How Can We Provide Assured Autonomy?

Rick Kuhn
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
kuhn@nist.gov
What is the problem, and why is it hard?

Autonomous vehicles must be **safe and secure**

Current testing and assurance methods are not suitable

Security flaws are generally a small subset of s/w flaws

*Remember*: a vulnerability is usually just a bug that can be used to use to defeat security properties
Outline

• Why current critical system testing processes are not suitable
• Assurance based on input space coverage,
• Transfer learning

Some problems in assured autonomy, and potential solutions
What is NIST and why are we doing this?

- US Government agency, which supports US industry through developing better measurement and test methods
- 3,000 scientists, engineers, and staff including 4 Nobel laureates
- Broad involvement with industry and academia
What are interaction faults?

- NIST studied software failures in 15 years of FDA medical device recall data
- What causes software failures?

**Interaction faults:** e.g., failure occurs if
  - pressure < 10 & volume > 300
  (interaction between 2 factors)

So this is a **2-way interaction**

=> testing all pairs of values can find this fault
How are interaction faults distributed?

• Interactions e.g., failure occurs if
  - pressure < 10 (1-way interaction)
  - pressure < 10 & volume > 300 (2-way interaction)
  - pressure < 10 & volume > 300 & velocity = 5 (3-way interaction)

• Surprisingly, no one had looked at interactions > 2-way before

![Graph showing percentage of faults caused by single and multiple factors.]
Various domains collected

Wide variation in percent of failures caused by single factor

Variability decreases as number of factors increases

More testing or users => harder to find errors, fewer single factor failures

- Number of factors involved in failures is small
- No failure involving more than 6 variables has been seen
How is all this related to autonomous systems?
DSB 2012 The Role of Autonomy in DoD Systems Study recommends:

“USD(AT&L) to create developmental and operational T&E techniques that focus on the unique challenges of autonomy (to include developing operational training techniques that explicitly build trust in autonomous systems).”

**Recommendation:**
USD(AT&L) establish developmental and operational T&E techniques that focus on the unique challenges of autonomy

- Coping with the difficulty of enumerating all conditions and non-deterministic responses
- Basis for system decisions often not apparent to user
- Measuring trust that the autonomous system will interact with its human supervisor as intended
- Leverage the benefits of robust simulation
Software assurance is already very expensive

Consumer level software cost: about 50% code development, 50% testing and verification

For aviation life-critical, 12% code development, 88% testing and verification

Autonomy makes the problem even harder!
Why can’t we use same processes as other high assurance software?

• Conventional critical software testing is based on structural coverage – ensuring that conditions, decisions, paths are covered in testing.

• Life-critical aviation software requires MCDC testing, white-box criterion that doesn’t fit neural nets and other black-box methods where input is what matters.
Code coverage works well - for conventional software

Test coverage has traditionally been defined using graph-based structural coverage criteria:
- statement (weak)
- branch (better)
- etc.

Based on paths through the code

We may have perfect structural coverage of code, but what does that tell us about response to rare inputs?

What if the code is always the same, and only the inputs matter?
Can we use code coverage for machine learning?

• Much of AI/ML depends on various neural nets
• Algorithm and code stays the same
• Connections and weights vary
• Behavior changes depending on inputs used in training
Input space coverage is needed

• Gold standard of assurance and verification of life-critical software is not suitable for much of new life-critical autonomy software

• We can measure “neuron coverage”, but indirect measure and not clear how closely related to accuracy and ability to correctly process all of the input space

• Measure the input space directly

• Then see if the AI system handles all of it correctly
Outline

• Why current critical system testing processes are not suitable
• Assurance based on input space coverage
• Transfer learning
Scientists have trained rats to drive tiny cars to collect food.

But can they do it under all kinds of conditions?

It doesn’t take much intelligence to drive a car. Even rats can do it!

The problem is harder outside of a constrained environment.
Things get tricky as the scene becomes complex

• Multiple conditions involved in accidents
  • "sensors failed to pick up street signs, lane markings, and even pedestrians due to the angle of the car shifting in rain and the direction of the sun" (3 factors)
  • “... the white side of the tractor trailer against a brightly lit sky, so the brake was not applied. The high ride height of the trailer combined with its positioning across the road ...” (4 factors)

• We need to understand what combinations of conditions are included in testing
How can we measure interaction fault detection capability?

19 combinations included in test set


<table>
<thead>
<tr>
<th>Vars</th>
<th>Combination values</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>a c</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>a d</td>
<td>00, 01, 11</td>
<td>.75</td>
</tr>
<tr>
<td>b c</td>
<td>00, 11</td>
<td>.50</td>
</tr>
<tr>
<td>b d</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
<tr>
<td>c d</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
</tbody>
</table>
### Rearranging the Table:

<table>
<thead>
<tr>
<th>Vars</th>
<th>Combination values</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>a c</td>
<td>00, 01, 10</td>
<td>.75</td>
</tr>
<tr>
<td>a d</td>
<td>00, 01, 11</td>
<td>.75</td>
</tr>
<tr>
<td>b c</td>
<td>00, 11</td>
<td>.50</td>
</tr>
<tr>
<td>b d</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
<tr>
<td>c d</td>
<td>00, 01, 10, 11</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Total possible 2-way combinations

\[ = 2^2 \binom{4}{2} = 24 \]

\[ S_2 = \text{fraction of 2-way combinations covered} = \frac{19}{24} = 0.79 \]
### Graphing Coverage Measurement

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Combinations Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>00 00</td>
</tr>
<tr>
<td>0.75</td>
<td>01 01 00 00 00</td>
</tr>
<tr>
<td>0.50</td>
<td>10 10 01 01 01 00</td>
</tr>
<tr>
<td>0.25</td>
<td>11 11 10 10 11 11</td>
</tr>
<tr>
<td><strong>bd cd</strong></td>
<td><strong>ab ac ad bc</strong></td>
</tr>
</tbody>
</table>

**Bottom line:** All combinations covered to at least .50

100% coverage of .33 of combinations
75% coverage of .50 of combinations
50% coverage of .16 of combinations

\[ S_2 = \text{area under curve} \]
\[ = 0.79 \]
What else does this chart show?

$S_t = \text{combinations in training and testing} \Rightarrow \text{model works for these}$

$1 - S_t = \text{Missing combinations}$

(look for problems here)
How much combinatorial coverage is achieved with conventional tests?

Spacecraft software example
- 82 variables,
- 7,489 tests,
- conventional test design
Outline

• Why current critical system testing processes are not suitable
• Assurance based on input space coverage
• Transfer learning – example application
Transfer learning – what is the problem?

- Differences inevitably exist between training data sets, test data sets, and later real-world data.
- Further differences exist between data from two or more different environments.
- How do we predict performance of a model trained on one data set when applied to another?
  - New environment
  - Changed environment
  - Additional possible values, etc.
Transfer learning – conventional practice

Randomized selection – but how much random data will be sufficient, especially with smaller data sets?

Ensure at least one of each object type – but this may not be representative of object attribute distributions

Interactions are critical to consider in most ML problems, especially for safety, but conventional practice does little to ensure data sets are adequately representative of interactions
Example – image analysis

- Planes in satellite imagery – Kaggle ML data set – determine if image contains or does not contain an airplane

- Two data sets – Southern California (SoCal, 21,151 images) or Northern California (NorCal, 10,849 images)

- 12 features, each discretized into 3 equal range bins
Transfer learning problem

- Train model on one set, apply to the other set
- Problem –
  - Model trained on larger, SoCal data applied to smaller, NorCal data → performance drop
  - Model trained on smaller, NorCal data applied to larger, SoCal data → NO performance drop
- This seems backwards!
- Isn’t it better to have more data?
- Can we measure, explain and predict it next time?
Density of combinations in one versus the other data set, 2-way

For $C = \text{SoCal}$, $N = \text{NorCal}$,

$\frac{|C\setminus N|}{|C|} = 0.02$

$\frac{|N\setminus C|}{|N|} = 0.12$

The NorCal data set has fewer “never seen” combinations, even with half as many observations.
Summary – Transfer learning

• Current approaches to estimating success for transfer learning are largely ad-hoc and not highly effective

• Combinatorial methods show promise for improvements – measurable quantities directly related to determining if one data set is representative of the field of application

• Much additional work is needed to evaluate this idea, and to understand the link between combinatorial difference values and prediction accuracy

• Empirical studies planned
Assured autonomy – more questions than answers

• Interactions of learning components with programmed components – especially replacing humans

• Changes the nature of system failures

• More like failures involving human factors issues?

😊 Turing test for bugs! Distinguish between human-triggered and AI-triggered system failures?
Assured autonomy – key points & current state

• For capability and cost reasons, autonomous components are becoming routine in software engineering

• Many, or most, methods used in high assurance conventional systems are not sufficient for many autonomous components
  • Structural coverage – not for neural nets, and others
  • Formal proofs – for some parts but limited

• How to deal with learning, dynamic changes in system?
• Understanding and measuring interaction coverage is necessary
Learning and Applying Combinatorial Methods

• Self-contained tutorial on using combinatorial testing for real-world software
• Advanced topics such as the use of formal models and test oracle generation
• Costs and practical considerations
• Designed for testers or undergraduate students of computer science or engineering
Automated Combinatorial Testing for Software (ACTS)

Overview

Combinatorial methods can reduce costs for software testing, and have significant applications in software engineering:

- **Combinatorial or t-way testing** is a proven method for more effective testing at lower cost. The key insight underlying its effectiveness resulted from a series of studies by NIST from 1999 to 2004. NIST research showed that most software bugs and failures are caused by one or two parameters, with progressively fewer by three or more, which means that combinatorial testing can provide more efficient fault detection than conventional methods. Multiple studies have shown fault detection equal to exhaustive testing with a 20X to 700X reduction in test set size. New algorithms compressing combinations into a small number of tests have made this method practical for industrial use, providing better testing at lower cost. See articles on [high assurance software testing](http://csrc.nist.gov/acts) or [security and reliability](http://csrc.nist.gov/acts).

- **Autonomous systems assurance**: Input-space coverage measurements are needed in life-critical assurance and verification of autonomous systems, because current methods for assurance of safety critical systems rely on measures of structural coverage, which do not apply to many autonomous systems. Combinatorial methods, including a theorem relating measures of input space coverage, offer a better approach for autonomous system verification.

- **Metrology** for software engineering. Sound engineering requires accurate measurement and analysis. Structural coverage enables formally defined criteria for test completeness, but even full coverage may miss faults related to rare inputs. Combinatorial methods open new possibilities for metrology in software engineering, providing a more scientific approach to assurance and verification.

*Metrology is the science of measurement (NIST is the US national metrology institute).
Freely Available Tools

• **Covering array generator** – basic tool for test input or configurations;

• **Combinatorial coverage measurement** – detailed analysis of combination coverage; automated generation of supplemental tests; helpful for integrating c/t with existing test methods

• **Sequence covering array generator** – new concept; applies combinatorial methods to event sequence testing

• **Input modeling tool** – design inputs to covering array generator using classification tree editor; useful for partitioning input variable values

• **Fault location tool** – identify combinations and sections of code likely to cause problem

ICCS 2024
Please contact us if you’re interested!

Rick Kuhn, Raghu Kacker, M.S. Raunak
{kuhn, raghu.kacker, raunak}@nist.gov

http://csrc.nist.gov/acts