https://github.com/futurice/schedules-project/issues/533

3. False positives

(from the “synthetic” environment)

What kinds of issues did it find?



https://github.com/futurice/schedules-project/issues/524

How does it work?

How does it work?

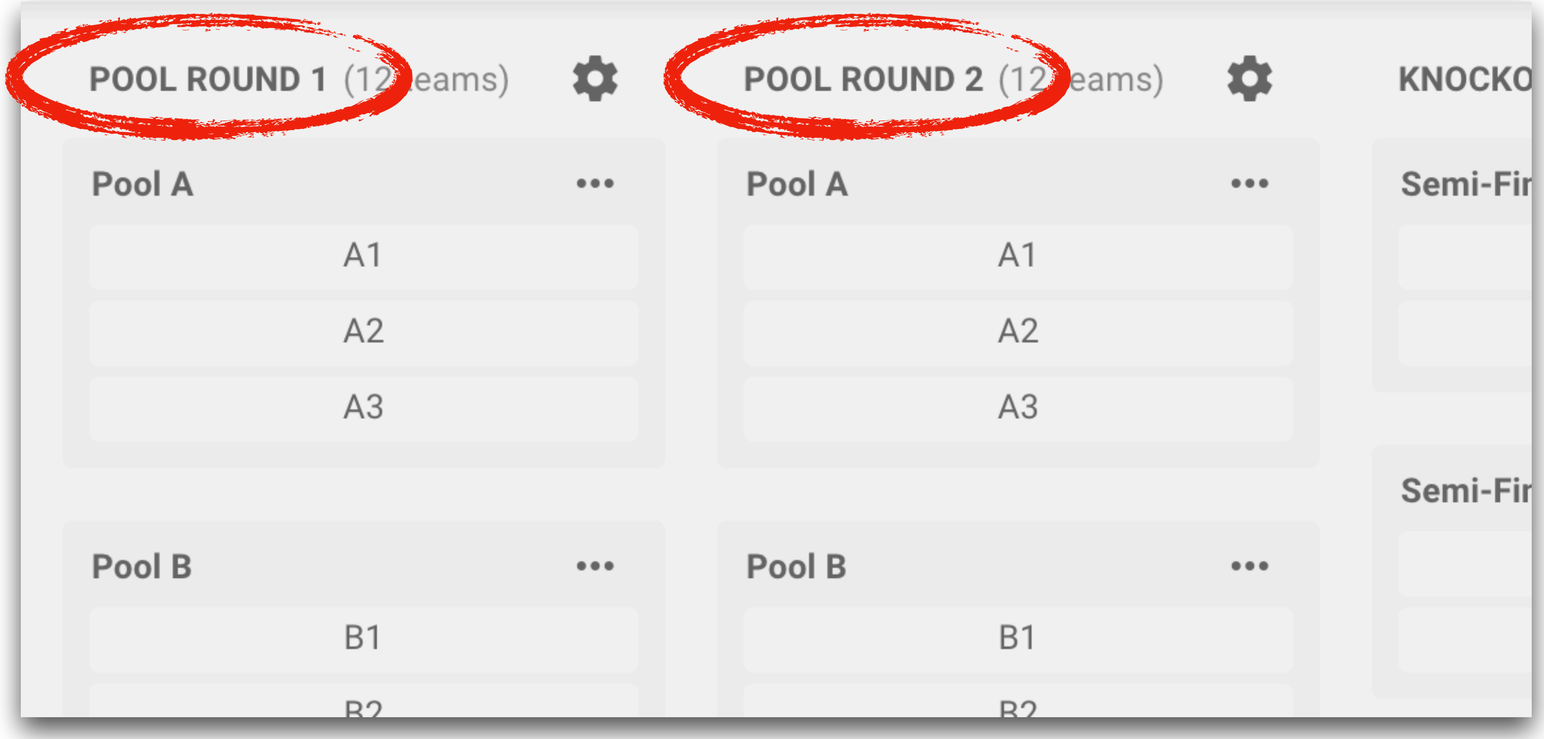
1. Setup
2. Getters
3. Steps
4. Invariants
5. Repro
6. Main

```
export function prepareTestApp() {

  let dispatchedActions = List<FluxActionModel>();
  const app: App = createTestApp(undefined, action => {
    echo(PROGRESS_MAIN_APP_DISPATCH);
    dispatchedActions = dispatchedActions.push(action);
  });
  const $: AugmentedCheerio = createTestRenderer(FibaSchedulesUi, app);

  app.actions.configActions.eventSelected(EVENT_ID);
  app.actions.routeActions.browserNavigated(`/categories/${DIVISION_ID}/setup`);
  app.actions.scheduleActions.fetchEventDataSucceeded(getInitialPayload());
  app.actions.editActions.divisionSetupUpdated([ DIVISION_ID, 'AmountOfTeams', random(2, 32) ]);
  app.actions.editActions.divisionSetupApplied(DIVISION_ID);
  app.actions.editActions.divisionSetupUpdated([ OTHER_DIVISION_ID, 'AmountOfTeams', random(2, 32) ]);
  app.actions.editActions.divisionSetupApplied(OTHER_DIVISION_ID);

  return {
    app,
    $,
    getDispatchedActions: () => dispatchedActions,
    runId: Date.now() + '',
  };

}
```

How does it work?

1. Setup
2. **Getters**
3. Steps
4. Invariants
5. Repro
6. Main

```
// @example [ "Pool Round" ]
// @example [ "Pool Round 1", "Pool Round 2" ]
visiblePoolRoundNames(): string[] {
  return $.toEls('.DivisionPreviewPanel-round:not(.DivisionPreviewPanel-koRound) .SubHeading-title')
    .map(el => el.text());
},
```



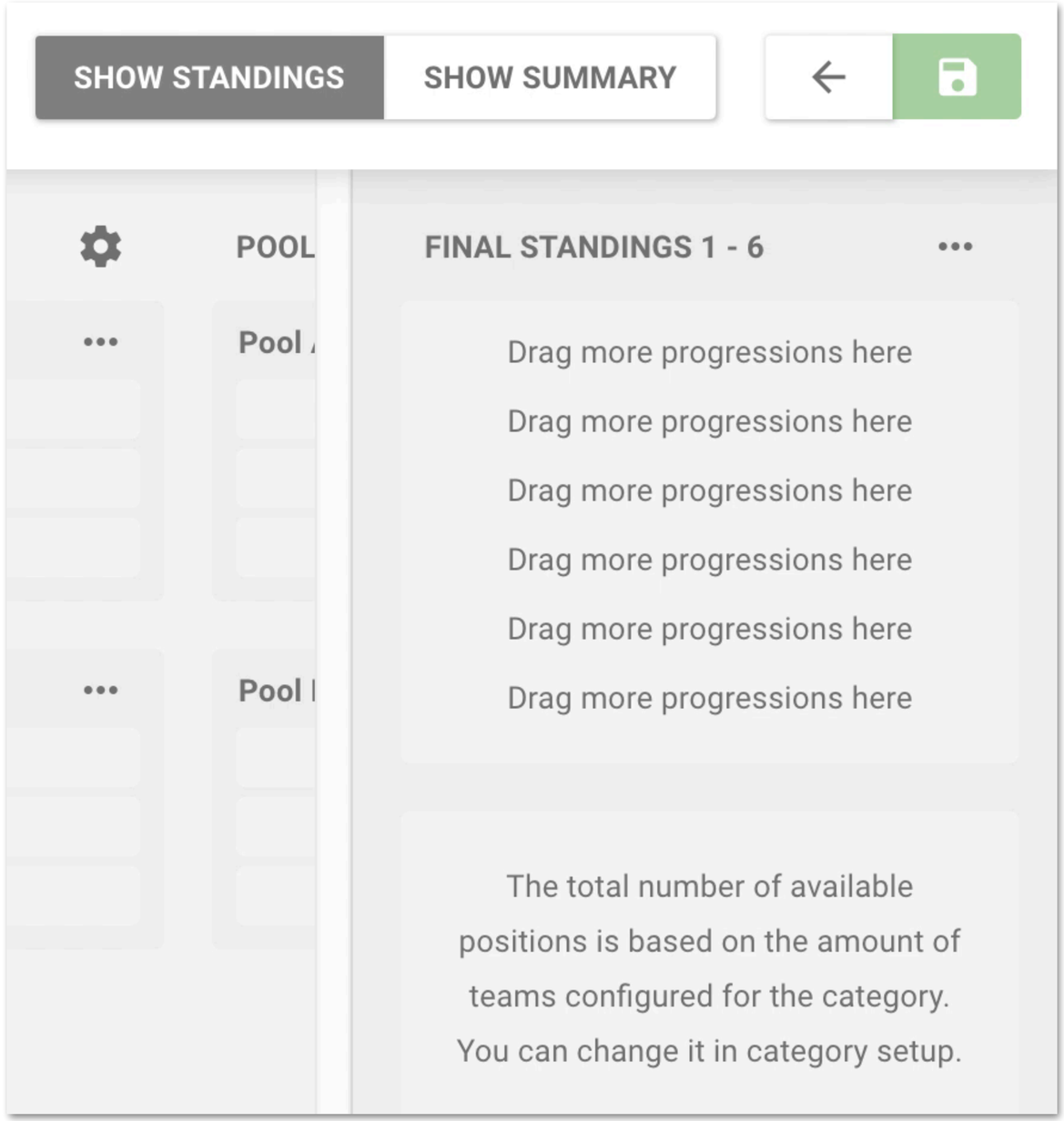
How does it work?

1. Setup
2. **Getters**
3. Steps
4. Invariants
5. Repro
6. Main

```
get.visiblePoolRoundNames();
get.visiblePoolRoundTeamCounts();
get.expectedRegularProgressionsPerRound();
get.visibleTeamCardNames();
get.visibleRoundIds();
get.visiblePoolGroupIds();
get.availableDropTargets();
get.availableSelectables();
get.teamsAvailableForSeeding();
get.koRoundExists();
get.koRoundTeamCount();
get.seedablePositionsInKoTree();
get.divisionTeamCount();
get.directStandingsBounds();
get.possibleDirectStandingsSplits();
get.possibleDirectStandingsMerges();
get.possibleDirectStandingsOrderChanges();
get.finalStandingsPositions();
get.visibleGameCards();
get.summarySidebarStats();
get.followUpGameIds();
get.deletedGameIds();
```

How does it work?

1. Setup
2. **Getters**
3. Steps
4. Invariants
5. Repro
6. Main



How does it work?

1. Setup
2. **Getters**
3. Steps
4. Invariants
5. Repro
6. Main

```
// @example [ [ "Registrations", 6, "teams" ], ... ]
summarySidebarStats(): List<Tuple3<string, number, string>> {
  const { app2, $2 } = forkTestApp(app);
  app2.actions.configActions.divisionStatsToggled();
  return fromJS(
    $2.toEls('.DivisionStatsPanel > :nth-child(2) .DivisionStatsPanel-tuple')
      .map($el => $el.text())
      .map(txt => txt.replace(/([0-9.]+)\s*/, '|$1|'))
      .map(txt => txt.split('|'))
      .map(parts => parts.map((v, i) => i === 1 ? parseFloat(v) : v.trim())),
  );
},
```

How does it work?

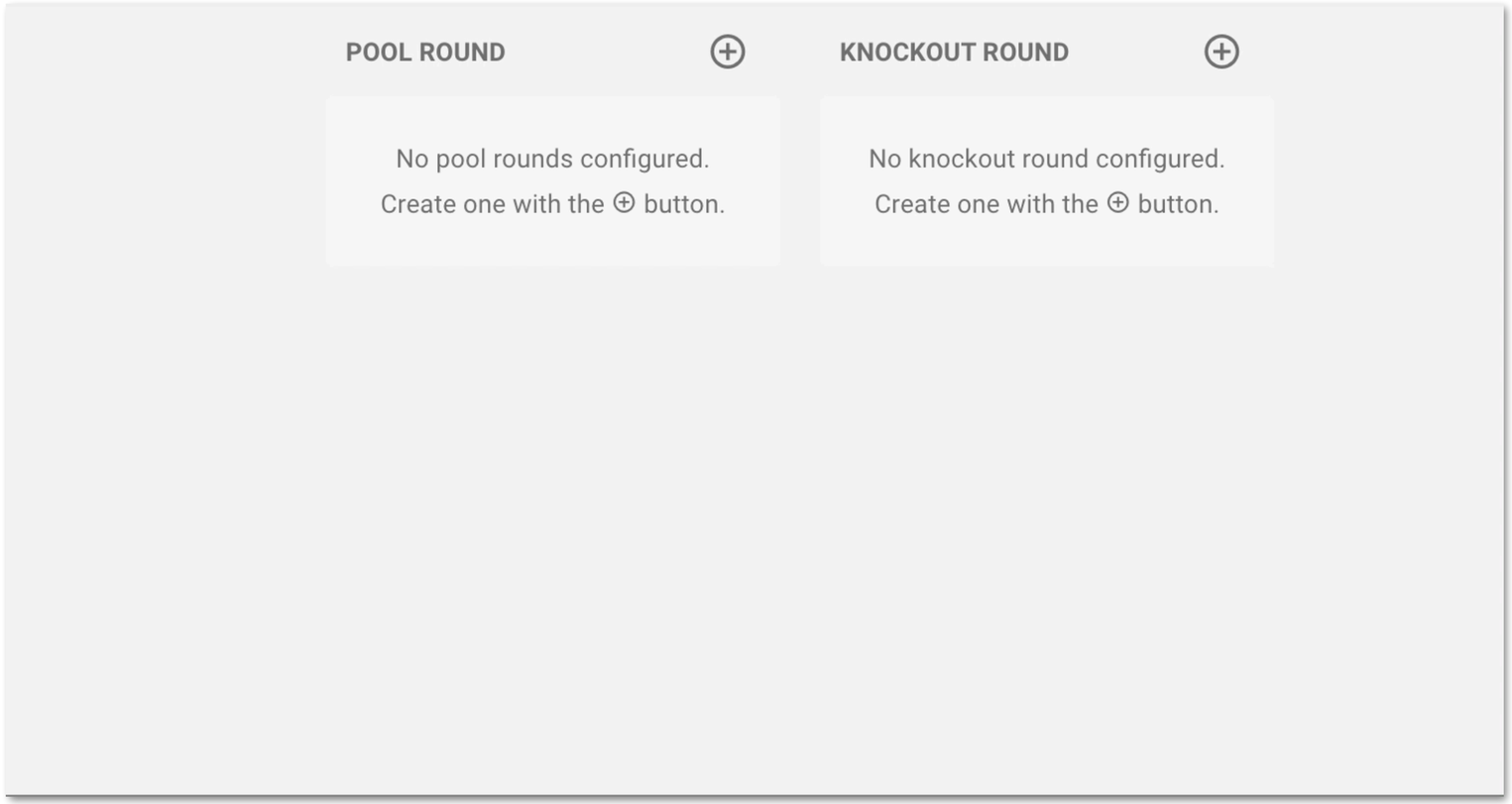
1. Setup
2. **Getters**
3. Steps
4. Invariants
5. Repro
6. Main

```
export function forkTestApp(app: App) {
  const app2: App = createTestApp(
    app['reduxStore'].getState(), // start from the cloned state of the main app
    () => echo(PROGRESS_FORKED_APP_DISPATCH),
  );
  const $2: AugmentedCheerio = createTestRenderer(FibaSchedulesUi, app2);
  return { app2, $2 };
}
```

How does it work?

1. Setup
2. Getters
3. **Steps**
4. Invariants
5. Repro
6. Main

```
addNewPoolRound() {
  const existingRounds = get.visibleRoundIds();
  if (existingRounds.length >= 10) return; // let's not get excessive here
  app.actions.scheduleActions.newPoolRoundAdded(divisionId);
},
```



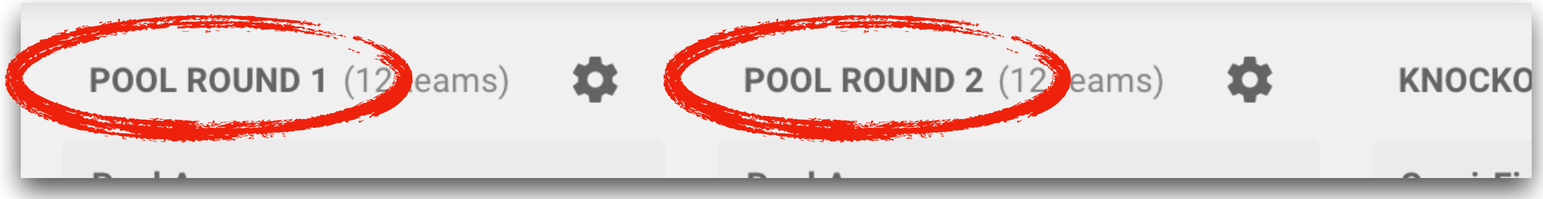
How does it work?

1. Setup
2. Getters
3. **Steps**
4. Invariants
5. Repro
6. Main

```
steps.reconfigureDivisionSetup();
steps.addNewPoolRound();
steps.removeExistingPoolRound();
steps.addKoPhaseIfMissing();
steps.removeExistingKoPhase();
steps.toggleFirstRoundAndQd();
steps.autoSeedTeamsToFirstRound();
steps.resetTeamSeedingsForARound();
steps.reconfigurePoolRound();
steps.reconfigureIndividualPool();
steps.reconfigureKoRound();
steps.splitDirectStandings();
steps.mergeDirectStandings();
steps.moveDirectStandings();
steps.dragThingsAround();
steps.fuzzUnrelatedDivision();
```

How does it work?

1. Setup
2. Getters
3. Steps
4. **Invariants**
5. Repro
6. Main




```
visiblePoolRoundHeadingsAreContiguous() {
  const names = get.visiblePoolRoundNames();
  if (names.length === 1) {
    assertEqual(
      names[0],
      'Pool Round',
    );
  } else {
    assertEqual(
      names,
      range(1, names.length + 1).map(i => `Pool Round ${i}`),
    );
  }
},
```

How does it work?

1. Setup
2. Getters
3. Steps
4. **Invariants**
5. Repro
6. Main

```
invariants.eachRegularProgressionIsVisibleExactlyOnce();
invariants.visiblePoolRoundHeadingsAreContiguous();
invariants.eachPoolRoundContainsAsManySeedablePositionsAsItReportsHavingTeams();
invariants.allTeamsAreSomewhereOnTheScreen();
invariants.koPhaseContainsCorrectSeedablePositions();
invariants.directStandingsAreContiguous();
invariants.correctNumberOfFinalStandings();
invariants.finalStandingsBoundsAddUp();
invariants.summaryPanelGameCountsAreLegit();
invariants.followUpGameIdsAreNotDeadEnds();
```

How does it work?

1. Setup
2. Getters
3. Steps
4. Invariants
5. **Repro**
6. Main

```
export function writeCodeToReproduce(err: Error, id: string, actions: List<FluxActionModel>): void {
  const file = `fiba/schedules/fuzz-${id}.tsx`;
  echo(`Wrote reproduce code to ${file}\n`);
  writeFileSync(file, `/*\n${expandErrorMessage(err)}\n*/\n` + getCodeToReproduce(actions));
}

function getCodeToReproduce(actions: List<FluxActionModel>): string {
  return `
    // APPLICATION SETUP:

    import 'fiba/schedules/utils/browserPointerInput';
    import FibaSchedulesUi from 'fiba/schedules/ui/FibaSchedulesUi';
    import { renderAppRoot } from 'fiba/common/react';
    import { fromJS } from 'fiba/schedules/utils/dto';
    import { createTestApp } from 'fiba/common/test';
    import { initOutsideClicks } from 'fiba/schedules/utils/browser';

    const app = createTestApp();

    initOutsideClicks(app);

    renderAppRoot(FibaSchedulesUi, app, document.querySelector('#fiba-app-root'));

    // ACTION REPLAY:
    \n${serializeActions(actions).join('\n')}
  `;
}

function serializeActions(actions: List<FluxActionModel>): List<string> {
  return actions.map(action => `app.actions.${action.type}(${serializePayload(action.payload)});\n`);
}
```

How does it work?

1. Setup
2. Getters
3. Steps
4. Invariants
5. Repro
6. **Main**

```
function run() {
  echo(`Setting up test:\n\n${ts()}`);
  const { app, $, getDispatchedActions, runId } = prepareTestApp();
  const getters = bindGetters(app, $);
  const steps = bindSteps(app, $, getters);
  const invariants = bindInvariants(getters);
  const getActionCount = () => getDispatchedActions().size;
  let keepStepping = STEPS_PER_RUN;
  echo(`\n\nRunning fuzz test #${runId} for ${keepStepping} steps:\n\n${ts()}`);
  return new Promise(resolve => {
    takeNextStep();
    function takeNextStep() {
      try {
        // Choose and take a random step:
        const prevActionCount = getActionCount();
        sample(steps)();
        const nextActionCount = getActionCount();
        if (nextActionCount === prevActionCount) {
          return defer(takeNextStep); // the step dispatched nothing -> no point in further checks
        }
        // Run all invariant checks:
        each(invariants, invoke);
        // Decide whether to keep going or if we're finished with this run:
        if (keepStepping--) {
          defer(takeNextStep); // schedule next step
        } else {
          echo(` ${GREEN_TICK} reached step limit of ${STEPS_PER_RUN} (dispatched ${getDispatchedActions().size}`
          resolve(); // reached step limit
        }
      } catch (err) {
        // Report the error that was found:
        echo(` ${RED_CROSS} got error: ${simplifyErrorMessage(err)}\n\n`);
        writeCodeToReproduce(err, runId, getDispatchedActions());
        echo(`\nOriginal error was: ${expandErrorMessage(err)}\n\n`);
        resolve(); // done with this run
      }
    }
  });
}
```

Lessons learned?

1. Should have started earlier

2. Simplifier was complicated & useless

Lessons learned?

```
Running fuzz test #1517478095379 for 255 steps:

[09:41:35] ..................................................... ✗ got error: expected [ Array(5) ] to deeply equal [ Array(6) ]

Wrote reproduce code to fiba/schedules/index-original-1517478095379.tsx

Simplifying 66 actions:

[09:42:26] ..................................................... ✓ couldn't simplify further (25 attempts remaining)
[09:43:11] ..................................................... ✓ eliminated 2 irrelevant actions
[09:43:55] ..................................................... ✓ couldn't simplify further (25 attempts remaining)
[09:44:12] ..................................................... ✓ eliminated 6 irrelevant actions
[09:44:46] ..................................................... ✓ eliminated 7 irrelevant actions
[09:45:18] ..................................................... ✓ couldn't simplify further (25 attempts remaining)
[09:45:48] ..................................................... ✓ couldn't simplify further (24 attempts remaining)
[09:46:20] ..................................................... ✓ couldn't simplify further (23 attempts remaining)
[09:46:41] ..................................................... ✓ eliminated 5 irrelevant actions
[09:47:12] ..................................................... ✓ couldn't simplify further (25 attempts remaining)
[09:47:27] ..................... ✗ unrelated error: Assertion failed: Expected a Record of type "RoundDto", instead got: undefined
[09:47:42] ..................... ✓ couldn't simplify further (24 attempts remaining)
[09:47:54] ..................... ✗ unrelated error: Cannot read property 'RoundCode' of undefined
[09:48:01] ..................................................... ✓ couldn't simplify further (23 attempts remaining)
[09:48:31] ..................................................... ✓ couldn't simplify further (22 attempts remaining)
[09:48:59] ..................................................... ✓ couldn't simplify further (21 attempts remaining)
[09:49:12] ..................................................... ✓ eliminated 8 irrelevant actions
[09:49:24] ..................................................... ✓ couldn't simplify further (25 attempts remaining)
[09:49:36] ..................................................... ✓ couldn't simplify further (24 attempts remaining)
[09:49:48] ................. ✗ unrelated error: Cannot read property 'RoundCode' of undefined
[09:49:49] ..................... ✓ couldn't simplify further (23 attempts remaining)
[09:49:58] ..................................................... ✓ eliminated 2 irrelevant actions
[09:50:10] ..................... ✓ couldn't simplify further (25 attempts remaining)
[09:50:21] ..................... ✓ couldn't simplify further (24 attempts remaining)
[09:50:29] ..................................... ✓ eliminated 11 irrelevant actions
[09:50:44] ..................... ✓ couldn't simplify further (25 attempts remaining)
[09:50:57] ..................................... ✓ eliminated 2 irrelevant actions
[09:51:11] ..................... ✓ couldn't simplify further (25 attempts remaining)
[09:51:26] ..................... ✓ eliminated 2 irrelevant actions
[09:51:40] ............... ✓ couldn't simplify further (25 attempts remaining)
[09:51:41] ..................... ✓ couldn't simplify further (24 attempts remaining)
[09:51:55] ................. ✓ couldn't simplify further (23 attempts remaining)
[09:51:57] ................. ✓ couldn't simplify further (22 attempts remaining)
[09:51:58] ..................... ✓ eliminated 2 irrelevant actions
[09:52:12] ................. ✓ couldn't simplify further (25 attempts remaining)
[09:52:15] ..................... ✓ couldn't simplify further (24 attempts remaining)
[09:52:27] ..................... ✓ couldn't simplify further (23 attempts remaining)
[09:52:40] ..................... ✓ couldn't simplify further (22 attempts remaining)
[09:52:55] ..................... ✓ couldn't simplify further (21 attempts remaining)
[09:53:07] ..................... ✓ eliminated 5 irrelevant actions
[09:53:10] ............... ✓ couldn't simplify further (25 attempts remaining)
[09:53:11] ................. ✓ couldn't simplify further (24 attempts remaining)
[09:53:13] ................. ✓ couldn't simplify further (23 attempts remaining)
[09:53:14] ................. ✓ couldn't simplify further (22 attempts remaining)
[09:53:15] ................. ✓ couldn't simplify further (21 attempts remaining)
[09:53:18] ................. ✓ couldn't simplify further (20 attempts remaining)
[09:53:19] ................ ✓ couldn't simplify further (19 attempts remaining)
[09:53:20] ............... ✓ couldn't simplify further (18 attempts remaining)
[09:53:21] ................ ✓ couldn't simplify further (17 attempts remaining)
[09:53:22] ................. ✓ couldn't simplify further (16 attempts remaining)
[09:53:25] ................. ✓ couldn't simplify further (15 attempts remaining)
[09:53:26] ................. ✓ couldn't simplify further (14 attempts remaining)
[09:53:28] ................ ✓ couldn't simplify further (13 attempts remaining)
[09:53:31] ................. ✓ couldn't simplify further (12 attempts remaining)
[09:53:34] ................. ✓ couldn't simplify further (11 attempts remaining)
[09:53:35] ................. ✓ couldn't simplify further (10 attempts remaining)
[09:53:38] ................. ✓ couldn't simplify further (9 attempts remaining)
[09:53:39] ................ ✓ couldn't simplify further (8 attempts remaining)
[09:53:40] ................ ✓ couldn't simplify further (7 attempts remaining)
[09:53:43] ................ ✓ couldn't simplify further (6 attempts remaining)
[09:53:46] ................ ✓ couldn't simplify further (5 attempts remaining)
[09:53:49] ................ ✓ couldn't simplify further (4 attempts remaining)
[09:53:52] ............... ✓ couldn't simplify further (3 attempts remaining)
[09:53:52] ............... ✓ couldn't simplify further (2 attempts remaining)
[09:53:53] ................. ✓ couldn't simplify further (1 attempts remaining)
[09:53:54] ................. ✓ couldn't simplify further (0 attempts remaining)

Done: simplified 66 actions into 14 actions
Wrote reproduce code to fiba/schedules/index-simplified-1517478095379.tsx

Original error was: AssertionError: expected [ Array(5) ] to deeply equal [ Array(6) ]
    at Function.assert.deepEqual (/Users/jara/Projects/fiba-3x3-schedules/node_modules/chai/lib/chai/interface/assert.js:205:32)
    at Object.assertEqual (/Users/jara/Projects/fiba-3x3-schedules/fiba/common/test.tsx:74:10)
    at allTeamsAreSomewhereOnTheScreen (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/utils/invariants.tsx:58:5)
    at exports.invoke (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/utils/setup.tsx:46:29)
    at /Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3106:15
    at baseForOwn (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:2053:14)
    at /Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3076:18
    at Function.<anonymous> (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:3356:13)
    at step (/Users/jara/Projects/fiba-3x3-schedules/contrib/fuzzer/fuzzer.tsx:29:9)
    at Timeout._onTimeout (/Users/jara/Projects/fiba-3x3-schedules/node_modules/lodash/index.js:1777:43)

Expected: [ 'Team #1', 'Team #2', 'Team #3', 'Team #4', 'Team #5', 'Team #6' ]

Actual: [ 'Team #1', 'Team #2', 'Team #3', 'Team #5', 'Team #6' ]
```

3. Automatic repro code generation is genuinely useful

4. Not that many false positives

5. Usefulness grows as a function of invariant & step count

Limitations?

1. Requires determinism, serializability & isolation

2. Can’t really test this:

“I just did X,
so now that I do Y,
it should Z”

3. Don’t mistake for exhaustive testing

Limitations?

+ fuzzer

=

https://www.currys.co.uk/gbuk/computing/laptops/laptops/apple-macbook-pro-13-space-grey-2017-10165801-pdt.html
http://marvelcinematicuniverse.wikia.com/wiki/Helicarrier

4. Doesn’t fit every project

5. Doesn’t generalise easily

Future work?

1. Running invariant checks in prod..?

2. Using to find performance bottlenecks..?

3. Using to find memory leaks..?

https://commons.wikimedia.org/wiki/File:Thats_all_folks.svg

Questions?