Using Machine Learning Techniques to Identify Dependent Failures

Australian System Safety Conference 2024

Ben Merange, RGB Assurance
Lucas Schuurmans-Stekhoven, The University of Queensland

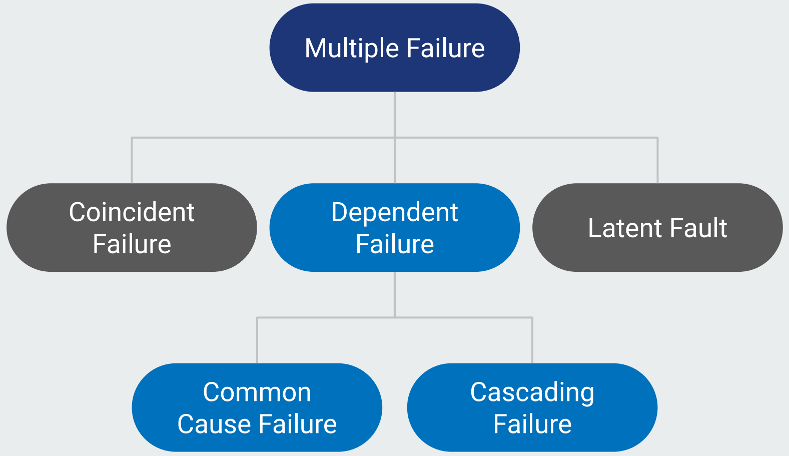
Overview

1. Review of dependent failures
2. Common analysis techniques
3. Building upon these techniques
4. An example application
5. Proof of concept

How can we use disruptive technologies to build upon existing assurance techniques?

Image by Pete Linforth from Pixabay.

Dependent failures



Dependent failure:
Where there is a causal relationship between multiple failures.

Common cause failure:
Where two or more failures resulted from a single cause, either simultaneously or separated in time.

E.g., an earthquake causes multiple subsystem failures.

Cascading failure:
Where one failure causes another.

E.g., a power regulator fails, causing downstream components to receive the wrong voltage and fail.

Common techniques

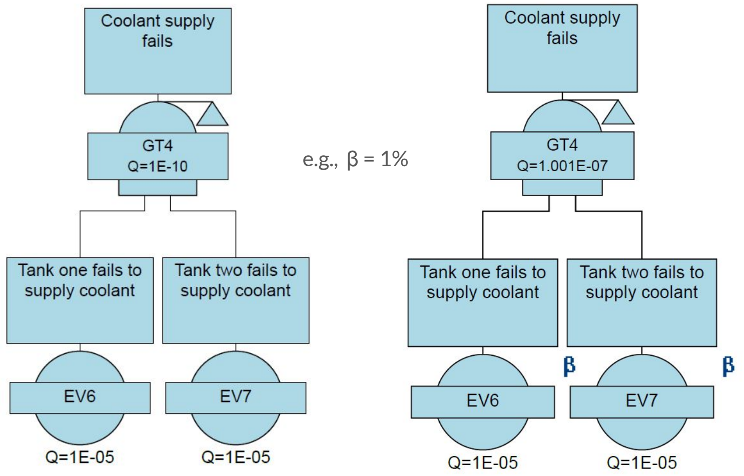
Implicit modelling:

- Does not attempt to identify or eliminate causes of dependent failures
- Assumes that dependent failures can occur, and attempts to model their effects conservatively
- Often captured in a fault tree (e.g., via beta factors, alpha factors, or shock model)

Explicit analysis:

- Attempts to identify and eliminate causes of dependent failures
- Relies heavily on experience, historical data, checklists, and guide words
- Often a top-down analysis, considering different characteristics of the system
- Rigorous bottom-up analysis is generally not feasible for complex systems

Implicit modelling



Coolant supply fails
GT4
Q=1E-10
Tank one fails to supply coolant
Tank two fails to supply coolant
EV6
EV7
Q=1E-05
Q=1E-05
e.g., β = 1%
Coolant supply fails
GT4
Q=1.001E-07
Tank one fails to supply coolant
Tank two fails to supply coolant
β
β
EV6
EV7
Q=1E-05
Q=1E-05

Explicit analysis

Consider common:

- Designs
- Components
- Energy sources
- Data sources
- Requirements
- Manufacturers
- Location and environment
- Procedures, policies, or staff
- …

Bottom-up analysis is “too hard”

It’s hard to do efficiently.

“Nowadays, even simulation is used widely to automate the work of producing an FMEA report, it is still not feasible and very time-consuming in engineering design considering all possible combinations of failures, especially for some very large and complicated systems.”

(Xiao et al., 2011).

It’s hard to produce meaningful results.

“Consideration of all possible combinations of failures is impractical for all but the simplest example systems. Even if the task of producing the FMEA report for the full multiple failure scenario were automated, it would still be impractical for the engineers to read, understand, and act on all of the results.”

(Price & Taylor, 2002)

Building upon these techniques

Thinking like a “big data” analyst

- High-Throughput Screening (HTS) is used in the biomedical industry for drug discovery, rapidly testing chemical interactions
- Combinatorial testing is used in software to rapidly test interactions between inputs
- Design of Experiments (DoE) screening designs are used in manufacturing to identify the interactions between inputs
- Machine learning techniques can identify relationships between variables

The proposed approach

Assumptions:

- Access to a simulation of the system
- Knowledge of the failure modes of each component
- That traditional techniques have been applied

The process:

1. Build an efficient test plan
2. Simulate many combinations of component failures, looking for system failure
3. Process the data to identify correlations between component failures and system failure
4. Output identified failure mode combinations
5. Review each result, looking for causal relationships between the failure modes

Coming from both directions

Top-down:

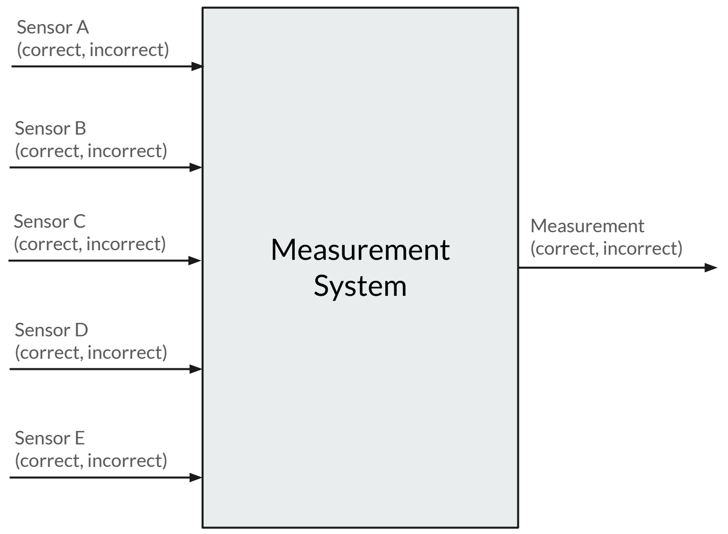
Work through possible dependent failure causes, looking for credible failure mode combinations that could cause the system to fail.

Bottom-up:

Work through failure mode combinations that cause the system to fail, looking for credible causes.

Example application

A simple system,
with simple failure modes,
that can be manually analysed



Single failure scenarios

#	Sensor A	Sensor B	Sensor C	Sensor D	Sensor E	Result
1	×	✓	✓	✓	✓	✓
2	✓	×	✓	✓	✓	✓
3	✓	✓	×	✓	✓	✓
4	✓	✓	✓	×	✓	✓
5	✓	✓	✓	✓	×	✓

Multiple failure scenarios (brute force)

#	Sensor A	Sensor B	Sensor C	Sensor D	Sensor E
1	×	×	✓	✓	✓
2	×	✓	×	✓	✓
3	×	✓	✓	×	✓
4	×	✓	✓	✓	×
			…		
32	✓	✓	✓	×	×

Multiple failure scenarios (covering array)

#	Sensor A	Sensor B	Sensor C	Sensor D	Sensor E
1	✓	✓	✓	✓	✓
2	✓	×	×	×	×
3	×	✓	×	✓	×
4	×	×	✓	×	✓
5	✓	✓	✓	×	×
6	×	×	×	✓	✓

Generated using the FIPOG algorithm implemented by CAgen.

Processing the data

#	Sensor A	Sensor B	Sensor C	Sensor D	Sensor E	Result
1	✓	✓	✓	✓	✓	✓
2	✓	×	×	×	×	×
3	×	✓	×	✓	×	✓
4	×	×	✓	×	✓	×
5	✓	✓	✓	×	×	✓
6	×	×	×	✓	✓	✓

Covering arrays

Scenario selection:

- Many established methods
  - Plackett-Burman designs
  - Fractional factorial designs
  - All-pairs testing
  - Combinatorial testing
  - …
- Several existing tools
  - Advanced Combinatorial Testing System
  - Pairwise Independent Combinatorial Tool
  - Covering Array Generator
  - …

Data processing options:

- Single failures
  - Linear regression
  - …
- Two failures
  - Linear regression with interaction terms
  - Neural networks
  - …
- Many failures
  - Neural networks
  - Decision trees
  - …

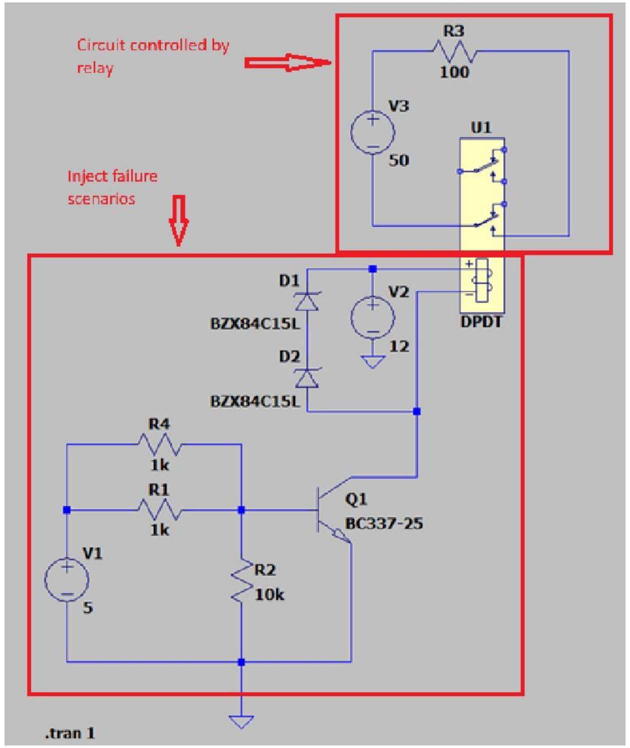
Proof of concept

Lucas’ Internship Project at RGB Assurance

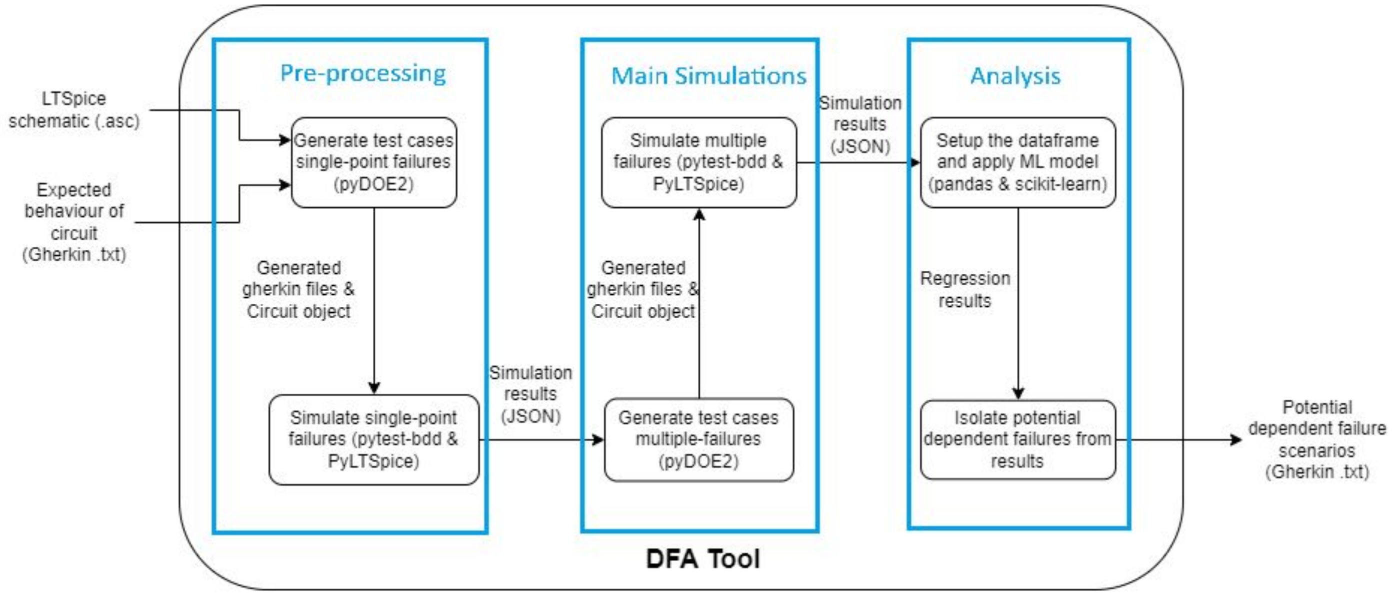
- A program written in Python that incorporates ML to identify potential Dependent Failure scenarios at the component level for electrical circuits
- Utilises DoE methods to cut-down on the total number of test cases needed to obtain meaningful results
- Can simulate and make modifications to a users input circuit
- Uses an automation framework that streamlines simulations
- Users interact with the program using plain English
- The purpose of the program is to support existing methods of Dependent Failure Analysis - not replace it

Example circuit

- The Relay Switch Circuit is typically used to control high voltage instruments (motors, heaters, lamps)
- Has many safety related use cases such as Emergency Stop Pushbuttons
- Potential dependent failures occur if: R1 and R4 open circuit, D1 and D2 short circuit (For a high input signal)
- No potential dependent failures evident for a low input signal only single points
- Voltage source failures are trivial and are not included in the analysis

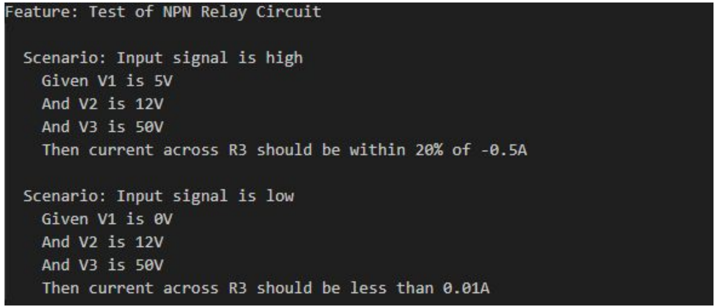


Software overview

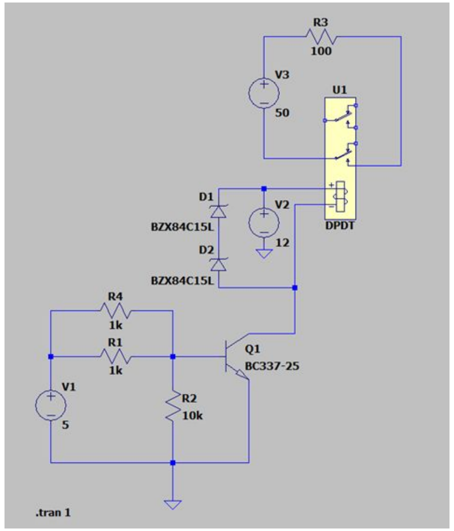

LTSpice schematic (.asc)
Expected behaviour of circuit (Gherkin .txt)
Pre-processing
Generate test cases single-point failures (pyDOE2)
Generated gherkin files & Circuit object
Simulate single-point failures (pytest-bdd & PyLTSpice)
Simulation results (JSON)
Main Simulations
Generate test cases multiple-failures (pyDOE2)
Generated gherkin files & Circuit object
Simulate multiple failures (pytest-bdd & PyLTSpice)
Simulation results (JSON)
Analysis
Setup the dataframe and apply ML model (pandas & scikit-learn)
Regression results
Isolate potential dependent failures from results
Potential dependent failure scenarios (Gherkin .txt)
DFA Tool

Inputs

- A text file detailing high-level system requirements written in Gherkin and a LTSpice (.asc) schematic file
- Gherkin follows a "Given", "When", "Then" syntax
- Allows a user to change or add scenarios that test system requirements
- Works hand-in-hand with the selected automation framework

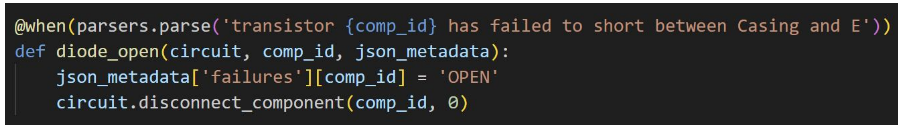


```
Feature: Test of NPN Relay Circuit

  Scenario: Input signal is high
    Given V1 is 5V
    And V2 is 12V
    And V3 is 50V
    Then current across R3 should be within 20% of -0.5A

  Scenario: Input signal is low
    Given V1 is 0V
    And V2 is 12V
    And V3 is 50V
    Then current across R3 should be less than 0.01A
```

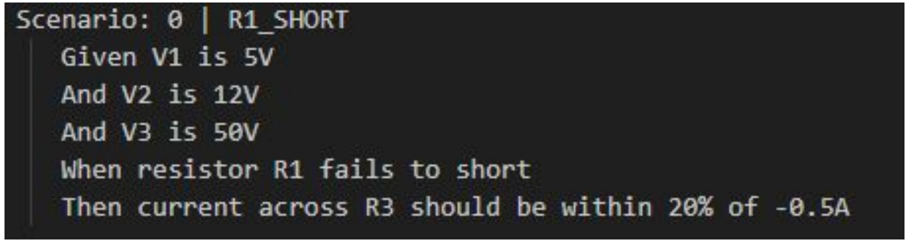


Pre-processing

- Generate test-cases for single-point failures and create gherkin file
- Pytest-bdd is used here to execute necessary steps for simulations
- Single-point failures that result in system failure are recorded and excluded from further analysis



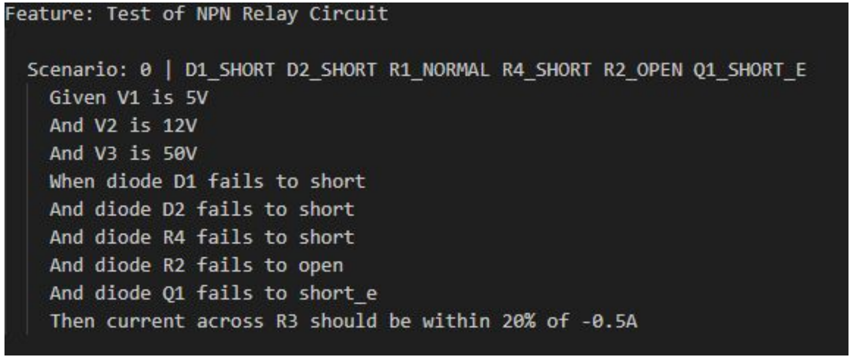
```
@when(parsers.parse('transistor {comp_id} has failed to short between Casing and E'))
def diode_open(circuit, comp_id, json_metadata):
    json_metadata['failures'][comp_id] = 'OPEN'
    circuit.disconnect_component(comp_id, 0)
```



```
Scenario: 0 | R1_SHORT
    Given V1 is 5V
    And V2 is 12V
    And V3 is 50V
    When resistor R1 fails to short
    Then current across R3 should be within 20% of -0.5A
```

Main Testing

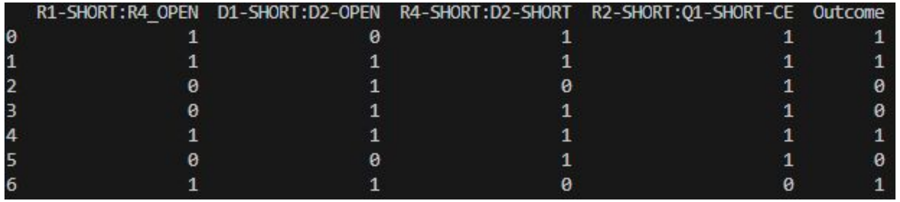
- pyDOE2 library is used to generate multiple-failure test cases
- Fractional factorial was the selected test design
- Again, pytest-bdd is employed to automate the circuit simulations



```
Feature: Test of NPN Relay Circuit

  Scenario: 0 | D1_SHORT D2_SHORT R1_NORMAL R4_SHORT R2_OPEN Q1_SHORT_E
    Given V1 is 5V
    And V2 is 12V
    And V3 is 50V
    When diode D1 fails to short
    And diode D2 fails to short
    And diode R4 fails to short
    And diode R2 fails to open
    And diode Q1 fails to short_e
    Then current across R3 should be within 20% of -0.5A
```

Analysis

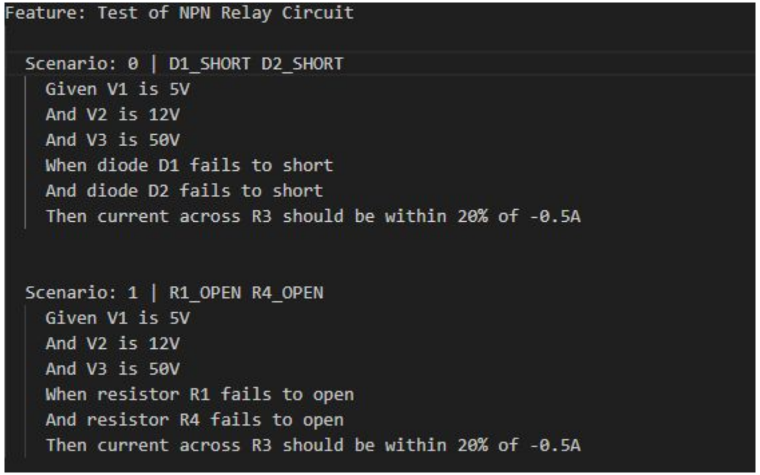
- Every combination of 2-way failures are set as features
- ‘1’ represents the combination was present in the simulation, ‘0’ otherwise
- Logistic Regression is applied to the main simulation results
- Likelihood that a combination may contribute to a system failure is given to each feature
- Gherkin file is generated with high likelihood combinations



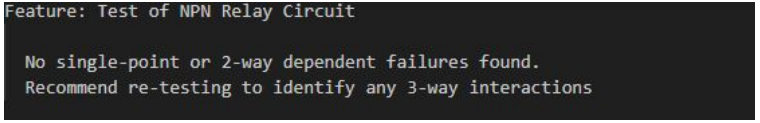
```
   R1-SHORT:R4_OPEN  D1-SHORT:D2-OPEN  R4-SHORT:D2-SHORT  R2-SHORT:Q1-SHORT-CE  Outcome
0                 1                 0                  1                     1        1
1                 1                 1                  1                     1        1
2                 0                 1                  0                     1        0
3                 0                 1                  1                     1        0
4                 1                 1                  1                     1        1
5                 0                 0                  1                     1        0
6                 1                 1                  0                     0        1
```

Outputs

- The output is 2 Gherkin files detailing any single point failures and most likely dependent failure scenarios
- If no dependent failure scenarios were found, it will suggest to try the next highest order of interactions (i.e. 2-way vs 3-way)



```
Feature: Test of NPN Relay Circuit

  Scenario: 0 | D1_SHORT D2_SHORT
    Given V1 is 5V
    And V2 is 12V
    And V3 is 50V
    When diode D1 fails to short
    And diode D2 fails to short
    Then current across R3 should be within 20% of -0.5A


  Scenario: 1 | R1_OPEN R4_OPEN
    Given V1 is 5V
    And V2 is 12V
    And V3 is 50V
    When resistor R1 fails to open
    And resistor R4 fails to open
    Then current across R3 should be within 20% of -0.5A
```



```
Feature: Test of NPN Relay Circuit

  No single-point or 2-way dependent failures found.
  Recommend re-testing to identify any 3-way interactions
```

How did it perform?

Outcome:

- Was able to detect potential dependent failures for simple circuits
- Struggled with larger, more complex circuits
- Logistic regression was the most accurate

Future work:

- Improve performance
- Explore different regression and classification methods
- Explore different system types (alternative simulation methods)

This is just one example

Can we use disruptive technologies to build upon other existing assurance techniques?

In this presentation, we:

- Revised dependent failures
- Reviewed common analysis techniques
- Proposed a method to build upon these techniques
- Presented a proof of concept