Testing cyber-physical-systems with explicit output coverage

Jarkko Peltomäki

Information Technology
Åbo Akademi University

27.5.2024

Joint work with J. Winsten, M. Methais, and I. Porres

CPS Testing

- A *cyber-physical system* (CPS) is a function $\mathcal{M} : \mathcal{I} \to \mathcal{O}$, where $\mathcal{I}$ ($\mathcal{O}$) is the input (output) space of $\mathcal{M}$.
- A requirement φ for a CPS is a requirement the CPS should satisfy whenever it is operated.
- Here we assume that a monitor ρ is given for φ to decide if the system behavior for a test $t \in \mathcal{I}$ complies with φ.
- Formally, ρ is a function $\mathcal{I} \times \mathcal{O} \to [0, 1]$ with the interpretations:
  - $\mathcal{M}$ does not comply with φ for a test $t \in \mathcal{I}$ if and only if $\rho(t, \mathcal{M}(t)) = 0$,
  - if $\rho(t, \mathcal{M}(t)) > 0$, the smaller $\rho(t, \mathcal{M}(t))$ is, the closer t is in not complying with φ.
- The latter condition ensures that we can find noncomplying inputs by minimizing ρ. For example, if φ is given as a signal temporal logic (STL) formula, ρ can be taken as appropriately scaled STL robustness monitor.
- We scale the monitor to $[0, 1]$ so that machine learning models can be used. Otherwise it might not be necessary.

A Running Example: BeamNG.tech

- The SBST 2021 workshop introduces the BeamNG.tech problem: create roads for a driving agent such that the agent drives out of its designated lane.
- Formally, input is a sequence of control points (fixed length) that determine a road, output is the *body-out-of-lane percentage* (BOLP) signal, and the requirement is $\Box\,\mathrm{BOLP}(t) \leq 0.95$.
- Monitor can be taken to be $\min\{1, \max\{0, 1 - \max_t \mathrm{BOLP}(t)/0.95\}\}$.

The Requirement Falsification Problem

- Given a CPS $\mathcal{M}$ with requirement φ and monitor ρ, the *requirement falsification problem* (RF-problem) asks to falsify φ, that is, to find $t \in \mathcal{I}$ such that $\rho(t) = 0$ if such a t exists.
  - "Find a road so that the car will drive out of its designated lane."
- A well-studied problem with dozens of algorithms solving it. For example, the annual ARCH-COMP friendly competition welcomes algorithms to be compared on a selection of CPS benchmarks.

Finding Several Falsifying Inputs

- In addition to finding a single falsifying input, it interesting to see the system fail in varied situations. For example, having multiple falsifying inputs may help in detecting the fault root cause.
- "Varied" can mean multiple things. We might be interested in structurally different tests (variety in the input space), different output behaviors (variety in the output space), different internal behavior (variety in the state space), or a combination of these.
- Variety can be specified
  - Implicitly: variety by maximizing dissimilarity to previous tests.
  - Explicitly: the desired variety is explicitly described.
- Here we focus on "explicit output coverage", that is, we look for different output behaviors explicitly determined by a set of several requirements. We target black-box systems so, in particular, we do not have access to the internals of the system.

The Output Requirement Problem

- The *output requirement problem* is about falsifying a set $\Phi = \{\varphi_1, \ldots, \varphi_N\}$ of output requirements related to a single CPS $\mathcal{M}$.
  A *witness* for Φ is a set of test whose tests together falsify all φ_i.

The Output Requirement Problem

- We are interested in algorithms that solve the output requirement problem.
- A straightforward solution is to take an algorithm solving the RF-problem and to use it sequentially on each requirement.
- This approach might be inefficient when the requirements are related: the process of finding a single output that satisfies one formula can be beneficial in finding inputs for the remaining requirements.
- Efficiency can be measured, e.g., on the number of tests that need to be executed to find a witness or on the total time to find a witness. Here we focus on the former.

BeamNG.tech Continued

- The BeamNG.tech simulator has an additional output $\mathrm{SA}(t)$ which is the driving agent's *steering angle signal* during the simulation.
- We want to observe the failure of keeping the lane with various steering angles. This is captured by the requirement

$$\gamma(\alpha, \beta) = \Diamond(\mathrm{BOLP}(t) \geq 0.95 \wedge \alpha \leq \mathrm{SA}(t) \leq \beta).$$

- Here we use the requirements $\varphi_1 = \gamma(-120°, -72°)$, $\varphi_2 := \gamma(-72°, -24°)$, $\varphi_3 := \gamma(-24°, 24°)$, $\varphi_4 := \gamma(24°, 72°)$, $\varphi_5 := \gamma(72°, 120°)$ which correspond to splitting the interval $[-120°, 120°]$ into five pieces.
- A witness is a set of 5 tests that together make the driver fail with steering angles in the given intervals.

Solving the Output Requirement Problem Efficiently

- We propose an explicit output coverage (EOC) algorithm that solves the output coverage problem efficiently using generative machine learning.
- Let us first look at WOGAN which is a RF-algorithm using generative ML.

WOGAN (P., Spencer, Porres 2022)

- Consider a requirement φ with monitor ρ (we want to find test t such that $\rho(t, \mathcal{M}(t)) = 0$).
- The idea of WOGAN is to train a generator $\mathcal{G}$ that is a probability distribution supported on tests that have low monitor value.
- The generator $\mathcal{G}$ is then sampled until a test t with $\rho(t, \mathcal{M}(t)) = 0$ is found.
- The generator is modeled as a Wasserstein Generative Adversarial Network (WGAN) which is trained to minimize the Wasserstein distance between the generator's distribution and the uniform distribution on tests t with $\rho(t, \mathcal{M}(t)) = 0$.
- The WOGAN algorithm achieves the training online by augmenting the WGAN training data with tests with lower and lower monitor value.
- WOGAN has been successful in falsification (SBST 2022, SBFT 2023,2024).

EOC Main Ideas

- Say we have requirements $\varphi_1, \ldots, \varphi_N$ with monitors $\rho_1, \ldots, \rho_N$.
- Create and enable N WOGAN generators $\mathcal{G}_1, \ldots, \mathcal{G}_N$, one for each φ_i.
- Sample one generator $\mathcal{G}_i$ for a test t (i selected uniformly randomly).
- If $\rho_j(t, \mathcal{M}(t)) = 0$ for some j, mark φ_j satisfied and disable $\mathcal{G}_j$ (monitor is fast to evaluate).
- Train all enabled generators on the executed tests (we know the monitor values of a test with respect to all requirements).

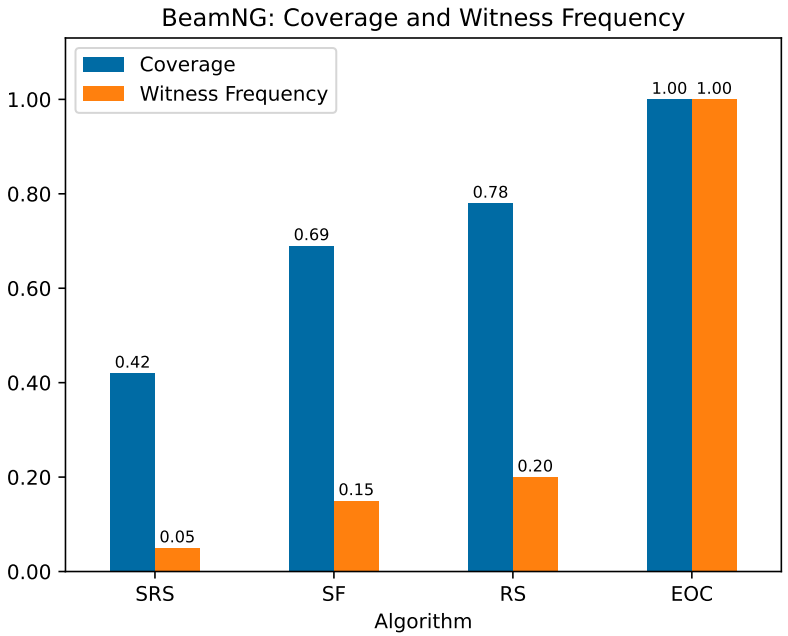
Intuitions

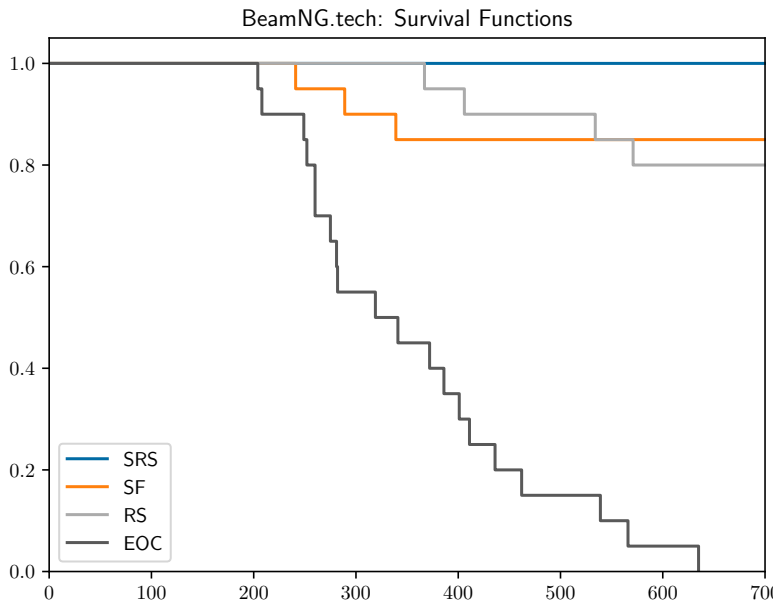
- Even when $\rho_j(t, \mathcal{M}(t))$ is large, it is possible that $\rho_k(t, \mathcal{M}(t))$ is low. This benefits the training of $\mathcal{G}_k$.
- All generators are used for sampling, but only one at a time (less test executions per round).
- Training can take as much as N times more resources when compared to the training of a single generator, but we assume that evaluating $\mathcal{M}(t)$ is slow, not the training of a generator.
- The use of generative ML allows to utilize data related to a sample from another generator. It is not always possible to do this (e.g., in gradient descent).

Results

- We used EOC to solve the output requirement problem for the BeamNG.tech problem (5 requirements). We did
- 20 replicas, total execution budget 700.
- Baseline algorithms used for comparison:
  - Sequential Random Search (SRS): use 5 random searches sequentially for each φ_i (budget $700/5 = 140$ per search).
  - Random Search (RS): sample 700 random tests and check which requirements where satisfied.
  - Sequential Falsification (SF): use WOGAN sequentially for each φ_i (budget $700/5 = 140$).

Results


BeamNG: Coverage and Witness Frequency
Coverage
Witness Frequency
1.00
0.80
0.60
0.40
0.20
0.00
0.42
0.05
0.69
0.15
0.78
0.20
1.00
1.00
SRS
SF
RS
EOC
Algorithm


BeamNG.tech: Survival Functions
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
SRS
SF
RS
EOC

Thank You for Your Attention

STGEM tool that implements EOC:
https://gitlab.abo.fi/stc/stgem (includes references to previous works).