Preprint: Kuhn, D. R., Raunak, M. S., Kacker, R. N., Chandrasekaran, J., Lanus, E., Cody, T., & Freeman, L. (2024). Assured Autonomy Through Combinatorial Methods. Computer, 57(5), 86-90.

Assured Autonomy through Combinatorial Methods

D. Richard Kuhn, National Institute of Standards and Technology
M S Raunak, National Institute of Standards and Technology
Raghu N. Kacker, National Institute of Standards and Technology
Jaganmohan Chandrasekaran, Virginia Tech National Security Institute
Erin Lanus, Virginia Tech National Security Institute
Tyler Cody, Virginia Tech National Security Institute
Laura Freeman, Virginia Tech National Security Institute

Abstract: Autonomous systems are proliferating rapidly, with strong interest in everything from vacuum cleaners and lawnmowers to self-driving cars and autonomous farm equipment. Can these systems be trusted to function safely? Many conventional software engineering methods for high-trust software are not well suited to assured autonomy, but concepts from combinatorial testing can add confidence by providing a quantitative measure of the usefulness of a data set.

Keywords: autonomy, combinatorial testing, machine learning, software, trust

Software assurance [1] is already very expensive. While for consumer-level software, the split between code development and verification/test is often estimated to be roughly 50/50 [2], the situation is understandably very different for safety-critical systems. For example, the extraordinary safety record of civil aviation is due to an intense safety culture throughout the field, and software is no exception. In contrast to consumer software, aviation life-critical software has a roughly 1:7 split between code development and verification/testing [3], i.e., about 90 % of software cost is for assurance.

Despite the cost, we clearly have processes that can produce ultra-high levels of trust in software. But these processes have been developed for conventional software. How will the situation change with the incorporation of greater autonomy and machine learning?

As one example, a RAND Corp. study considered the time and cost to establish 95 % confidence that autonomous vehicle failure rates are 20 % better than human drivers, using a fleet of 100 vehicles driving 24 hours a day, 365 days a year, at an average speed of 25 miles per hour [4]. For assumed real-world environments, an estimated 11 billion miles of driving, requiring 500 years, would be needed for confidence that the systems behaved correctly across the huge range of conditions that would be encountered in actual driving.

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

But can they do it under all kinds of conditions ?

The problem is harder outside of a constrained environment

Early experiments in autonomous driving illustrate some of the extreme challenges in ensuring safety in these systems. In one case, a system that performed correctly in dry conditions had repeated problems in rain. The cause was not the slippery road condition, but "the real problem is that depending on the *angle* between the car, *wet surfaces*, and the *sun*, the car's cameras can have a difficult time recognizing all kinds of things, including lane markings and street signs." [5] (emphasis added). That is, three conditions led to the problems; when only two conditions (without wet surfaces) held, the system functioned properly. In another example, we can identify four conditions that led to a crash: "Neither Autopilot nor the driver

noticed 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 …*" [6].

As we can see in these and other examples, there is a need to ensure that complex combinations of conditions have been tested. This need exists with most software, but the testing problem for autonomous systems is significantly different from conventional software. For life-critical aviation software, a structural coverage criterion known as modified condition decision coverage (MC/DC) [8] is used by the aerospace industry to achieve a high degree of assurance. It ensures that each decision and each condition has taken all possible outcomes, and each condition in a decision is shown to independently affect the decision outcome. Since the assurance processes incorporating this criterion are so successful, it seems natural to use the same approach for autonomous systems as well. Unfortunately, structural coverage measures do not apply so well to these systems.

The problem is that structural coverage measures are based on paths through the code, and many machine learning components make decisions using logic encoded in weights that are very different from conventional code. Various forms of neural network algorithms are used to create models for image recognition and other tasks, and the connections and weights of the generated net are dependent on the inputs used in training. Some research has developed measures such as neuron coverage, that are analogous to conventional structural code coverage [7], but it is not yet clear how closely this form of coverage is related to model performance. Moreover, it is not clear that neuron coverage will ensure that the input combinations encountered in practice will be handled correctly, just as we can have complete branch and path coverage yet have some input combinations that have not been tested.

Why not measure the input space coverage directly? Fortunately, concepts developed within the field of combinatorial testing can be applied in measuring how thoroughly combinations of inputs have been included in training and testing of machine learning models [9]. To illustrate these concepts, consider Fig. 1 below, with four inputs containing Boolean attributes *a, b, c,* and *d* :

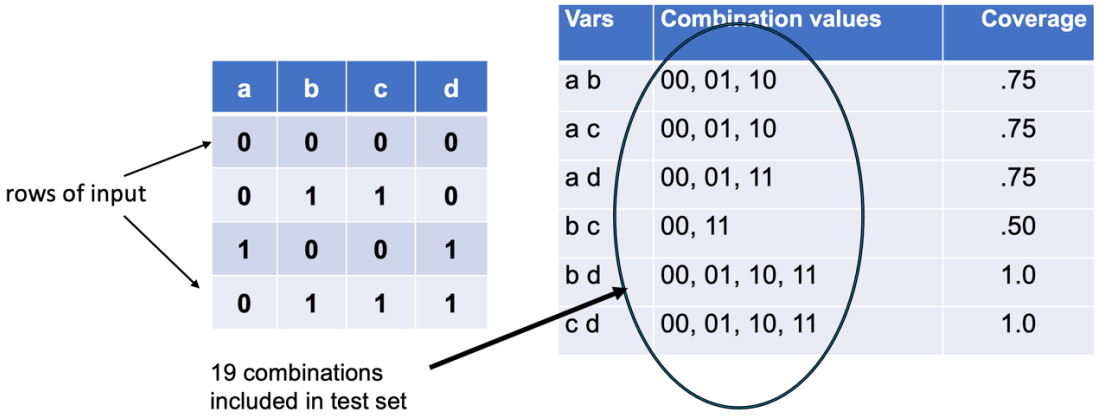

Fig. 1. Combination value coverage – a third of variable combinations have 1.0 value coverage, one-sixth have .50 coverage, and half have coverage of .75

Looking at 2-way combinations of these attributes, we can see that, for example, pair *ab* has values 00, 01, and 10 covered, or 75 % of the four possible value combinations for this pair. Continuing in this manner, we can determine the level of coverage for every pair of attribute values, finding 19 of a possible

24 settings have been covered – note that there are 2^2 possible settings possible for each of the $\binom{4}{2} = 6$ 2-way combinations, for a total of 24 possible. Rearranging this table a bit leads to a step graph (Fig. 2):

1.00	00	00				
.75	01	01	00	00	00	
.50	10	10	01	01	01	00
.25	11	11	10	10	11	11
	bd	cd	ab	ac	ad	bc

Fig. 2. Transforming coverage table to a step graph.

The area under the curve, designated S_2, is populated by the 2-way combinations that are covered in the four input rows. What else does this graph show? One interesting point is the minimum coverage, M_2, which is 0.50, because the combination *bc* is covered to this level and all others are above it. Additionally, as seen in Fig. 3 below, the area above the curve shows the *fraction of the total input space that is missing from training and testing data*. For our trivial example, this area represents *ab* = 11, *ac* = 11, *ad* = 10, *bc* = 01, 10. All other combinations are below the curve and have been included in training the machine learning model, but those in the orange space above the curve have not been included in training. This is where the performance of the ML model is unknown.

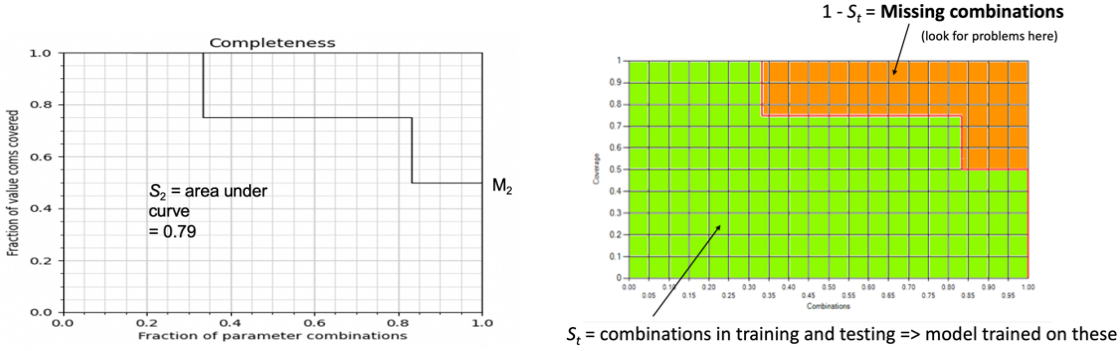

Fig. 3. Combination coverage

For an illustration of how these combinatorial coverage concepts can be developed and applied as useful measures, consider the problem of transfer learning [10]. In machine learning, differences are almost certain to exist between training data sets, test data sets, and later real-world data. Because the time and data available for training are finite, the training data may not include some input combinations that will eventually show up in use. Additional inconsistencies are likely for data from two or more usage environments. Transfer learning concerns predicting performance of a model trained on one data set when applied to another set of data, such as a new environment, changed environment, new ranges or values for data, or other changes.

Conventional methods of dealing with this problem include randomizing data selection and using very large volumes of training data to improve chances of including relevant inputs. But how much random data will be sufficient, especially with domains where a limited volume of data is available? Another strategy is to ensure inclusion of at least one of each object type in training data, but this may not adequately cover the range and distribution of object attributes (such as the color or reflectivity), or combinations of attributes of different object types in practice. As discussed above, the inability of an ML system to respond correctly to unanticipated inputs may lead to disaster. It is essential to consider potential interactions between inputs.

To illustrate the deficiencies of conventional approaches to transfer learning, and show how combinatorial methods produce improvements, we can consider an image recognition problem of aircraft in satellite imagery [10,11]. The problem is to determine if an image contains or does not contain an airplane, using two data sets: one from Southern California with 21 151 images, and one from Northern California with 10 849 images. Metadata were computed for the images and discretized resulting in 12 metadata features with three equal range bins each. If a model is trained on one set and applied to the other, a surprising result occurs. The model trained on the larger data set loses accuracy when applied to the smaller data set. Conversely, the model trained on the smaller data set shows no performance drop when applied to the large data set. This result seems backwards – isn't it better to have more data?

Using a measure called *set difference combinatorial coverage (SDCC)*, we can understand the result, and see that this counter-intuitive result could have been predicted from the SDCC measure [10]. Measures were developed for 2-way combinations of input data, i.e., all pairs of input values contained in the training data set. For C = larger, Southern California data; N = smaller, Northern California data: $SDCC(C \backslash N) = \frac{|C \backslash N|}{|C|} = 0.02$. This represents the feature combinations that are present in data set C, but absent in N, as a proportion of C. Also, $SDCC(N \backslash C) = \frac{|N \backslash C|}{|N|} = 0.12$, combinations in N not in C, as a proportion of N. Additionally, the basic combinatorial coverage S_2 (area under the curve in graphs introduced earlier) will be greater for data set N than it is for C. In other words, the N data set has fewer "never seen" combinations, even with roughly half as many observations. Combinatorial coverage provides a way to measure the degree of this greater coverage. Evaluating this metric in the training phase can help identify where even large volumes of data are insufficient. SDCC provides a way to measure the distance between the training domain and a new environment. Evaluating this metric during transfer learning can help identify when a model may not transfer well and can be used to identify samples for retraining that cover the gap. A number of these measures are now being studied to ensure that even rare combinations or sequences of inputs produce correct and safe responses from the autonomous system [12].

Conclusion

Conventional test methods based on structural coverage of code are not well suited to assurance of neural net models and other black-box techniques in machine learning. What matters most for these models is the inputs that are used in training and testing. Combinatorial methods provide ways to measure the adequacy of data used in developing ML models, enabling sound approaches for assured autonomy.

References

1. https://csrc.nist.gov/glossary/term/software_assurance
2. B. Beizer, *Software testing techniques*, Dreamtech Press, 2003. ISBN: 9788177222609.
3. G. Brat, Verification and Validation of Flight Critical Systems, invited talk at the Second NASA Formal Methods Symp., Washington, DC, 13 April 2010.
4. N. Kalra, and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, 182-193, 2016.

5. E. Ackerman, Korean competition shows weather still a challenge for autonomous cars. *IEEE Spectrum, Cars that Think Website*, 2014. Available: https://spectrum.ieee.org/japan-competition-shows-weather-still-a-challenge-for-autonomous-cars
6. "A Tragic Loss", Tesla Blog, June 30, 2016. Available: https://www.tesla.com/blog/tragic-loss
7. F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?" in *Proc. 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, pp. 851-862), 2020. doi: 10.5281/zenodo.4021473
8. K. J. Hayhurst, D. S., Veerhusen, J. J., Chilenski, and L. K. Rierson, "A practical tutorial on modified condition/decision coverage," Technical memorandum TM-2001-210876, National Aeronautics and Space Administration, Langley Research Center, Hampton, VA, 1 May 2001. Available: https://dl.acm.org/doi/10.5555/886632
9. D. R., Kuhn, I. Dominguez Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial coverage measurement concepts and applications," in *Proc. 2013 IEEE Sixth International Conf. on Software Testing, Verification and Validation Workshops*, pp. 352-361, March 2013. doi: 10.1109/ICSTW.2013.77
10. E. Lanus, L. J. Freeman, D. R. Kuhn, and R. N. Kacker, "Combinatorial Testing Metrics for Machine Learning," in *Proc. 2021 IEEE Intl. Conf. on Software Testing, Verification and Validation Workshops*, April 2021. doi: 10.1109/ICSTW52544.2021.00025
11. https://www.kaggle.com/code/pankajsharma127/planes-in-satellite-imagery
12. D. R. Kuhn, M. S. Raunak, and R. N. Kacker, Combinatorial coverage difference measurement, Technical report NIST CSWP 19 (initial public draft), National Institute of Standards and Technology, Gaithersburg, MD, 22 June 2021. doi: 10.6028/NIST.CSWP.19.ipd

D. Richard Kuhn is a computer scientist in the Computer Security Division at the National Institute of Standards and Technology, Gaithersburg, MD, 20899, U.S.A. He is an IEEE Fellow. Contact him at kuhn@nist.gov.

M. S. Raunak is a computer scientist in the Computer Security Division at the National Institute of Standards and Technology, Gaithersburg, MD, 20899, U.S.A. Contact him at ms.raunak@nist.gov.

Raghu N. Kacker is a scientist in the Applied and Computational Mathematics Division of the Information Technology Laboratory of the National Institute of Standards and Technology, Gaithersburg, MD, 20899, U.S.A. He is a fellow of the American Society for Quality and the American Statistical Association. Contact him at raghu.kacker@nist.gov.

Jaganmohan Chandrasekaran is a postdoctoral associate in the Intelligent Systems Division of the Virginia Tech National Security Institute, Arlington, VA, 22203, U.S.A. Contact him at jagan@vt.edu.

Erin Lanus is an assistant research professor in the Intelligent Systems Division of the Virginia Tech National Security Institute and affiliate faculty of Computer Science at Virginia Tech, Arlington, VA, 22203, U.S.A. Contact her at lanus@vt.edu.

Tyler Cody is an assistant research professor in the Intelligent Systems Division of the Virginia Tech National Security Institute, Arlington, VA, 22203, U.S.A. Contact him at tcody@vt.edu.

Laura Freeman is the deputy director of the Virginia Tech National Security Institute, Assistant Dean for Research for the College of Science, and a hub faculty member in the Commonwealth Cyber Initiative, Arlington, VA, 22203, U.S.A. Contact her at laura.freeman@vt.edu.