RoboStar Technology
Systematic Software Testing for Robotics

Robert M. Hierons[1] and Raluca Lefticaru[2]

University of Sheffield, University of Bradford, UK

14th November 2019

Thanks: *Ana Cavalcanti, James Baxter, Manuel Núñez, Mercedes Merayo*

ROYAL ACADEMY OF ENGINEERING

EPSRC
Engineering and Physical Sciences Research Council

Overview

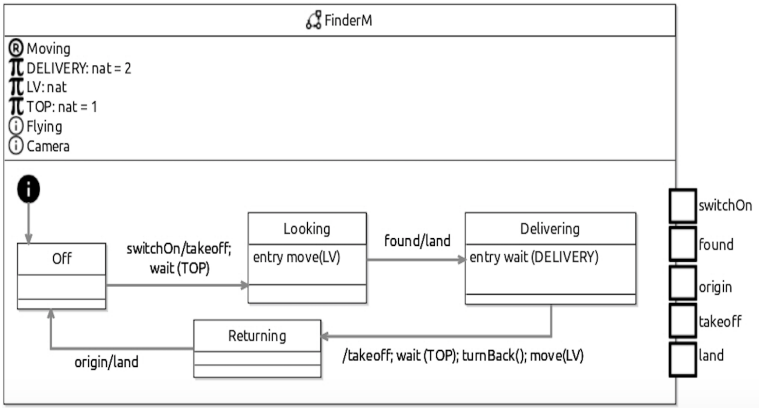
- Motivation
- The Approach
- Mutation with Wodel
- Counterexample generation
- Test Generation

Motivation

Testing

- Aim: generating tests cases from RoboChart models
- Why test?
  - Reality gap
  - Validates the behaviour of the model
  - Explore cases developer may not have considered
- Mutation testing — finds problems arising from variations
- RoboChart testing can be extended to RoboSim testing

Example RoboChart Model



Motivation

Formal Semantics

- Models: state-based, cyclic nature, discrete time
- Semantics: provided by mapping models to a discrete variant of timed CSP – tock CSP
  - CSP models have states
  - there are transitions between states
  - transitions have labels (inputs, outputs, tock - progression of time)
- Aim: automate testing on the basis of this formal semantics.

Motivation

Implementation relations

- Standard minimal test hypothesis: the system under test (SUT) behaves like an unknown model that can be described using the same formalism.
- Define *correctness* of a system under test (SUT) via *implementation relations* between models.
  - Timed trace inclusion
  - Timed refusal trace inclusion
- This enables us to have automated testing that is sound.

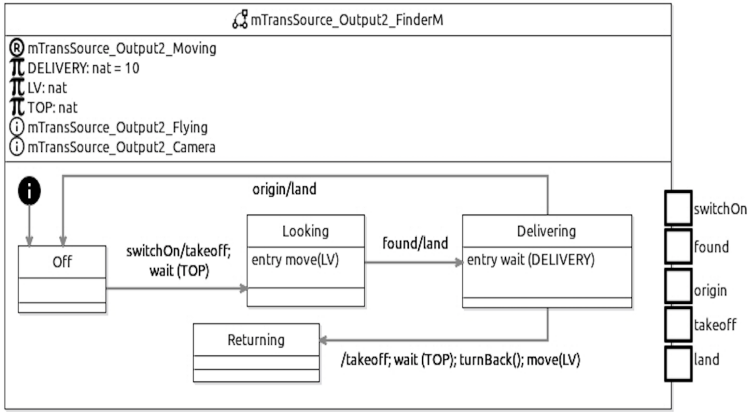
Overview of Approach

Mutation Testing

- Use a mutation testing approach (fault seeding).
- Overall structure:
  - Seed faults into the specification.
  - Each seeded fault defines a mutant.
  - Given a mutant M' of specification M, find a test case that detects (kills) M'.
- We run the resultant test cases on the system under test

Overview of Approach

A mutant of the RoboChart Model

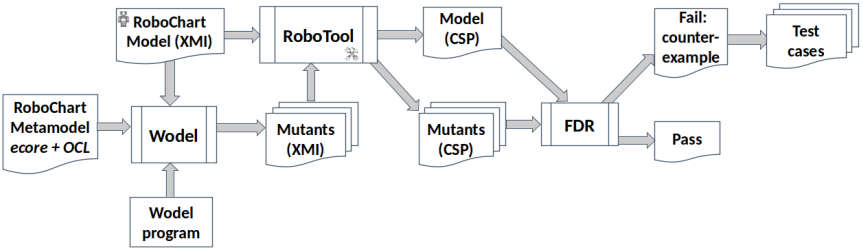


Overview of Approach

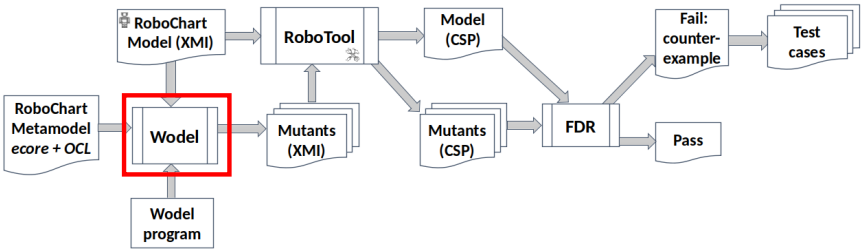
Mutation Testing

- Our mutants are first order mutants: generated by making one change
- If the SUT passes tests that kill the mutants then the SUT cannot be one of these mutants
- We mutate the model, not the CSP generated from the model
- Should more appropriately capture likely faults

Overview of Approach

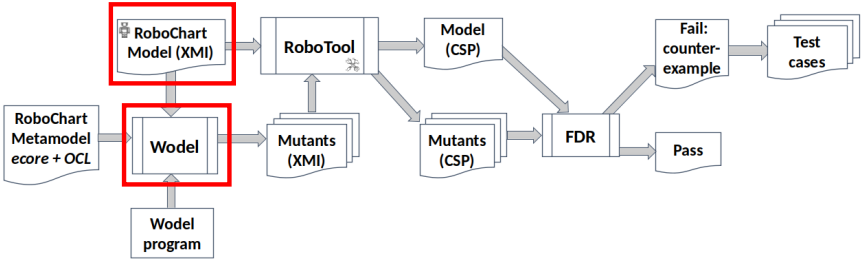


Overview of Approach



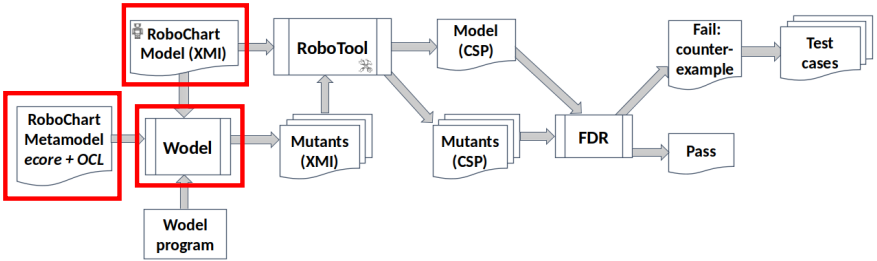
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



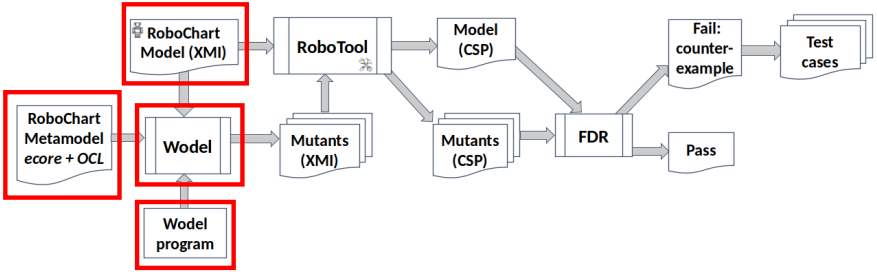
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



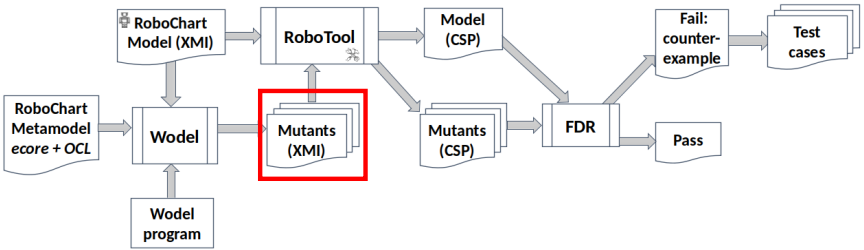
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



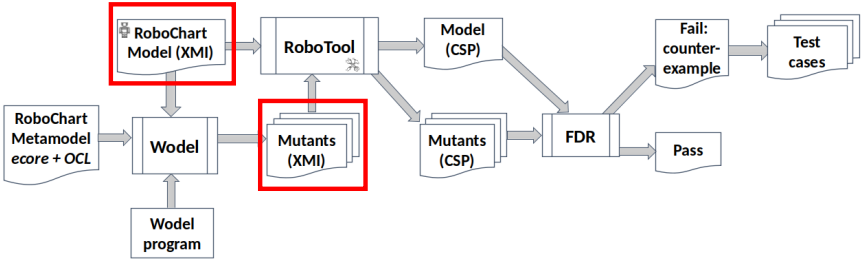
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



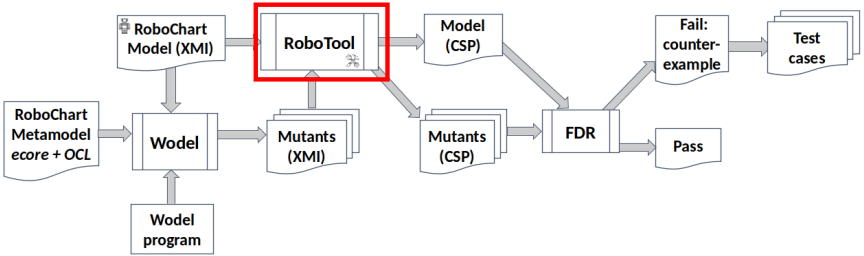
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



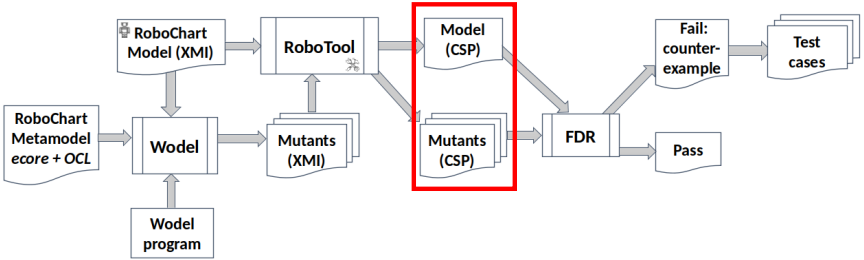
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



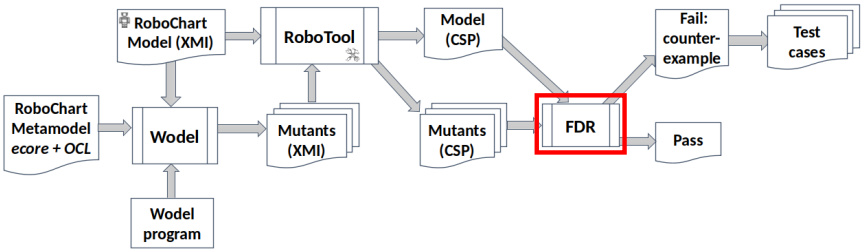
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



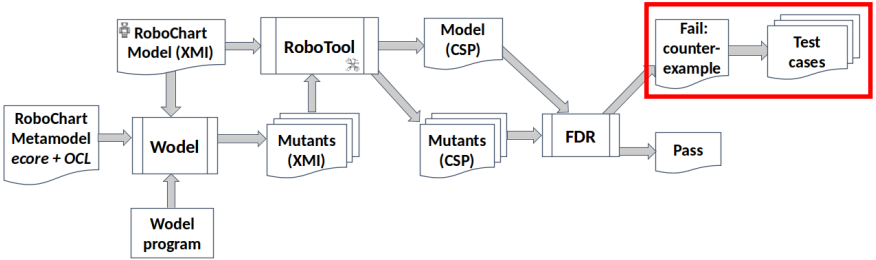
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Overview of Approach



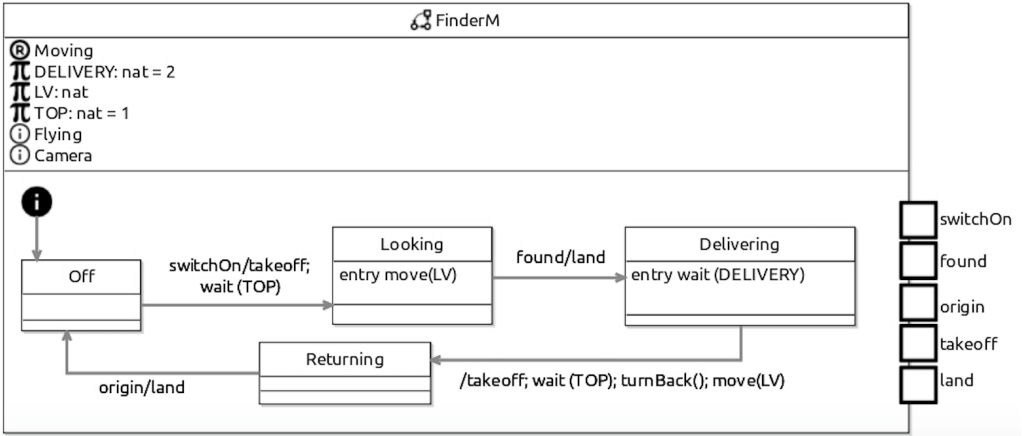
[10] https://www.cs.york.ac.uk/robostar/notations-tools/

Wodel

- Wodel[12] is a tool for generating mutants from provided models using a set of mutation operators.
- Wodel is provided as an Eclipse plugin that interacts with the Eclipse Modelling Framework (EMF).
- Models are instances of a metamodel provided to Wodel.
- The mutation operators are described by Wodel's DSL.

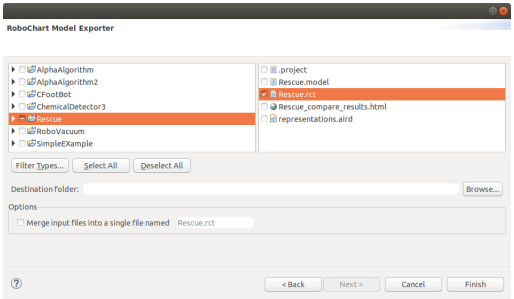
[12] `http://gomezabajo.github.io/Wodel/`

Example RoboChart Model



RoboChart XMI Models

- Wodel acts on models in XMI format
- We thus export RoboChart models from RoboTool to XMI
- Handle multi-file models by merging into one file



Mutation Operators

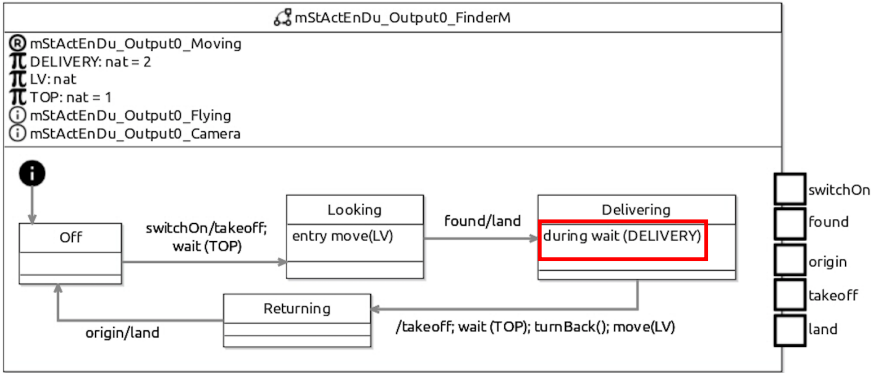
- Used to create the mutants.
- Currently, they:
  - Modify transitions between states; or
  - Delete states
- Wodel applies the mutation operators at random according to their specification.

Mutation Operators – Examples

```
mStActEnDu {
  retype one EntryAction as DuringAction
  // modifies a state by changing an entry action into a during action
}

mTransSource {
  tr = select one Transition where {^source <> one Initial}
  modify target ^source from tr to other State
  // changes the start state of a transition, except the one from the initial junction
}
```

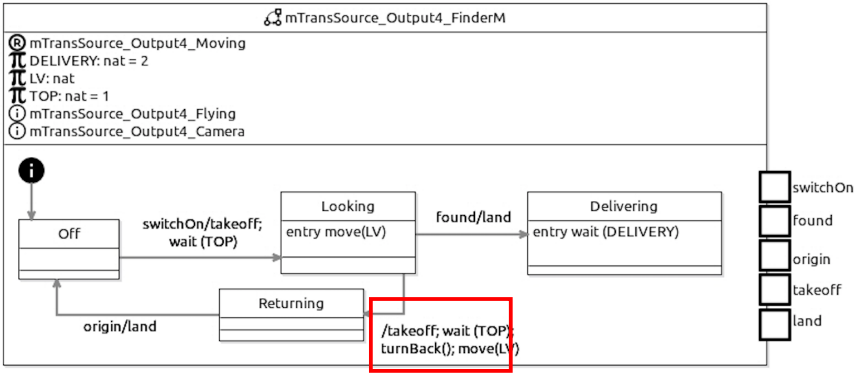
Mutant Example – *mStActEnDu* operator

Delivering state has its *entry* action changed into a *during* action



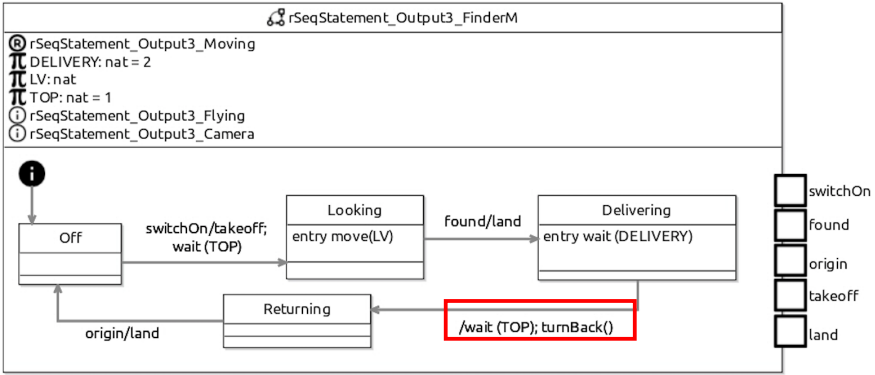
Mutant Example – *mTransSource* operator

The transition source is modified from *Delivering* to *Looking* state



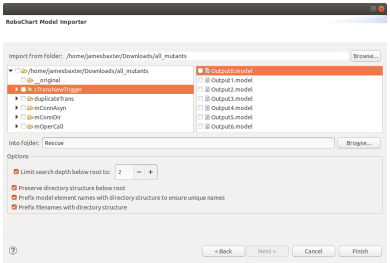
Mutant Example – *rSeqStatement* operator

Two actions are removed from the sequence in the transition from the *Delivering* state



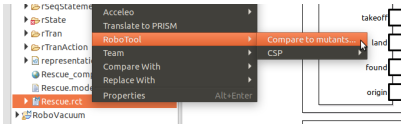
Generating CSP for Mutants

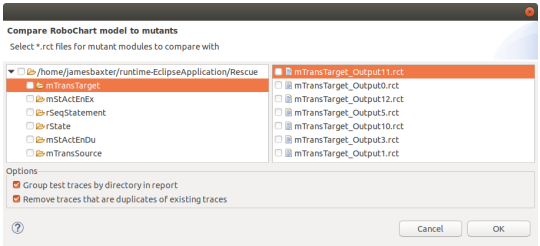
- Wodel outputs mutants in XMI format.
- They are imported into RoboTool.
- RoboTool generates CSP for imported mutants.



Comparing Mutants with FDR

- Check if mutants are correct implementations of original model
- Obtain counterexample traces from those that don't
- Eclipse plugin to handle mutant comparison



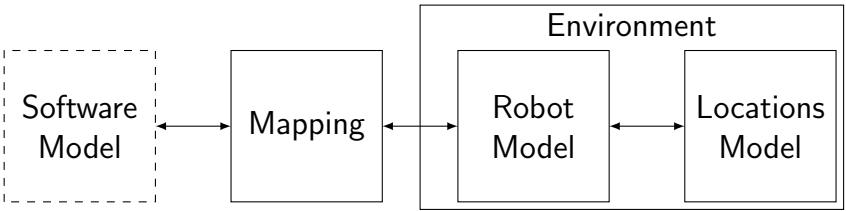


Counterexample Traces

```
Rescue_switchOn.in -> Rescue_takeoff.out -> tock -> moveCall -> moveRet
  -> Rescue_found.in -> Rescue_land.out -> Rescue_origin.in

Rescue_switchOn.in -> Rescue_takeoff.out -> tock -> tock

Rescue_switchOn.in -> Rescue_takeoff.out -> tock -> moveCall -> moveRet
  -> Rescue_found.in -> Rescue_land.out -> moveCall

Rescue_switchOn.in -> Rescue_takeoff.out -> tock -> moveCall -> moveRet
  -> Rescue_found.in -> Rescue_land.out -> Rescue_takeoff.out

Rescue_switchOn.in -> Rescue_takeoff.out -> tock -> moveCall -> moveRet
  -> Rescue_origin.in
```

Considerations about the Environment

- Some generated traces are infeasible — they don't represent cases that could occur.
- e.g. finding the target immediately after starting movement
- Such traces aren't useful, since they don't relate to the robot's environment.
- To eliminate these traces, we need to model the environment.



Test Generation

- From each counterexample trace, we generate a test T.
- T performs events of the system-under-test, SUT, and reports verdict using events *pass*, *fail* and *inc* (inconclusive).
- The test corresponding to a trace $a_1, \ldots, a_n$ is:

$$T \mathrel{\widehat{=}} inc \rightarrow a_1 \rightarrow \cdots \rightarrow inc \rightarrow a_{n-1} \rightarrow pass \rightarrow a_n \rightarrow fail \rightarrow Stop$$

- Test execution is represented by a parallel composition:

$$(SUT \; [\![\alpha SUT]\!] \; T) \setminus \alpha SUT$$

- The last event observed determines the result of the test.

Future Work

- So far we have just considered traces - we can also test for refusals at the end of a trace (or within a trace).
- Our tests are sequences - we could combine these to form trees
- Probabilities and continuous variables

Conclusion

- Generate tests from RoboChart by mutation using Wodel
- Comparison of mutants and original using FDR
- Counterexample traces generate sound tests
- Future work to allow testing for refusals