Extracting Runtime Monitors from Tests: An Overview and a Way Forward

Abigail Cauchi, Luke Chircop, Christian Colombo, Adrian Francalanza, Mark Micallef, and Gordon Pace

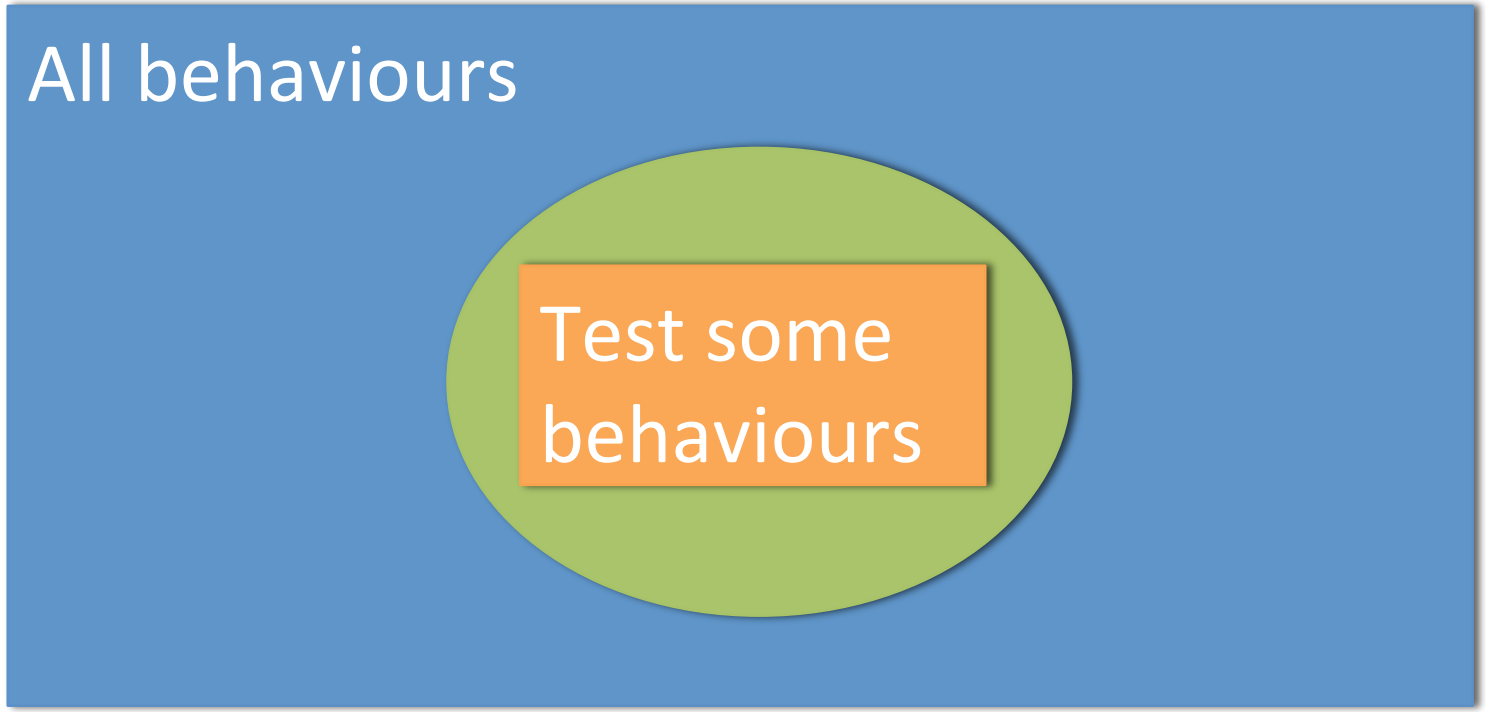
Project GOMTA financed by the Malta Council for Science & Technology through the National Research & Innovation Programme 2013

Why generate monitors from tests?

- Monitors can provide extra assurance at runtime
- Industry already invests a lot in testing
  (but little in runtime verification)
- Creating monitors after creating tests feels repetitive/waste

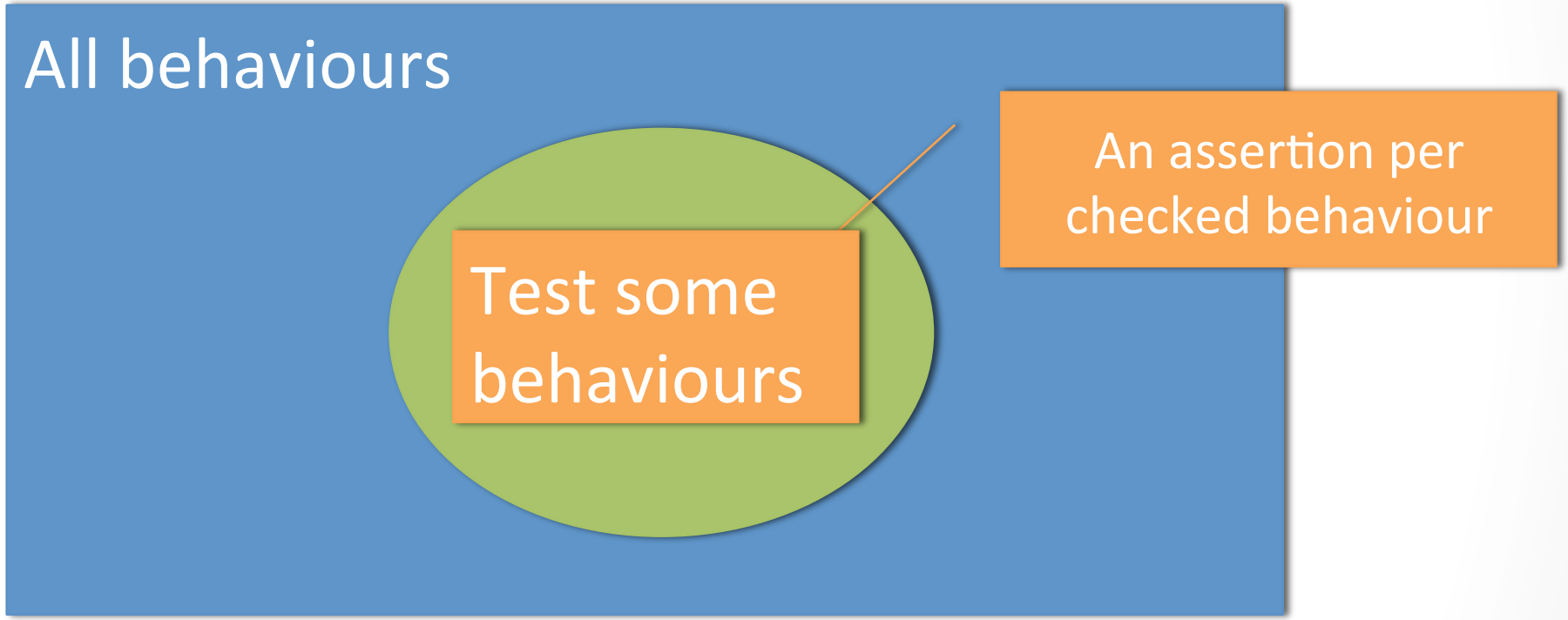
Verification – A language problem

Testing



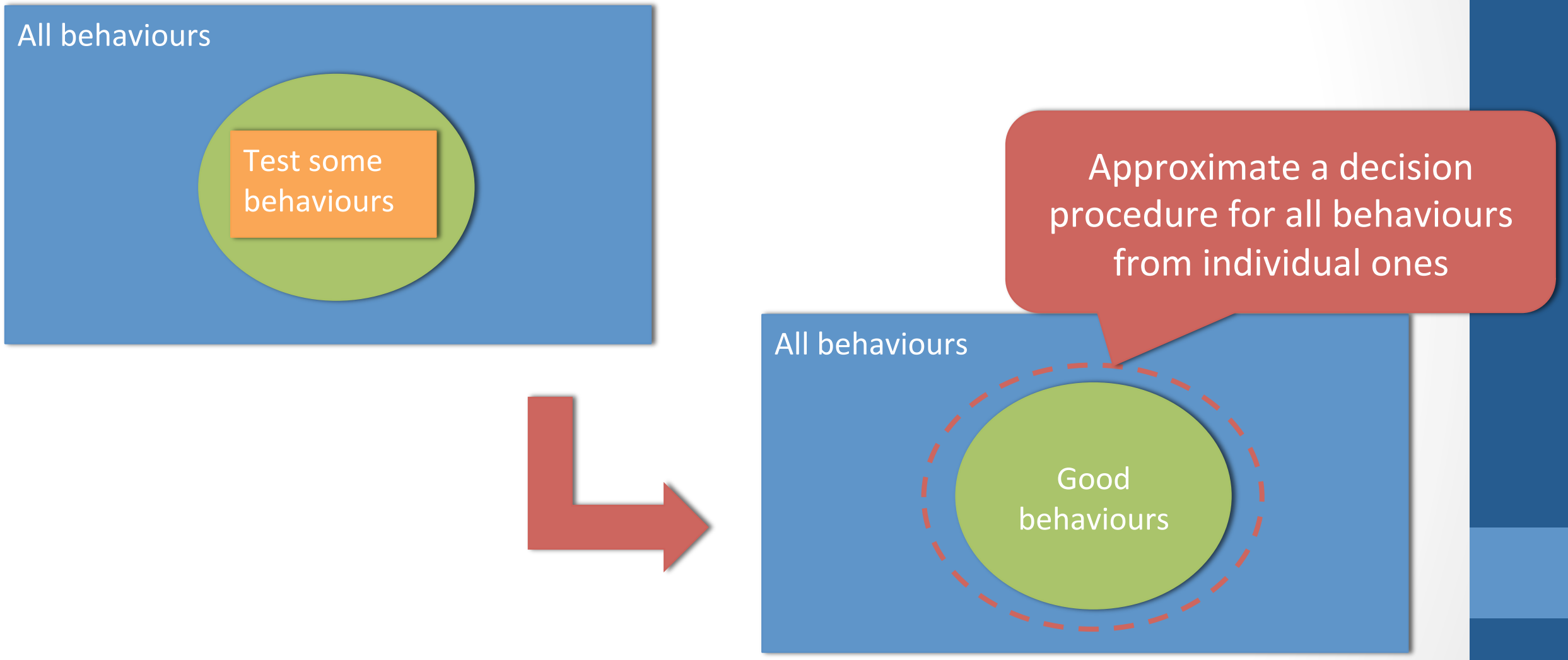
Runtime verification

Testing

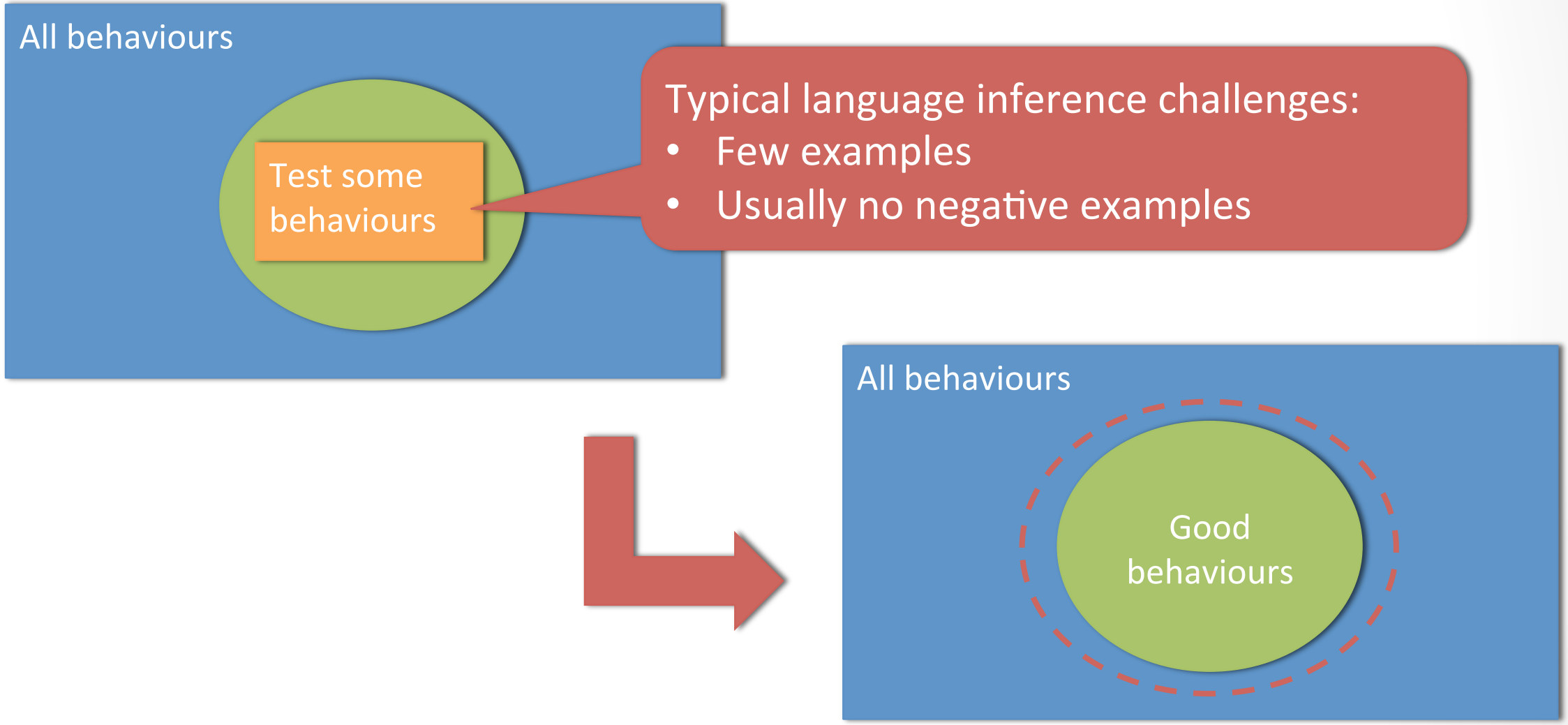

All behaviours
Test some behaviours
An assertion per checked behaviour

Runtime verification

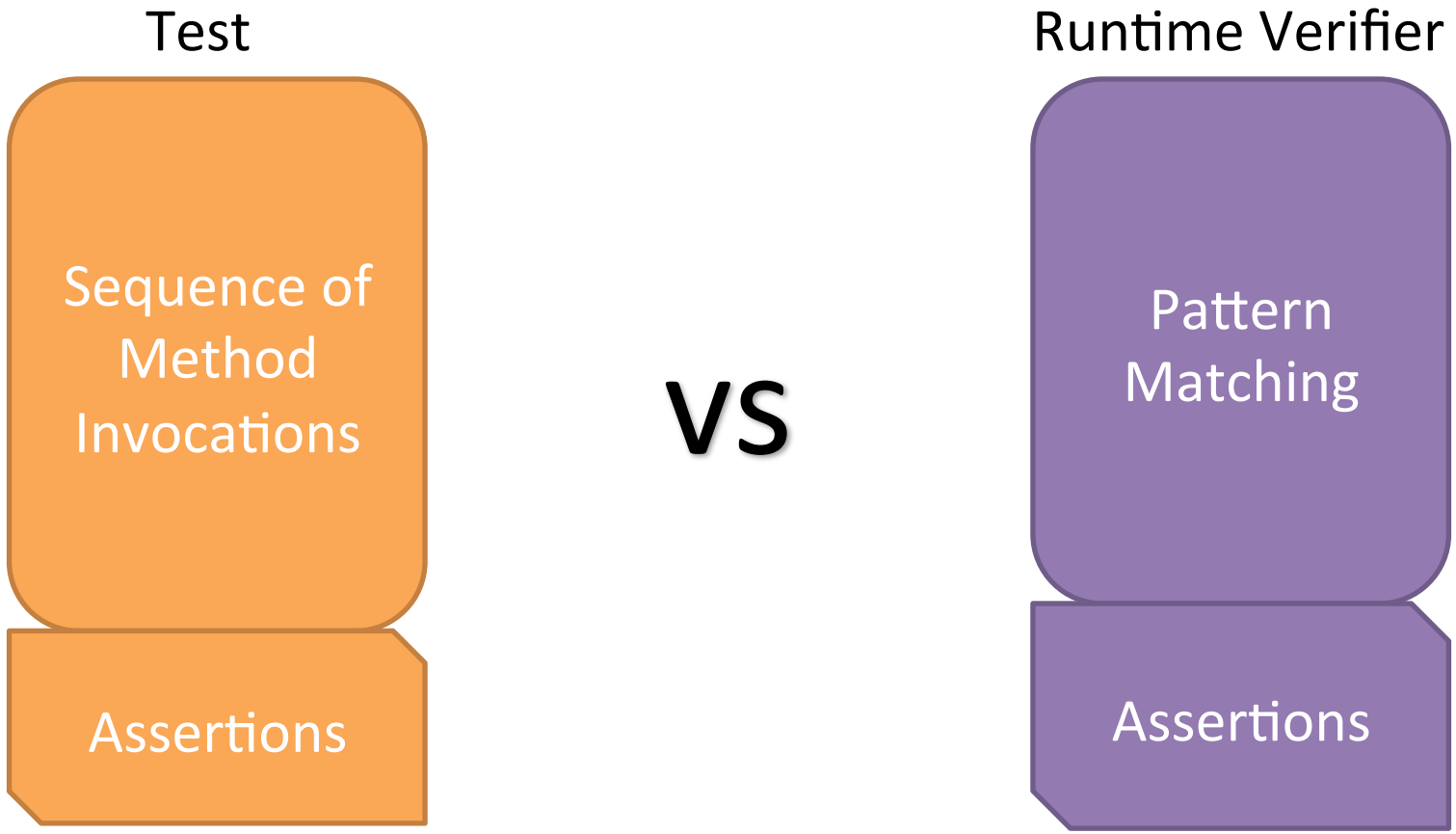
Generating runtime verifiers from tests



Why is it difficult?



Why not use test assertions directly?



Test assertions are typically very specific

```
@Test
public void testWithdraw(){
     Account a = new Account();
     a.setBalance(100);
     a.withdraw(60);
     assertEquals(a.getBalance(),40);
}
```

Idealistic test assertions

```
@Test
public void testWithdraw(){
     initialBalance = 100;
     withdrawAmount = 60;
     Account a = new Account();
     a.setBalance(initialBalance);
     a.withdraw(withdrawAmount );
assertEquals(a.getBalance(),initialBalance-withdrawAmount);
}
```

What if you insist on using assertions?

- There might be other hidden assumptions:
  - Assumptions on the global state (shared data structures, files, etc)
  - Assumptions on the control/data flow leading up to the assertion (test setup, method call sequence in test, etc)

A look at related approaches

Testing to more “generalised” testing

1. EUnit → QuickCheck (Thomas Arts et al.)
2. Gherkin → QuickCheck (Christian Colombo et al.)

Testing to RV

3. QuickCheck → Larva (Gordon Pace and Kevin Falzon)

Testing to Regression testing/Debugging

4. Invariant detection with Daikon (Pastore et al.)

1. EUnit → QuickCheck

- Generates QuickCheck automaton from sequences of method calls
- Uses algorithm to learn automata
- Uses learned automaton to improve testsuite

Points to consider

- Assumes the availability of negative traces
  - Not usually present in testsuites
- Suitable for testing, probably also for RV if negative traces are available

2. Gherkin → QuickCheck

- Similar to previous but state identification is easier as more explicit in Gherkin tests

Standard Business Specifications

```
Scenario: Model definition for myHealth - Doctors Section

Given I am on the "start state"
When I "login as a doctor"
Then I should go to the "doctors landing page"

Given I am on the "doctors landing page"
When I "click on Appointments"
Then I should go to the "appointments page"

Given I am on the "doctors landing page"
When I "click on Case Summaries"
Then I should go to the "case summaries page"

Given I am on the "doctors landing page"
When I "click on Laboratory Results"
Then I should go to the "lab results page"

Given I am on the "doctors landing page"
When I "click on Medical Image Reports"
Then I should go to the "medical image reports page"
```

Standard Business Specifications

```
Scenario: Model definition for myHealth - Doctors Section

Given I am on the "start state"
When I "login as a doctor"
Then I should go to the "doctors landing page"

Given I am on the "doctors landing page"
When I "click on Appointments"
Then I should go to the "appointments page"

Given I am on the "doctors landing page"
When I "click on Case Summaries"
Then I should go to the "case summaries page"

Given I am on the "doctors landing page"
When I "click on Laboratory Results"
Then I should go to the "lab results page"

Given I am on the "doctors landing page"
When I "click on Medical Image Reports"
Then I should go to the "medical image reports page"
```

States

Standard Business Specifications

```
Scenario: Model definition for myHealth - Doctors Section

Given I [illegible] the "start state"
When I "login as [illegible]
Then I [illegible] to th "doctors landing page"

Given I am on the "doctors landing page"
When I "click on Appointments"
Then I should go to the "appointments page"

Given I am on the "doctors landing page"
When I "click on Case Summaries"
Then I [illegible] summaries page"

Given I am on the "doctors landing page"
When I "click on Laboratory Results"
Then I should go to the "lab results page"

Given I am on the "doctors landing page"
When I "click on Medical Image Reports"
Then I should go to the "medical image reports page"
```

States

Actions

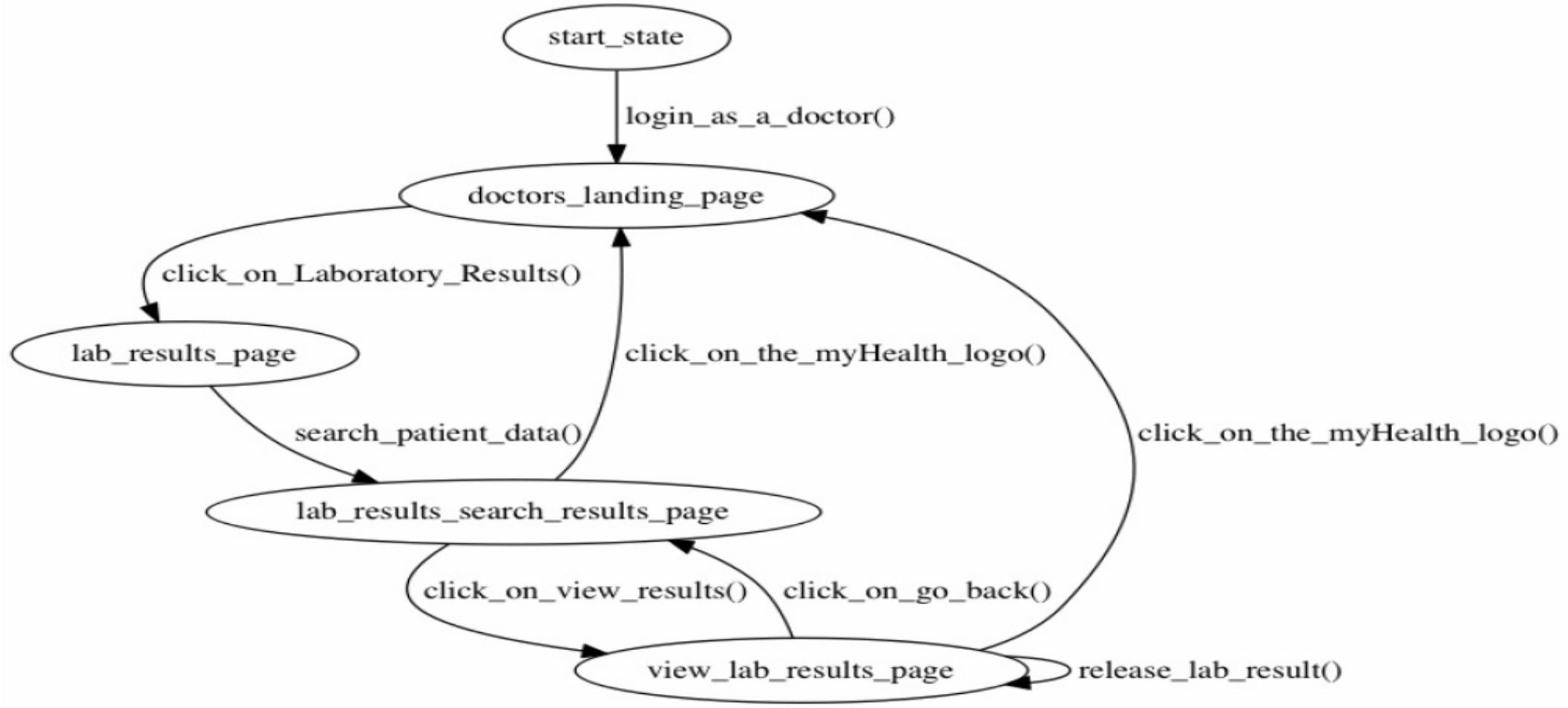
Standard Business Specifications

```
Given I am on the "doctors landing page"
When I "click on Laboratory Results"
Then I should go to the "lab results page"

Given I am on the "lab results page"
When I "search patient data"
Then I                    "lab results search result
and the result should be "true" when "no data found"
and th                    true" when "ok"

Given I am on the "lab results search results page"
When I "click on the myHealth logo"
Then I should go to the "doctors landing page"

G                         results search results page
and the result is "ok"
W                         results"
Then I should go to the "view lab results page"

Given I am on the "view lab results page"
When I "release lab result"
Then I should remain on the "view lab results page"

Given I am on the "view lab results page"
When I "click on go back"
Then I should go to the "lab results search results page"
```

Post Condition

Pre Condition

Automatically Generated QC Model



Points to consider

- The higher the testing level, the more useful for RV

3. QuickCheck → Larva

- Translates QC automata into Larva script
- Main challenge is to make sure you match corresponding entry and exit points
  - recursiveMethod() -entry
    - recursiveMethod() -entry
    - recursiveMethod() -exit
  - recursiveMethod() -exit

Points to consider

- It is easy to go from Model-Based Testing to RV
- Model-Based Testing not very commonplace

4. Invariant detection with Daikon

- Detect invariants from running testsuite
- Filter out invariants which no longer hold in modified testsuite
- Use model checking to detect invariants which are violated in update

Points to consider

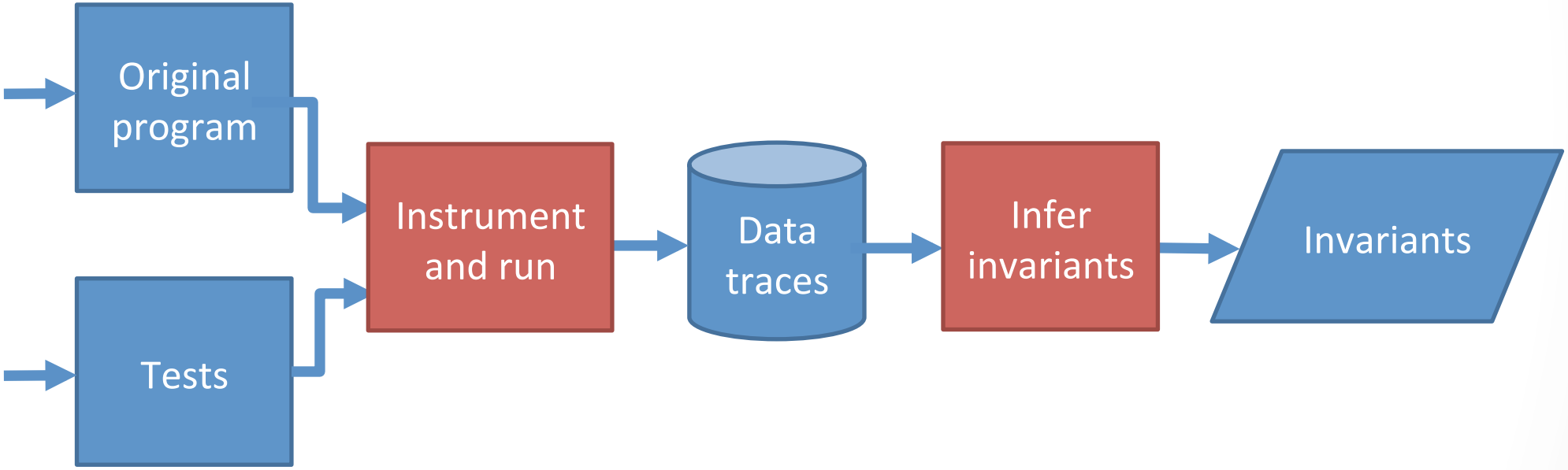
- How can we adapt it to RV?

Approach 1: Gherkin → QC → Larva

- We know how to go from Gherkin to QC
- We know how to go from QC to Larva
- Go from Gherkin to Larva

Approach 2: Infer invariants

- Daikon – an invariant generation tool

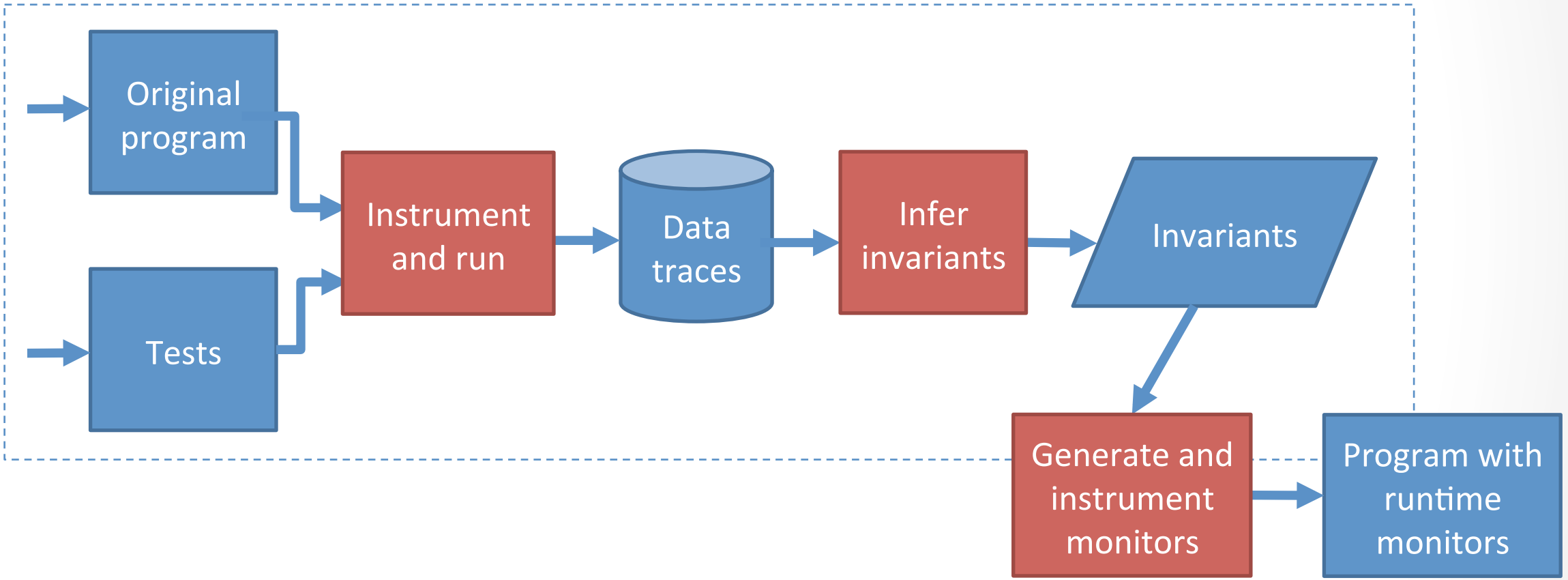


Approach 2: Infer invariants

- Daikon – an invariant generation tool

```
==========================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
==========================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
==========================================================================
```

Approach 2: Infer invariants



Approach 2: Infer invariants

```
==========================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
==========================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
==========================================================================
```

Pattern match on deposit
+
Check postconditions if preconditions hold

Two main challenges

- Make monitors useful
  - Weaken preconditions
  - Tighten postconditions

- Avoid false negatives

Challenge – Weaken preconditions

```
================================================================================
transactionsystem.UserAccount.deposit(double)
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
================================================================================
```

Is this deliberate?

Challenge – Weaken preconditions

```
================================================================================
transactionsystem.UserAccount.deposit(double):
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
================================================================================
transactionsystem.UserAccount.deposit(double):
this.opened == orig(this.opened)
this.account_number == orig(this.account_numbe
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
================================================================================
```

Is this deliberate?

Missing test cases?

Challenge – Weaken preconditions

```
==========================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
==========================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
==========================================================================
```

Remove such invariants

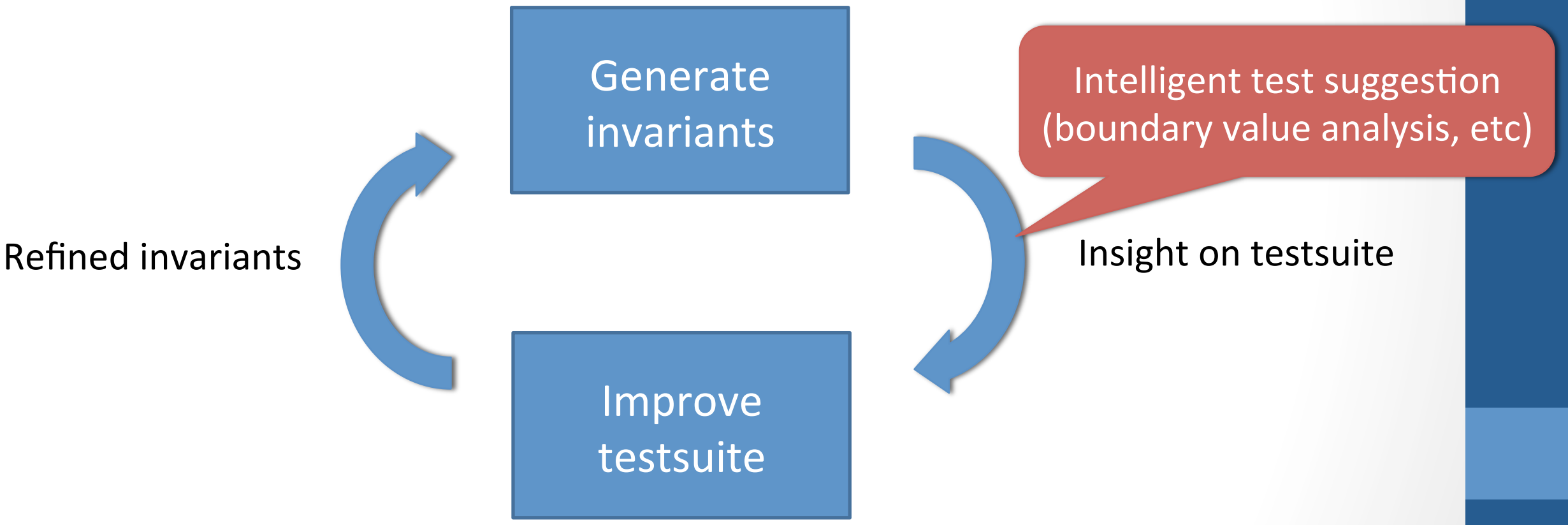
Challenge – Weaken preconditions

```
==========================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
==========================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
==========================================================================
```

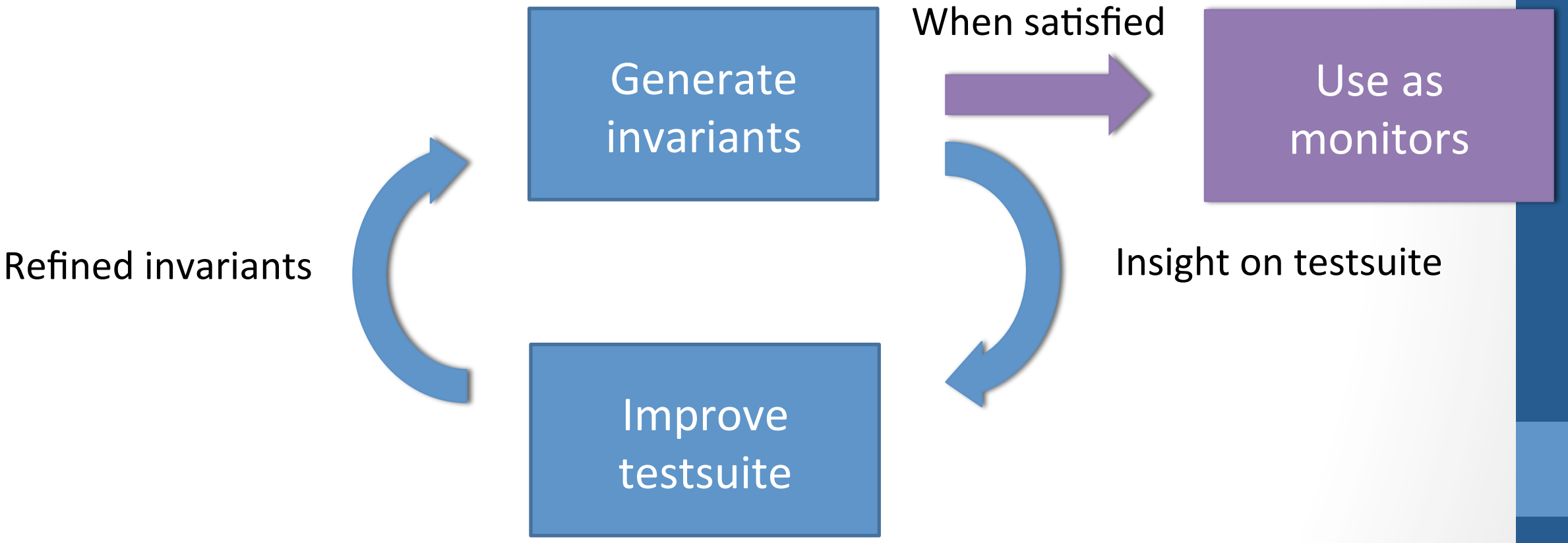
Remove such invariants

Set appropriate threshold

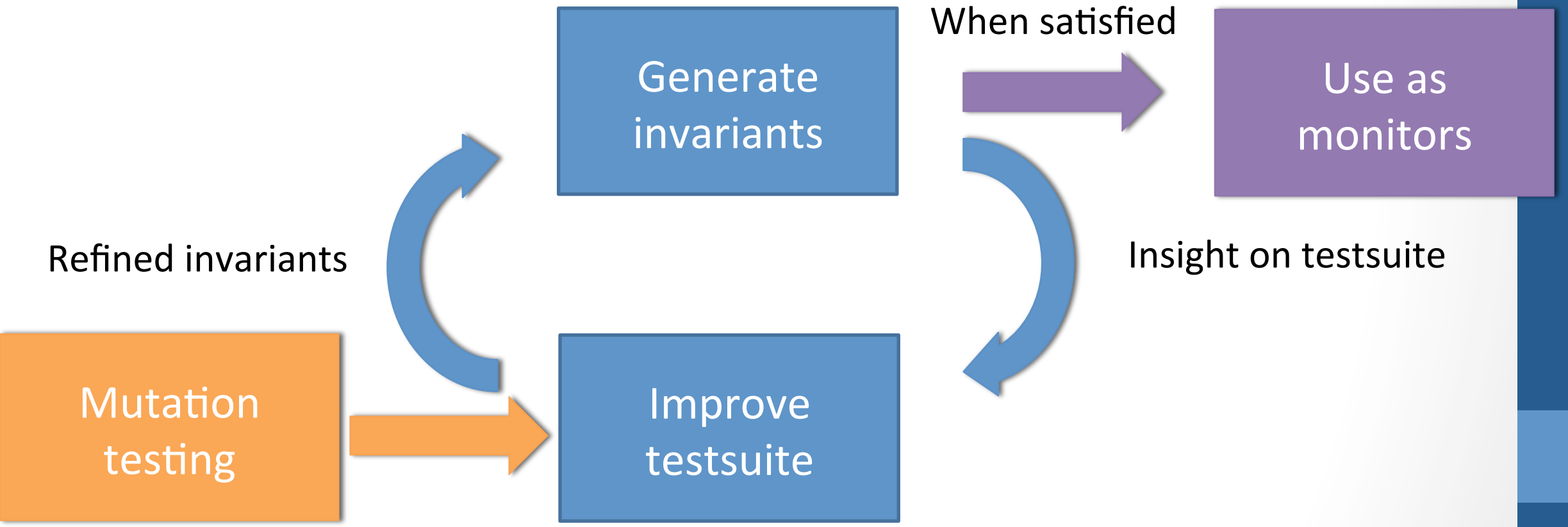
A test case improvement problem



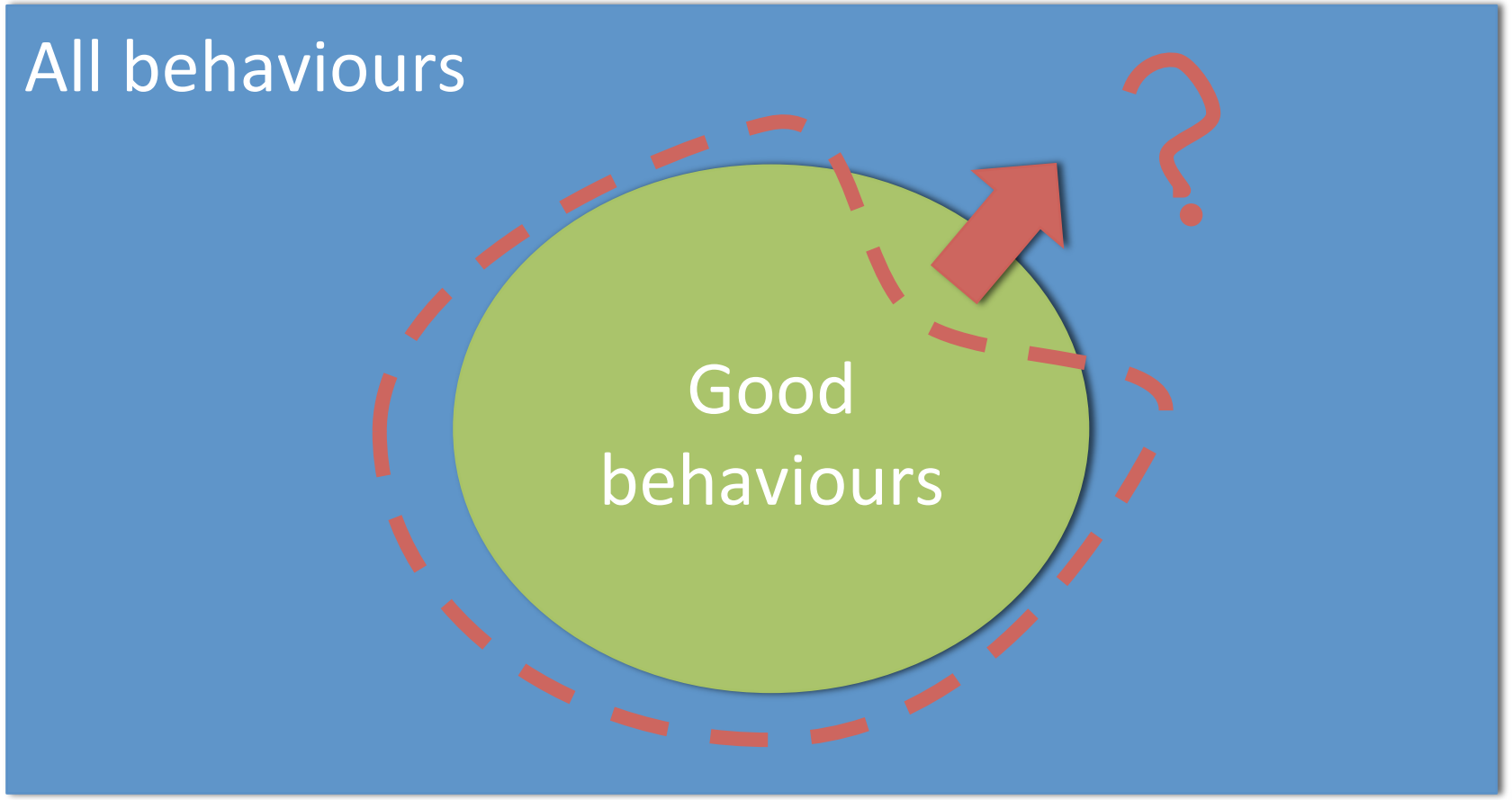
A test case improvement problem



A test case improvement problem



Challenge – Avoiding false negatives



Challenge – Avoiding false negatives

```
==========================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
==========================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_number.toString)
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
==========================================================================
```

amount > orig(this.balance)

Challenge – Avoiding false negatives

```
================================================================================
transactionsystem.UserAccount.deposit(double):::ENTER
this.opened == true
amount one of { 100.0, 500.0, 1000.0 }
================================================================================
transactionsystem.UserAccount.deposit(double):::EXIT
this.opened == orig(this.opened)
this.account_number == orig(this.account_number)
this.owner == orig(this.owner)
this.opened == true
this.account_number.toString == orig(this.account_numbe
this.balance > orig(this.balance)
this.balance >= orig(amount)
this.balance - orig(this.balance) - orig(amount) == 0
                                  ==================
```

amount > orig(this.balance)

Generate test cases which purposefully try to violate the postcondition

Approach 3: Combine testing and RV by design

- Specification of tests and monitors in a single language

  (like property-based testing but allowing some properties to be specified by examples)

  - If a precise specification is available, generate test cases automatically
  - If not, have test cases and specifications specified separately

Approach 3: Combine testin… design

- Specification of tests and monitors in a single la…

  (like property-based testing but allowing so… properties to be specified by examples)

  - If a precise specification is available, generate test cases automatically
  - If not, have test cases and specifications specified separately

E.g., balance'=balance + deposit
Automatically generates
200 = 150 + 50
350 = 290 + 60

E.g., balance' >= 0

Conclusion

- Generating monitors from tests is hard!
- 3 approaches being explored
- Still a lot of questions!