Automated Testing of Cyber-Physical Systems
Shiva Nejati
SnT Centre/University of Luxembourg

Huawei Workshop
December 15, 2017
Acknowledgements

- Lionel Briand
- Raja Ben Abdessalem
- Reza Matinnejad
- Industry partners: IEE, SES and Delphi
Cyber Physical Systems

Cyber Space

Physical Sensing

Object Domain

Networks

Actuation Information

Real Space

Model-Based Development
CPS Model-Based Development

Model in the Loop (MiL)
- Function modeling (Matlab/Simulink)
  - Controller
  - Plant/Environment

Software in the Loop (SiL)
- Architecture modeling (C-Code/SysML)
  - Real-time analysis
  - Integration

Hardware in the Loop (HiL)
- Deployment (embedded-C)
  - Testing (Expensive)
Do we find an error by testing models?
Fundamental Questions

- What are the useful and realistic models of CPSs?
- How to specify test oracles to enable effective testing of system requirements and design?
- How to design scalable testing techniques?
  - Test case generation
  - Test case selection
  - Fault localization
CPS Models

• have **dynamic** behaviors

• are **executable**

• are **hybrid** – capture both discrete (algorithms) and continuous (physical dynamics) computations

• exhibit **uncertainty** e.g., about the environment
Open Loop Controllers

Actuator

Controller

Reference Inputs
Closed Loop Controllers

Controllers + Plants
Autonomous Controllers

Controllers + Plants + Decision
CPS Test Oracles

• System outputs are **signals**

  • Engineers inspect changes in outputs over continuous time periods

• Test oracles

  • may be **heuristic** or **partial**
  
  • are often **quantitative** and not binary

  • might be **effort-intensive** or difficult to **automate**
Anti Patterns—Partial Oracles

- Instability
- Growth to infinity
- Discontinuity
Application Specific Oracles

• A reference signal + error margin

• (Sequences of) Signal features

• Temporal properties: "The system response should occur within 32ms"
CPS Testing Challenges

• Test input space is large and multi-dimensional

• Model executions are time consuming

• Fault localization is difficult

• Limited time budget for testing
  • Test oracles are expensive
  • Running the test cases on HiL is expensive
## Our Solutions

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Our solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test input space is large</td>
<td><strong>Metaheuristic search</strong> to identify <strong>worst case/critical behaviors</strong></td>
</tr>
<tr>
<td>Simulation takes time</td>
<td><strong>Surrogate models</strong> to predict the simulation outcome <strong>without running simulations</strong></td>
</tr>
<tr>
<td>Fault localization is difficult</td>
<td><strong>Classification techniques</strong> to explain system failures</td>
</tr>
<tr>
<td>Expensive HiL Testing</td>
<td><strong>Test case prioritization</strong> using <strong>multi-objective search</strong></td>
</tr>
</tbody>
</table>
Example Projects
Testing Advanced Driver Assistance Systems
Advanced Driver Assistance Systems (ADAS)

Decisions are made over time based on sensor data
Testing Advanced Driver Assistance Systems (self-driving cars)

Models -- A simulator based on Physical/Mathematical models

Oracles -- description of crashes

- Test generation based on meta-heuristic search
- Surrogate modeling to speed up search
- Classification to help with fault localization
Automated Emergency Braking System (AEB)

Decision making

- Vision (Camera)
- Sensor
- Objects’ position/speed

“Brake-request” when braking is needed to avoid collisions

Brake Controller
Physics-Based Simulations
Example:

CB: “AEB detects a pedestrian in front of the car with a high degree of certainty, but an accident happens where the car hits the pedestrian with a relatively high speed”
Generating Critical Test Scenarios via Metaheuristic Search
Black-Box Search-based Testing

Input data ranges/dependencies + Simulator + Fitness functions defined based on Oracles

Test input generation
- Select best tests
- Generate new tests (genetic operators)

Evaluating test inputs
- Simulate every (candidate) test
- Compute fitness functions

(candidate) test inputs
Fitness values

Test cases revealing worst case system behaviors
An example critical scenario
Improving Search Time Performance via Surrogate (Prediction) Models
Improving Time Performance

- Individual simulations take on average around 1 min.
- It takes 8 hours to run our search-based test generation (≈ 500 simulations).

→ We use surrogate modeling to improve the search.
  - **Goal:** Predict fitness based on dynamic variables.
  - Neural networks.
Surrogate Modeling

Input data ranges/dependencies + Simulator + Fitness functions defined based on Oracles

- Select best test inputs
- Generate new tests (genetic operators)

Uses prediction values & prediction errors to run simulations only for the solutions likely to be selected

- Simulate every (candidate) test
- Compute fitness functions

Fitness values

Test cases revealing worst case system behaviors
Results – Surrogate Modeling

(a) Comparing HV values obtained by NSGAII and NSGAII-SM

(b) Comparing HV values obtained by RS and NSGAII-SM

(c) HV values for worst runs of NSGAII, NSGAII-SM and RS
Guiding Search via Classification Models
Search Guided by Classification

Input data ranges/dependencies + Simulator + Fitness functions defined based on Oracles

Test input generation:
- Build a classification tree
- Select/generate tests in the fittest regions
- Apply genetic operators (Optional)

Evaluating test inputs:
- Simulate every (candidate) test
- Compute fitness functions

(candidate) test inputs → Fitness values

Test cases revealing worst case system behaviors + A characterization of critical input regions
Initial Classification Model

All Test Scenarios

- Count: 1200
  - non-Critical: %20 - %80

Road topology (CR=[10—40] )
- Count: 564
  - non-Critical: %42 - %58
- Count: 636
  - non-Critical: %2 - %98

Road topology (CR=5, Straight, RH=[4—12] )
- Count: 412
  - Critical: %51 - %49
  - Count: 152
    - non-Critical: %16 - %84
- Count: 230
  - Critical: %68 - %32
  - Count: 182
    - non-Critical: %28 - %72
Refined Classification Model
Outputs of Our Approach

- **Failure Detection**

  - (Search + Classification) generates 78% more distinct, critical test scenarios compared to a baseline search algorithm

- **Failure Explanation**

  - A characterization of the input space showing under what input conditions the system is likely to fail

  - Visualized by diagrams or regression trees
Failure Explanation

vehicle speed > 36km/h

pedestrian speed < 6km/h

[15m-40m]

road

sidewalk
Usefulness

The characterizations of the different critical regions can help with:

(1) **Debugging** the system or the simulator

(2) **Identifying hardware changes** to increase ADAS safety

(3) **Identifying proper warnings** to drivers
Other Project Examples
Automotive Systems

- Testing controller implemented in Simulink
- Analysis of CPU time usage in ECU software
- Fault localisation in Simulink models
Model Testing Satellite Systems

- Control system
- MiL/SiL testing

- Data communication system
- Test case prioritization for HiL
Conclusions
Model Testing

Do we find an error by testing models?

- Search
- Prediction models
- Classification models
Model Checking

- Symbolic techniques
- Exhaustive search via SAT/SMT solvers

Do models satisfy formal properties?
Related Work: Model Checking

- **Incompatibility** issues with CPS models
  - Continuous mathematical models, e.g., differential equations
  - Library functions in binary code
  - Non-linear behavior
    - Complex mathematical operators
    - Saturation of actuators and sensors
    - Reliance on measured data
Related Work: Model Checking

- Unrealistic assumptions about CPS test oracles
  - Discrete/exact/complete/binary/automatable
  - Focus on structural coverage

- Scalability
Search-Based Solutions

- Are Versatile
  - Decrease *modeling* requirements
  - Relax assumptions on *test oracles*
- Are scalable, e.g., *easy to parallelize*
- Can be combined with: *Machine learning; Statistics; Solvers, e.g., SMT, CP*
- But,
  - are *context-dependent*
  - require *massive empirical studies*
Future Work

- Model testing solutions in other CPS contexts
  - Heterogeneous modeling and co-simulation
  - Modeling dynamic properties and risk
  - Uncertainty modeling enabling probabilistic test oracles
  - Executable model at a proper level of precision for testing purposes
  - Systematic ways to build fitness functions for oracles
References


• Reza Matinnejad, Shiva Nejati, Lionel C. Briand, Thomas Bruckmann, “Effective Test Suites for Mixed Discrete-Continuous Stateflow Controllers”, ACM ESEC/FSE 2015 (Distinguished paper award)

• Reza Matinnejad, Shiva Nejati, Lionel C. Briand, Thomas Bruckmann, “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, IEEE/ACM ASE 2014 (Distinguished paper award)

Automated Testing of Cyber-Physical Systems

Shiva Nejati
SnT Centre/University of Luxembourg

Huawei Workshop on Applications of AI to Software Engineering

December 15, 2017
Results

- The test scenarios by our search-based approach helped engineers **identify** several **critical behaviors**
  - The critical test scenarios are available at: 
    - https://sites.google.com/site/testingpevi

- Under tight time budget, our search algorithm with surrogate models is more **accurate** and **safer** compared to the baseline search algorithm

- Our classification guided search generates 78% more **distinct**, **critical test scenarios** compared to the baseline search algorithm
Part II. Model Testing Satellite Systems
Test Case Prioritization (HiL)

• Problem
  • Test case prioritization

• Context
  • System validation and acceptance testing of CPS

Search space: exponential growth
E.g., two test cases: \( a, b \)
Possible test suites: \( (a), (b), (a,b), (b,a) \)

Black box testing
Results – Worst Runs

- (a) Comparing HV values obtained by NSGAII and NSGAII-SM
- (b) Comparing HV values obtained by RS and NSGAII-SM
- (c) HV values for worst runs of NSGAII, NSGAII-SM and RS