Validation of Industrial Cyber-Physical Systems: an application to HVAC systems

Thao Dang (VERIMAG) Alie El-Din Mady (UTC)
Menouer Boubekeur (UTC) Rajesh Kumar (UTC)
Mark Moulin (UTC)

Plan

1. Validation of Cyber-Physical Systems
2. Coverage-Guided Testing Generation
3. Tool Implementation and Execution
4. Application to the HVAC
5. Future Work

Validation of Cyber-Physical Systems

- Cyber-Physical Systems are integrations of computation with physical processes (Lee2006)
- Safety-critical: assuring correct behaviours of CPS is crucial
- Formal verification (exhaustive analysis): high computational complexity, limited to applications for low-dimensional systems
- Testing: validation technique par excellence in industrial practice, applicable to high-dimensional systems
- **Our goal**: adapt a hybrid systems testing technology to industrial CPS

Case study: a HVAC system modeled using Simulink

Complexity of the case study

- Mixture of hierarchical modelling formalisms (Simulink discrete and continuous blocks, lookup tables, embedded Matlab code)
- Mixture of dynamics: continuous thermodynamics and discrete control strategies

Interest of the case study

- Simulink is a standard industrial formalism for CPS
- Simulink exhibits the major challenges in validation of CPS (lack of formal semantics)
- HVAC model: real-life industrial model

Plan

1 Validation of Cyber-Physical Systems

2 Coverage-Guided Testing Generation

3 Tool Implementation and Execution

4 Application to the HVAC

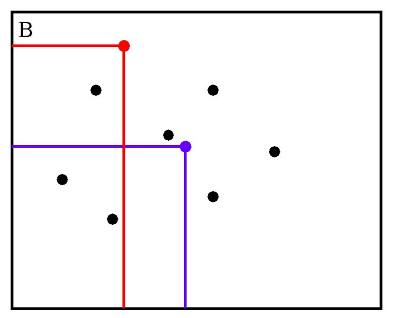
5 Future Work

Testing Methodology

Goal: adapt our hybrid systems testing technology to industrial CPS

- Consider a complex Simulink model as a **black-box system**, to **avoid semantics issues**
- Define a **coverage measure** to characterize the portion of tested behaviors
- Use the covergage measure to **guide the test generation process**

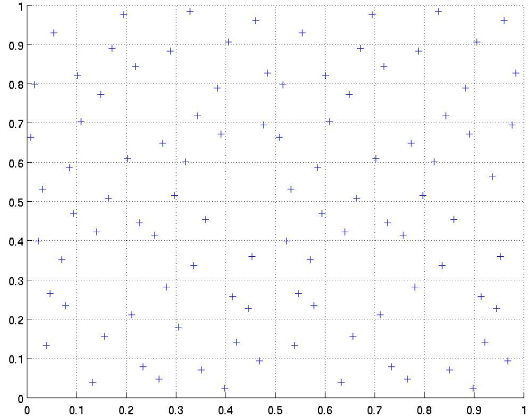
Test Coverage Measure



- Let P be a set of k points inside $B = [l_1, L_1] \times \ldots \times [l_n, L_n]$.
- Local discrepancy for a subbox **J**: $D(P, J) = |\frac{\#(P,J)}{k} - \frac{vol(J)}{vol(B)}|$.
- Example: $D(P, \mathbf{J}) = |\frac{\mathbf{2}}{\mathbf{7}} - \frac{\mathbf{1}}{\mathbf{4}}|$

Test Coverage Measure - Meaning

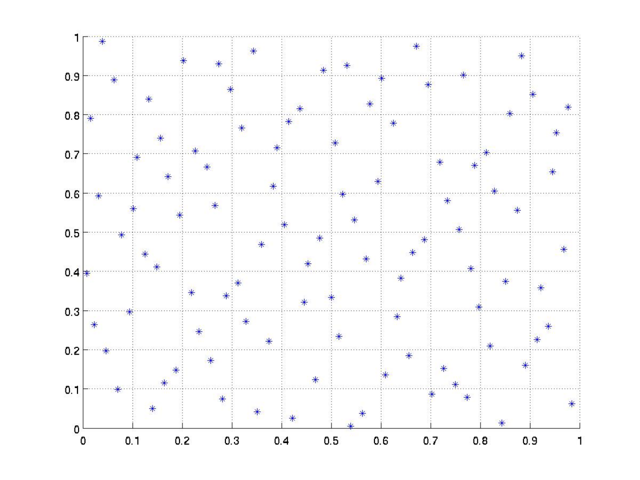
Degree of equidistribution of a set of points



Faure sequence of 100 points. Its star discrepancy value is 0.048 (well-equidistributed)

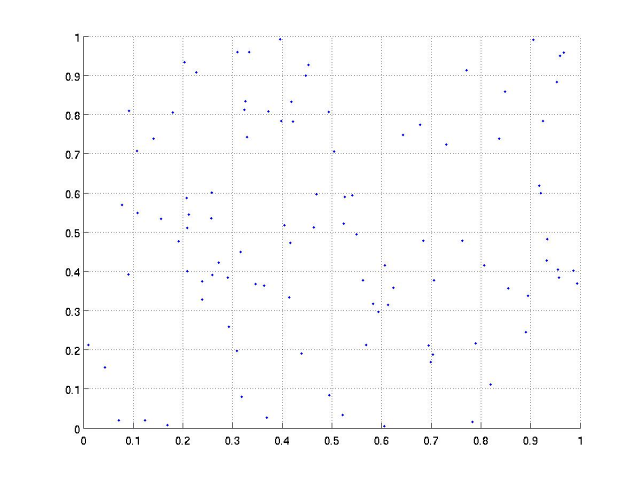
Test Coverage Measure - Meaning

Degree of equidistribution of a set of points



Halton sequence of 100 points. The star discrepancy value is 0.05 (less well-equidistributed)

Test Coverage Measure - Meaning



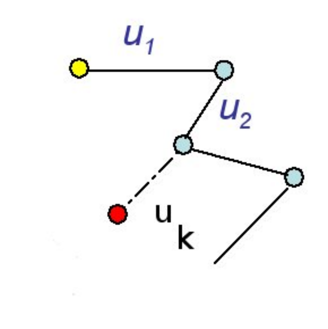
Sequence of 100 points generated by a **pseudo-random function in the C library**. Its star discrepancy value is 0.1 (poorly-equidistributed)

Test Generation

- **Randomized** exploration, inspired by probabilistic **motion planning** techniques **RRT** (Random Rapidly-Exploring Trees) in robotics (LaValle2000)
- **Coverage measure** reflects testing quality
- **Guided** by coverage criteria

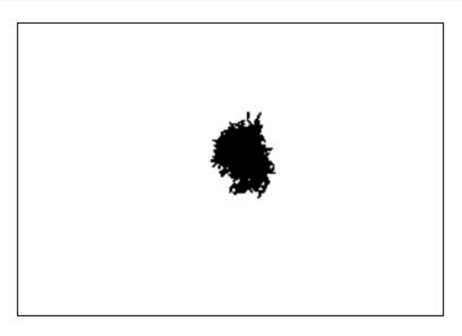
Simple randomized exploration

Pick a (visited) state, pick an input value, simulate one step and repeat



Simple randomized exploration

Bad coverage



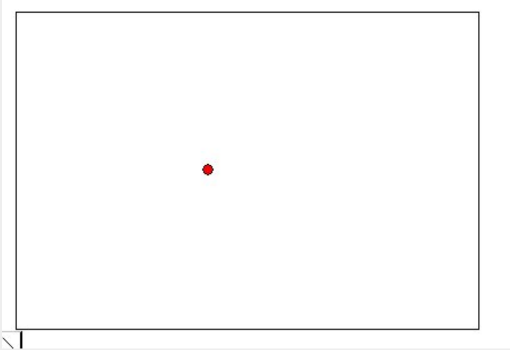
RRT Algorithm

$$\dot{x}(t) = f(x(t), u(t))$$

$\mathcal{T}.init(x_0), k = 1$ /* x_0: initial state */
Repeat
  $x_{goal} = \textsc{CoverageGuidedSampling}(X, \mathcal{T})$ /* X: state space */
  $x_{near} = \textsc{Neighbor}(\mathcal{T}, x_g)$
  $x_{new} = \textsc{Evolution}(x_n, h)$ /* h: time step */
  $\mathcal{T}.add(x_{new})$
  $k++$
Until $k \geq k_{max}$

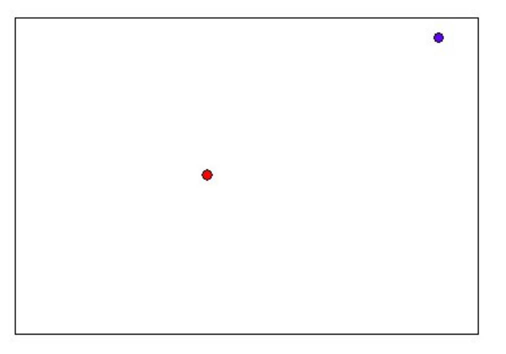
- Sample a goal state x_{goal} (to improve the coverage)
- Choose x_{near} to be a state near x_{goal}
- Procedure EVOLUTION tries to find the input u to take the system from x_{near} towards x_{goal}, as closely as possible
- The RRT algorithm can be extented to **hybrid dynamics** (provided that the procedure EVOLUTION can be computed by a simulator)

RRT Algorithm



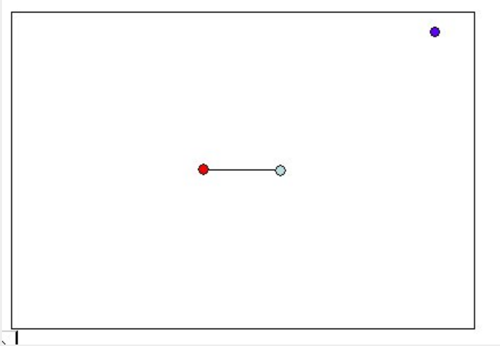
Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm



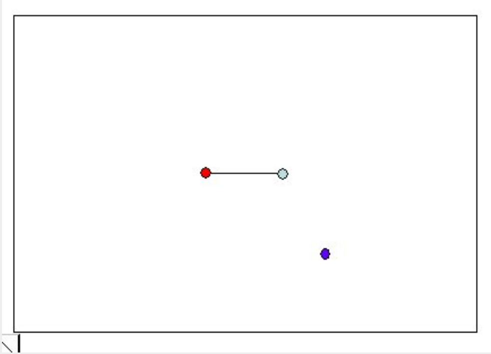
Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm



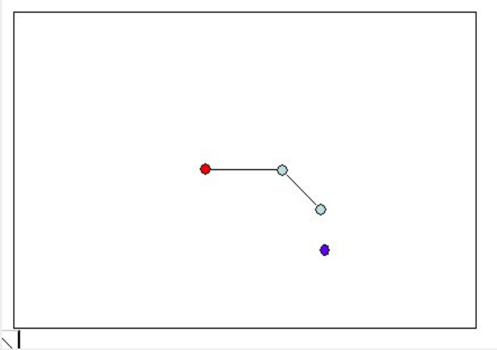
Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm



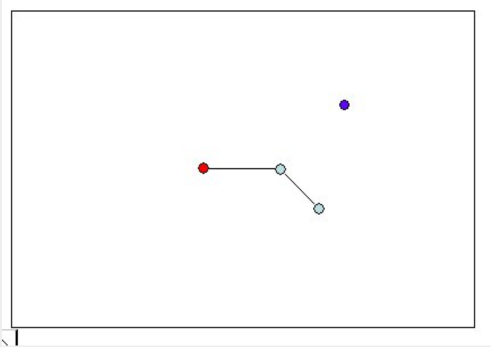
Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm



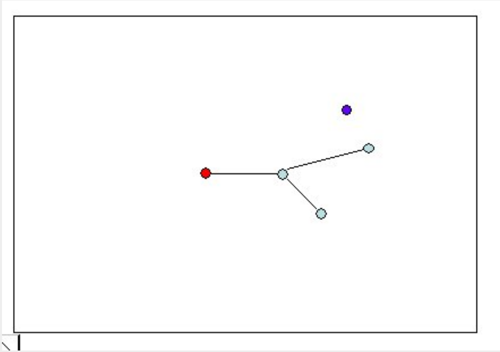
IInitial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm



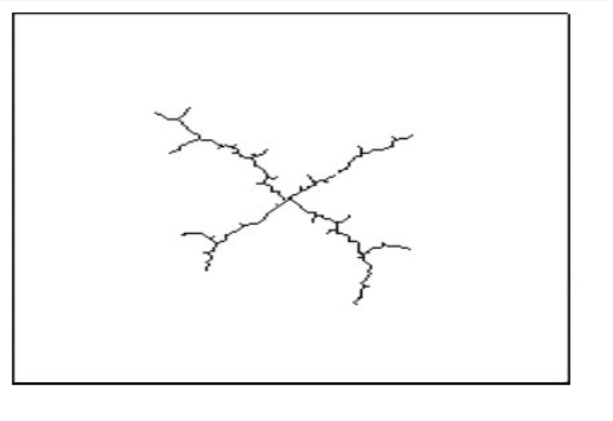
Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

RRT Algorithm

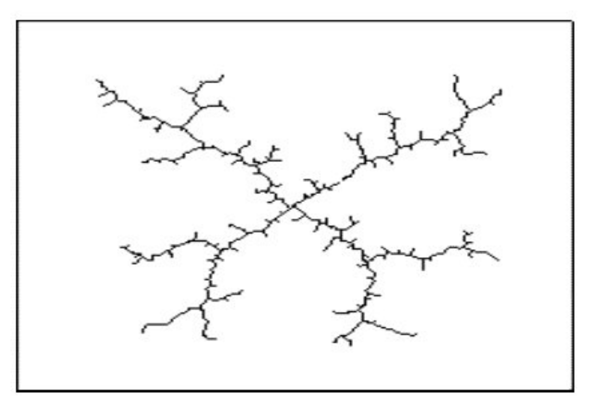


Initial state: **red**; Goal state: **dark blue**; Neighbor and new states: **cyan**

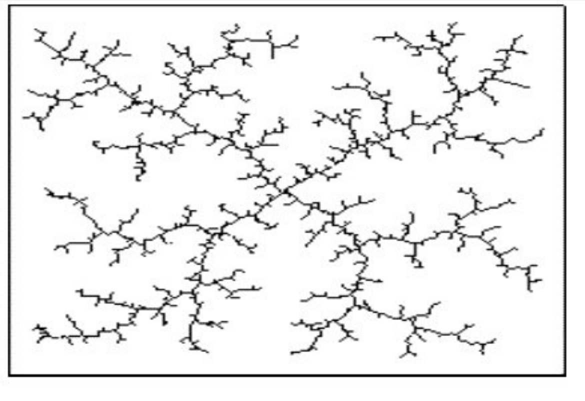
RRT Algorithm - example



RRT Algorithm - example



RRT Algorithm - example



Comparing with usual simulation approaches

Usual simulation approaches

- In **usual simulation settings**, **external inputs** from the environment should be **fully defined** before each simulation run
- Test cases are **single simulation traces**

Our approach

- Inputs can **dynamically varied** during a single simulation run, to improve coverage
- Test cases are a **tree of simulation traces**

Applications to Simulink models

Problems to address

- Identify state variables x, input variables u (to be controlled by the tester)
- Reinitialize the state variables to x_{near} in each iteration
- Choose an input u (random selection can be sufficiently good!)
- Compute EVOLUTION (can be done by Simulink simulator!)

State variables of a Simulink model

Problem

- Only **explicit state variables** are reported ⇒ **can be reinitialized** by setting the initial conditions of the blocks
- There are **hidden states** that are not reported ⇒ **cannot be reinitialized** by the user
- Industrial Simulink models (such as HVAC) contain blocks (such as continuous-time delays) and Matlab code, which essentially represent systems with an **infinite number of state variables**

Solution

- **Retrieve the sequence of input values** in the simulation tree $\mathcal{T}$ that leads from the initial state x_0 (root) to the state to restore x_{near}.
- Let the simulator **restart from the initial state** x_0, under the retrieved input sequence

Substate space coverage

Remark

- Star discrepancy estimation and neighbor computation become **expensive in high dimensions**
- Some **critical variables** have more influence on the **satisfaction or violation of the property**

Solution

- Trying to have a good coverage only on critical variables, called **covered variables** $\Rightarrow$
  - reduce computation complexity
  - allow discovering interesting behaviours more efficiently

Combining linear and branching traces

Remark

- Forming a tree by **branching** allows **good coverage** but is **expensive**
- Conditioning the simulation from some given states, in order to quickly lead to interesting behaviours
  - Construct segments of long **linear traces** (that is, traces in which the current state is also the next starting state) **between the subtrees**
  - To avoid simulating from the root of the tree, we could save the SimStates at the roots of these subtrees.

Plan

1 Validation of Cyber-Physical Systems

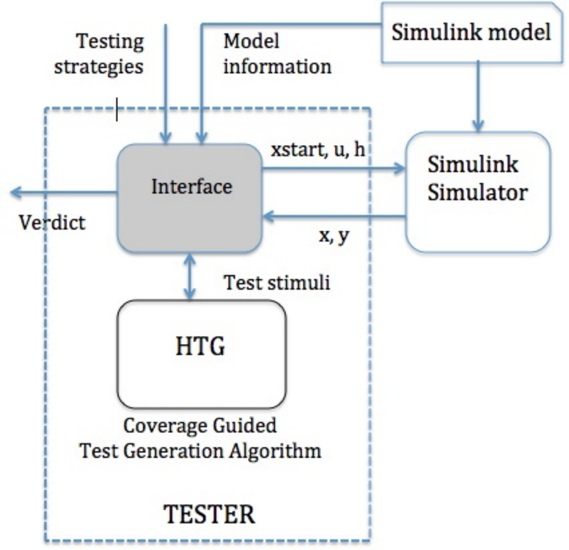
2 Coverage-Guided Testing Generation

3 Tool Implementation and Execution

4 Application to the HVAC

5 Future Work

Tester Structure



Interface

An interface (implemented as a Matlab program) between Simulink simulator and the algorithmic library of the Hybrid Systems Test Generation HTG tool (Dang2010), in charge of:

Extracting model information

1. Generate the list of input variables u, output variables y, and (explicit) state variables x of the model.
2. Identify among u the **inputs** that will be controlled by the tester. They are then replaced by Simulink **input ports**.
3. Identify among y the **observed outputs**, which are signals involved in the property to test. These signals are replaced by Simulink **output ports**.
4. Identify among x the **covered variables**.

Communication

- Communication between the **test generation algorithms** of the HTG tool (implemented in C++) and the **Simulink simulator**

Computation and Parameter settings

User writes a Matlab script to

- specify parameters and computation settings
- specify inputs, initial conditions, and covered state variables
- guide strategies reflecting specific scenarios the designer wants to test, a-priori knowledge (illustrated by the dependency flow graph in the application to the HVAC model)
- invoke the tool execution
- save testing data

Plan

1 Validation of Cyber-Physical Systems

2 Coverage-Guided Testing Generation

3 Tool Implementation and Execution

4 Application to the HVAC

5 Future Work

HVAC system structure

Property

- **Safety property**: opening (in %) of the cooling and heating valves should not be greater than 0 at the same time
- Activating cooling and heating at the same time leads to a high energy waste

Testing strategies: Initial state variation and Covered state variables

Initial state variation

- Vary the initial conditions of the **integrator of the zone temperatures**.
- Variation range is $[19, 23]$, while in the original model they were fixed at 22 (Celsius degrees).

Covered state variables

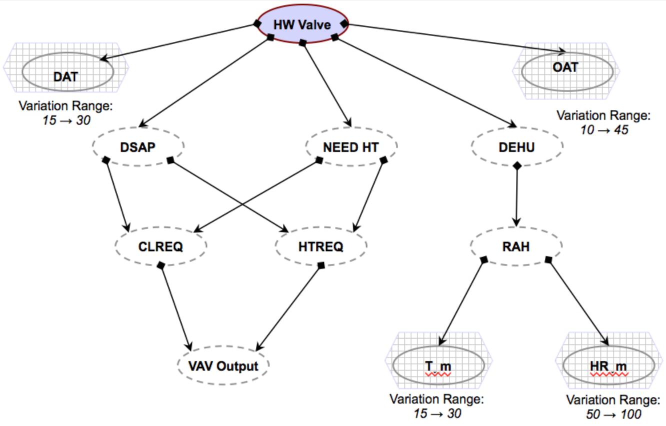
- We focused on covering a subspace formed by a few explicit state variables
- We chose them to be the **states of the integrator block** (important continuous component modelling the evolutions of the zone temperatures)

Testing strategies: Input Variations

Which inputs to vary?

- In **usual simulation settings**, external inputs from the environment are all set to **constant** for each simulation run.
- With our tool, inputs can **dynamically varied** during a single simulation run, to improve coverage
- Create **dependency flow graphs**
  - to explore efficiently the space of the **interdependent variables**
  - determine internal variables which influence the variables **involved in the property** (transformed then into inputs)

Dependency flow graphs of HVAC model

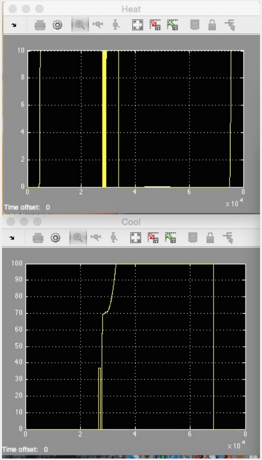


Choice of Inputs and Variations

- **Chosen inputs** (with strong effects on the system behaviours) are: outer air temperature, humidity (OAT, OAH) and CO_2. The other inputs remain fixed. The variation ranges are: $OAH \in [50, 100]$, $OAT \in [10, 45]$, $CO_2 \in [600, 1500]$.
- **Chosen internal variables** (reflecting possible external perturbations or fault injection): variables T_m (Room Air Temperature) and HR_m (Room Humidity Ratio). Variation ranges are: $T_m \in [15, 30]$, $HR_m \in [50, 100]$.

Testing results (1)

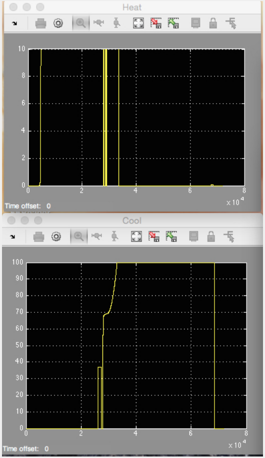
A property violation scenario: the initial zone temperatures in the integrator block are set to 20 (instead of 22 of the nominal regime), and $OAT \in [10, 45]$.



Temporal evolutions of the two valve outputs

Testing results (2)

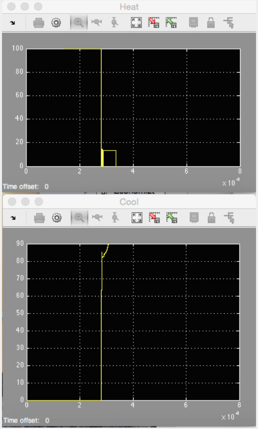
A property violation scenario: the initial zone temperatures in the integrator block are selected in $[19, 22]$, and OAT is fixed at 22 as in the nominal regime.



Temporal evolutions of the two valve outputs

Testing results (3)

A property violation scenario: a noise of 10% is added to the variable T_m



Temporal evolutions of the two valve outputs

The outcome of this example

- Demonstrated the scalability of the methodology and the tool
- Identified difficulties in applying a semi-formal validation technique to a real-life industrial model
- Experimental results showed potential significant improvements in design correctness (by detecting bugs at design time) and in design time reduction (automated test generation)

Plan

1 Validation of Cyber-Physical Systems

2 Coverage-Guided Testing Generation

3 Tool Implementation and Execution

4 Application to the HVAC

5 Future Work

Directions

- More general properties (beyond safety), specified by Signal Temporal Logic STL
- Biasing the exploration using a high-level abstraction of the model
- Conditioning the test generation process using the knowledge of the designer, in order to focus on some specific system behaviors.