Falsification of Multiple Requirements for Cyber-Physical Systems Using Online Generative Adversarial Networks and Multi-Armed Bandits

Jarkko Peltomäki

Information Technology
Åbo Akademi University

4.4.2020

Joint work with I. Porres

Context

- A cyber-physical system (CPS) is a system which consists of both software and hardware components.

Context

- A cyber-physical system (CPS) is a system which consists of both software and hardware components.
  - Example: self-driving car.

Context

- A cyber-physical system (CPS) is a system which consists of both software and hardware components.
  - Example: self-driving car.
- As CPSs interact with the real world, it is important to specify safety requirements and verify these requirements before the system is taken into production.

Context

- A cyber-physical system (CPS) is a system which consists of both software and hardware components.
  - Example: self-driving car.
- As CPSs interact with the real world, it is important to specify safety requirements and verify these requirements before the system is taken into production.
- Our work concerns black-box verification of CPSs. This means that we get to choose the inputs and to observe the behavior of a CPS without access to source code or other internals.

Setup

- We assume that the CPS is presented by a simulator which is a mapping from inputs x to output signals $\mathcal{M}(x)$ (possibly vector-valued).

Setup

- We assume that the CPS is presented by a simulator which is a mapping from inputs x to output signals $\mathcal{M}(x)$ (possibly vector-valued).
- We assume that the requirements are specified in signal temporal logic STL.

Setup

- We assume that the CPS is presented by a simulator which is a mapping from inputs x to output signals $\mathcal{M}(x)$ (possibly vector-valued).
- We assume that the requirements are specified in signal temporal logic STL.
  - E.g., $\square_{[0,30]}\text{SPEED} < 35$

Setup

- We assume that the CPS is presented by a simulator which is a mapping from inputs x to output signals $\mathcal{M}(x)$ (possibly vector-valued).
- We assume that the requirements are specified in signal temporal logic STL.
  - E.g., $\Box_{[0,30]}\mathrm{SPEED} < 35$
- An STL formula φ can be transformed into a real-valued robustness function ρ_φ such that φ is true if and only if $\rho_\varphi(\mathcal{M}(x)) > 0$ for all x.

Setup

- We assume that the CPS is presented by a simulator which is a mapping from inputs x to output signals $\mathcal{M}(x)$ (possibly vector-valued).
- We assume that the requirements are specified in signal temporal logic STL.
  - E.g., $\Box_{[0,30]}\text{SPEED} < 35$
- An STL formula φ can be transformed into a real-valued robustness function ρ_φ such that φ is true if and only if $\rho_\varphi(\mathcal{M}(x)) > 0$ for all x.
- This allows to express falsification of φ as an optimization problem minimizing the robustness. Let

$$x^* = \arg\min_x \rho_\varphi(\mathcal{M}(x)).$$

Then φ is falsified if and only if $\rho_\varphi(x^*) \leq 0$.

Algorithms

- The optimization problem can be solved in principle by any optimization algorithm.

Algorithms

- The optimization problem can be solved in principle by any optimization algorithm.
- However, for CPSs evaluating $\mathcal{M}(x)$ can be costly, so optimization methods minimizing the number of evaluations are preferred. Some recent examples:
  - Online generative adversarial network test generation algorithm (Porres et al. 2021).
  - Bayesian optimization based algorithm (Mathesen et al. 2021).

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.
  - Pros: simple to implement, single formula (it is more complicated though).

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.
  - Pros: simple to implement, single formula (it is more complicated though).
  - Cons: does not take different behaviors of formulas into account (say φ_1 unfalsifiable, φ_{100} easily falsifiable).

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.
  - Pros: simple to implement, single formula (it is more complicated though).
  - Cons: does not take different behaviors of formulas into account (say φ_1 unfalsifiable, φ_{100} easily falsifiable).
- Idea 2: Use a generator for each formula φ_i (total n generators).

Multiple Requirements

- Say we have multiple requirements $\varphi_1, \ldots, \varphi_n$.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.
  - Pros: simple to implement, single formula (it is more complicated though).
  - Cons: does not take different behaviors of formulas into account (say φ_1 unfalsifiable, φ_{100} easily falsifiable).
- Idea 2: Use a generator for each formula φ_i (total n generators).
  - Pros: simple to implement, uses information from all φ_i.

Multiple Requirements

- Say we have multiple requirements φ_1, ..., φ_n.
- How should we approach the falsification?
- We assume that for a formula φ we have a test generator algorithm G_φ which proposes candidate tests x for minimization.
- Idea 1: Form the conjunctive requirement $\varphi := \varphi_1 \land \cdots \land \varphi_n$ and use your favorite generator algorithm G_φ for this single requirement.
  - Pros: simple to implement, single formula (it is more complicated though).
  - Cons: does not take different behaviors of formulas into account (say φ_1 unfalsifiable, φ_{100} easily falsifiable).
- Idea 2: Use a generator for each formula φ_i (total n generators).
  - Pros: simple to implement, uses information from all φ_i.
  - Cons: Uses n times more resources.

Main Research Question

- We would like to use Idea 2 (take into account information from all φ_i) but use less resources (ideally as little as in Idea 1).

Main Research Question

- We would like to use Idea 2 (take into account information from all φ_i) but use less resources (ideally as little as in Idea 1).
- Solution: pick the formula which is easiest to falsify and use a generator only for it.

Main Research Question

- We would like to use Idea 2 (take into account information from all φ_i) but use less resources (ideally as little as in Idea 1).
- Solution: pick the formula which is easiest to falsify and use a generator only for it.
- Problem: how do we know which formula is easiest?

Main Research Question

- We would like to use Idea 2 (take into account information from all φ_i) but use less resources (ideally as little as in Idea 1).
- Solution: pick the formula which is easiest to falsify and use a generator only for it.
- Problem: how do we know which formula is easiest?
- We cannot know this, but we can attempt learn this online.

Solution: Multi-armed Bandit Algorithms

- In the multi-armed bandit problem, we have n slot machines (arms) with each having their random reward r_n (with unknown distribution). On each round, one of the arms is selected and reward r_n is obtained. What is the best strategy to maximize the total reward?

Solution: Multi-armed Bandit Algorithms

- In the multi-armed bandit problem, we have n slot machines (arms) with each having their random reward r_n (with unknown distribution). On each round, one of the arms is selected and reward r_n is obtained. What is the best strategy to maximize the total reward?
- This is a well-studied problem, and the literature provides dozens of good algorithms.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).
- Thus, on each round, only one generator is considered and the cost per round is equal to running one generator.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).
- Thus, on each round, only one generator is considered and the cost per round is equal to running one generator.
- The MAB algorithm (selection of the arm) is as follows:
  - For a warm-up period of M rounds, select all n arms.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).
- Thus, on each round, only one generator is considered and the cost per round is equal to running one generator.
- The MAB algorithm (selection of the arm) is as follows:
  - For a warm-up period of M rounds, select all n arms.
  - On each round, record which generator achieved the lowest robustness. This yields an empirical success frequency for each arm.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).
- Thus, on each round, only one generator is considered and the cost per round is equal to running one generator.
- The MAB algorithm (selection of the arm) is as follows:
  - For a warm-up period of M rounds, select all n arms.
  - On each round, record which generator achieved the lowest robustness. This yields an empirical success frequency for each arm.
  - After the warm-up period, select randomly an arm with probabilities according to the success frequencies, use the corresponding generator, update frequencies, repeat.

MAB and Multiple Requirements

- In our approach we use Idea 2 and the n generators (one for each φ_i) are the arms of the MAB problem.
- On each round, a generator i is selected, a test input x is generated, and the robustness $\rho_{\varphi_i}(\mathcal{M}(x))$ is computed (reward).
- Thus, on each round, only one generator is considered and the cost per round is equal to running one generator.
- The MAB algorithm (selection of the arm) is as follows:
  - For a warm-up period of M rounds, select all n arms.
  - On each round, record which generator achieved the lowest robustness. This yields an empirical success frequency for each arm.
  - After the warm-up period, select randomly an arm with probabilities according to the success frequencies, use the corresponding generator, update frequencies, repeat.
- Intuitively this algorithm most often considers the requirement which is easiest to falsify (has lowest robustness values fastest). Due to random chance, other requirements are occasionally considered as well (so there is a chance of correcting a wrong preference).

Experimental Results

- Consider the function (Mathesen et al.) $\text{mo3d}\colon \mathbb{R}^3 \to \mathbb{R}^3, \text{mo3d}(x) = (h_1(x), h_2(x), h_3(x))$ where

$$h_1(x_1, x_2, x_3) = 305 - 100 \sum_{i=1}^{3} \sin\left(\frac{x_i}{3}\right),$$

$$h_2(x_1, x_2, x_3) = 230 - 75 \sum_{i=1}^{3} \cos\left(\frac{x_i}{2.5} + 15\right), \text{ and}$$

$$h_3(x_1, x_2, x_3) = \sum_{i=1}^{3} (x_i - 7)^2 - \sum_{i=1}^{3} \cos\left(\frac{x_i - 7}{2.75}\right).$$

Experimental Results

- Consider the function (Mathesen et al.) $\text{mo3d}\colon \mathbb{R}^3 \to \mathbb{R}^3, \text{mo3d}(x) = (h_1(x), h_2(x), h_3(x))$ where

$$h_1(x_1, x_2, x_3) = 305 - 100 \sum_{i=1}^{3} \sin\left(\frac{x_i}{3}\right),$$

$$h_2(x_1, x_2, x_3) = 230 - 75 \sum_{i=1}^{3} \cos\left(\frac{x_i}{2.5} + 15\right), \text{ and}$$

$$h_3(x_1, x_2, x_3) = \sum_{i=1}^{3} (x_i - 7)^2 - \sum_{i=1}^{3} \cos\left(\frac{x_i - 7}{2.75}\right).$$

- Requirements $\varphi_1 = \Box h_1 > 0$, $\varphi_2 = \Box h_2 > 0$, $\varphi_3 = \Box h_3 > 0$.

Experimental Results

- Consider the function (Mathesen et al.) $\mathrm{mo3d}\colon \mathbb{R}^3 \to \mathbb{R}^3, \mathrm{mo3d}(x) = (h_1(x), h_2(x), h_3(x))$ where

$$h_1(x_1, x_2, x_3) = 305 - 100 \sum_{i=1}^{3} \sin\left(\frac{x_i}{3}\right),$$

$$h_2(x_1, x_2, x_3) = 230 - 75 \sum_{i=1}^{3} \cos\left(\frac{x_i}{2.5} + 15\right), \text{ and}$$

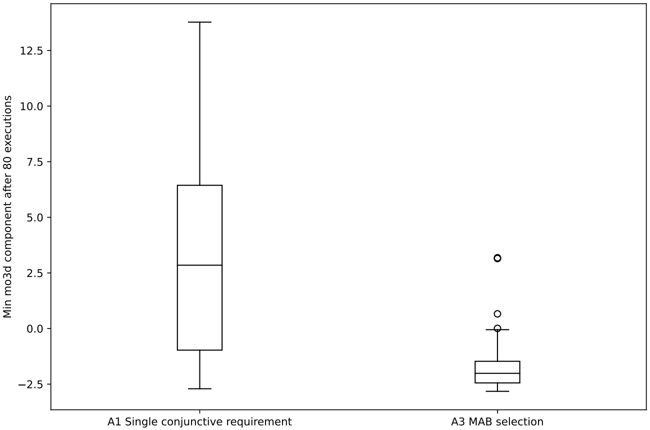
$$h_3(x_1, x_2, x_3) = \sum_{i=1}^{3} (x_i - 7)^2 - \sum_{i=1}^{3} \cos\left(\frac{x_i - 7}{2.75}\right).$$

- Requirements $\varphi_1 = \Box h_1 > 0$, $\varphi_2 = \Box h_2 > 0$, $\varphi_3 = \Box h_3 > 0$.
- φ_3 falsifiable, φ_1, φ_2 unfalsifiable.

Experimental Results

- We compared falsification of $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ (A1) versus falsification of φ_1, φ_2, φ_3 using the MAB approach (A3).
- We used online generative adversarial network algorithm (Porres et al.) for the generator.
- We allowed 80 executions on the system with warm-up period 30.
- We repeated the falsification task 50 times.

Experimental Results



Future Work

- More experiments.
- Compare to other algorithms.
- Consider other MAB algorithms.

Thank You

Thank you for your attention!