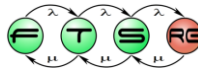


# Testing the robustness of communicating autonomous systems

Software Verification Techniques course

István Majzik  
Budapest University of Technology and Economics  
Dept. of Measurement and Information Systems



Budapest University of Technology and Economics  
Department of Measurement and Information Systems



## Outline

- The testing approach
- The role of models
- The testing strategy
  - Types of test sequences
  - View of robustness testing in the state space
- Modelling conventions
- Test sequence generation



## The testing approach

- **Model based testing**
  - **Model** = **specification of interactions** (cooperation) of autonomous robots (components)
- **Overall test objective:**
  - Checking whether the **implementation** of the tested robots (IUT: implementation under test) conforms to the **model**
    - Exercising the IUT by executing and checking interactions generated on the basis of the model
    - Test case = sequence of interactions relevant for the IUT
    - Test suite = test cases that satisfy given coverage criteria
- **Test environment: Test harness** (incl. test execution tool)
  - Replaces the environment of the IUT (its interacting partners and external environment)
  - Executes the interactions from the generated test cases that are relevant for the IUT



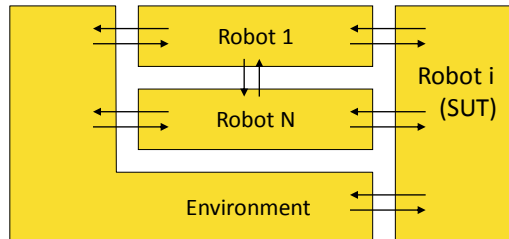
## The role of models

- **Closed (live) model** of a system and its environment
  - Interacting **robots** (components)
    - One or more of them is selected as subsystem under test (its model)
  - **Environment** (external, non-deterministic)
  - **Configuration** (parameters of the robots)
- **Modular modelling approach**
  - **Modules:** Robots, and the environment
  - **Interfaces:** Interaction points (typically „ports“)
- **Model based test generation**
  - Test case: **Trajectory in the state space** of the model
  - To execute the test: the interactions that are relevant for the IUT are selected and **performed on its ports**

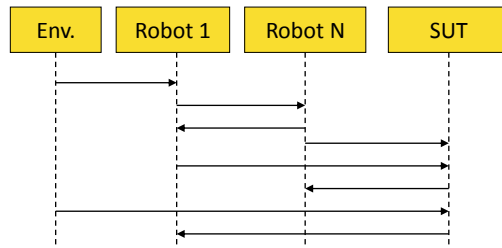


## The structure of models

- Structure of the model:



- Test case:  
Generated sequence of interactions:

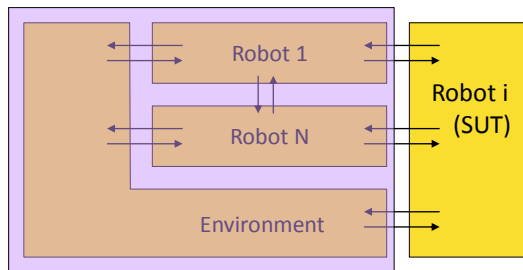


5

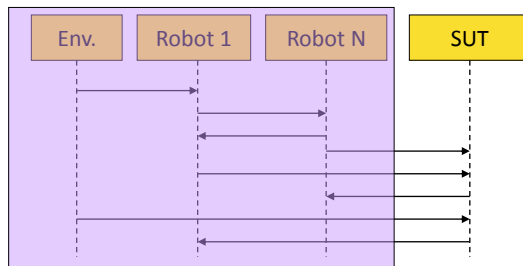


## The configuration of the test environment

- Test harness:
  - Here: for testing robot i only; subsystems consisting of more robots can also be tested



- Test case executed by the test harness:



6



## Testing strategy

- Goal: **Robustness testing** with respect to **extreme cases** in cooperation scenarios
- Testing strategy: Each test sequence consists of two parts:
  1. Driving the SUT to states in which it is able to **receive message**
    - through normal states,
    - through stressful states,
    - through boundary conditions,
    - using boundary messages
  2. **Sending invalid messages** and observing the reactions of the SUT
    - violating the message structure
    - applying extreme values
    - using invalid timing
- Coverage metrics: relevant states, conditions, messages
- Automated tool: Test sequence generation on the basis of **Coloured Petri Net model** of communicating systems
  - Challenge: Reachability of the specified state(s) of the SUT

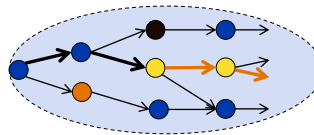
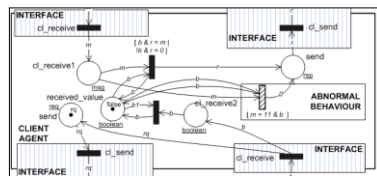


7



## Test sequences

Generated test sequences: Sequences of **messages needed to follow specific trajectories** in the state space of the SUT



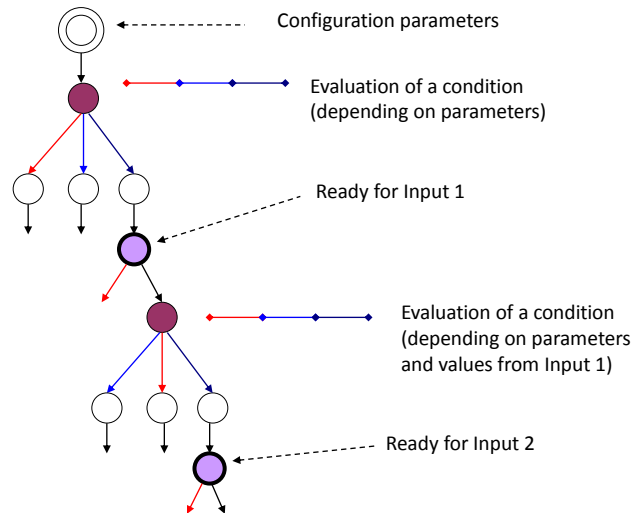
- Sending invalid input to the SUT in a **normal state** in which the SUT is ready to receive a message
- Sending invalid input to the SUT after reaching **stressful state**
- Sending invalid input to the SUT after **boundary conditions**
- Sending invalid input to the SUT after **boundary messages**
- Forcing the SUT to a specified **stressful state**
- Forcing the SUT to a specified **abnormal behaviour**



8



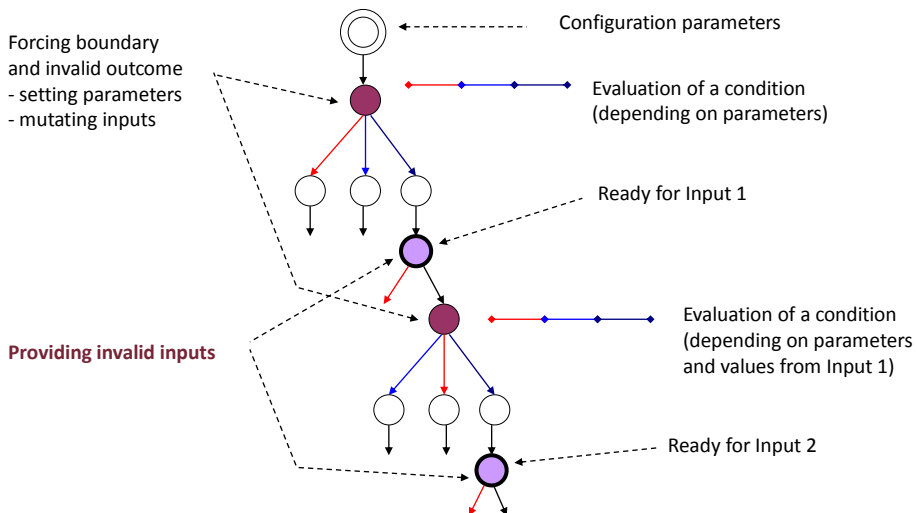
## View of robustness testing in the state space



11



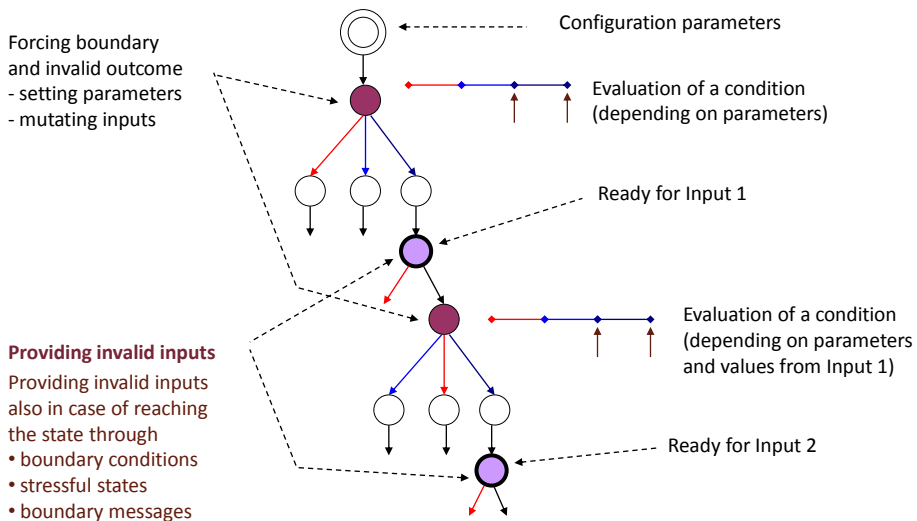
## View of robustness testing in the state space



12



## View of robustness testing in the state space



13



## Modelling conventions

- Identification of the **modules** that model the IUT
- Identification of **interfaces** of the IUT
  - Relevant state for interaction (when the IUT is able to receive input)
    - To drive the IUT to this state
  - Type of input (signal, message, call etc.)
    - To derive valid / invalid inputs
- Identification of **states** when conditions are evaluated (that process inputs from interactions)
  - Relevant state in which the condition is evaluated
    - To drive the IUT to this state
  - **Normal behaviour** determined by the decision
    - To generate inputs for **boundary condition**
    - To generate inputs to **violate the condition** (implicit abnormal behaviour)
  - **Explicit abnormal behaviour** determined by the decision
    - To generate inputs that check this behaviour

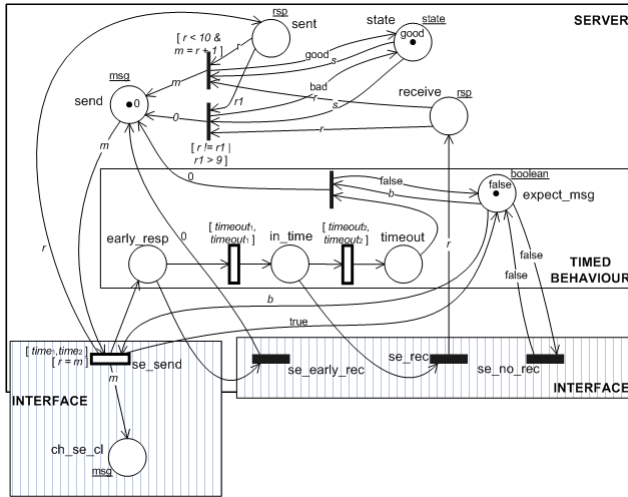


14



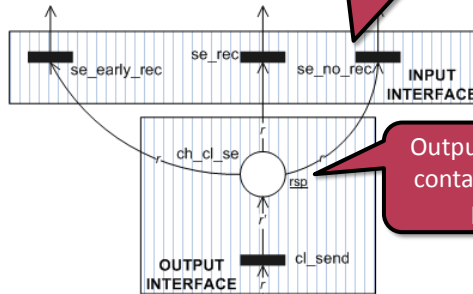
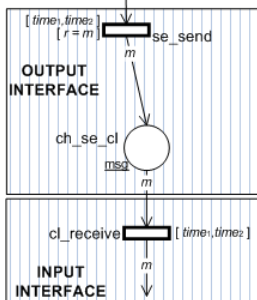
# Modelling conventions

- Modular (and hierarchical) models

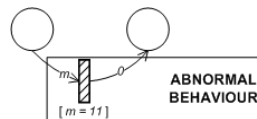


# Modelling conventions

- Interface definitions
  - Output interface
  - Input interface

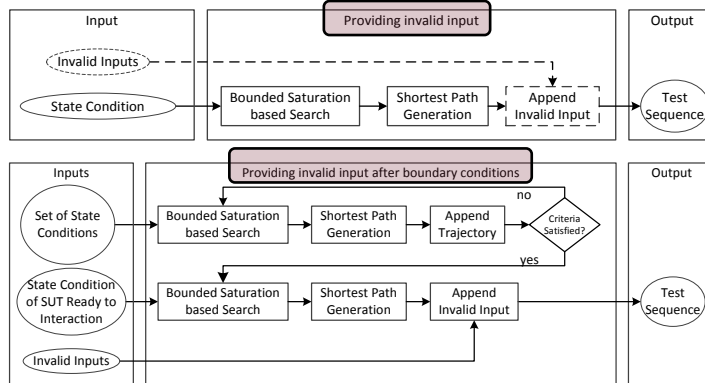


- Abnormal behaviour
- Stressful state



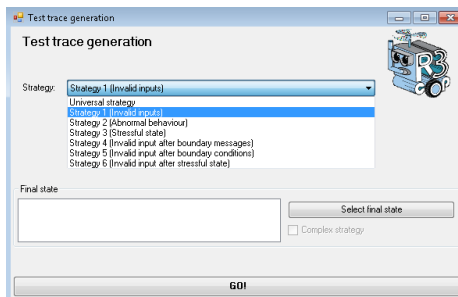
## Algorithmic background

- Goal: **Generating trajectory** to reach specified states
- Symbolic algorithms
  - Saturation based state space exploration (search in the state space)
  - Saturation based shortest path generation
- Combination of approaches for different test strategies

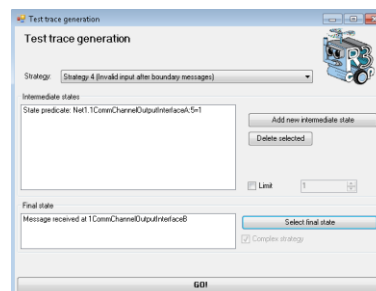
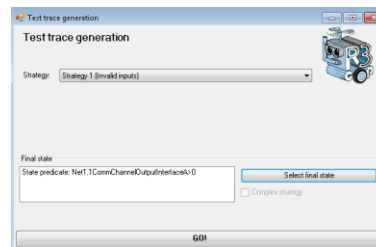
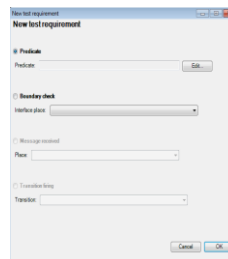


17

## Test generator tool



1. Selecting the strategy
2. Setting the parameters (state conditions of final and intermediate states)



18



# Example: Test case generation

**SMART Traffic Manager**

**Electric80 Centralized LGV Traffic management**

**LGV 1**      **LGV 2**      ...      **LGV n**

```

sequenceDiagram
    participant SMART
    participant LGV
    SMART->>LGV: Command
    LGV-->>SMART: Status response
    SMART->>LGV: Command response
    LGV-->>SMART: Status request
    
```

Here: Testing reaction to invalid inputs after boundary conditions of normal behaviour

```

graph TD
    A[Initial state] --> B[SMART: Set parameter of command message]
    B --> C[SMART: Send message]
    C --> D[LGV: Receive message]
    D --> E[LGV: Send acknowledgement]
    E --> F[SMART: Receive acknowledgement]
    F --> G[SMART: Set new mission]
    G --> H[SMART: Set parameter of command message]
    H --> I[SMART: Send message]
    I --> J[LGV: Receive message]
    J --> K[LGV: Set mission]
    K --> L[LGV: Send acknowledgement]
    L --> M[SMART: Receive acknowledgement]
    M --> N[SMART: Send message]
    
```

```

Test trace generation results

Test requirement
Net1.1.ApplyPosition:2,1,Net1.1.ReceiveCommand:2,1,Net1.1.CommChannelOutputInterface:

Test input sequence
1SetMission (p=0, p1=1)
1SetMission (p=1, p1=2)
1SetMission (p=2, p1=3)
1DecideToSend (p=3, c=0)
1SendPosCommand (p=3, c=0, command=pos3, c*=1)
1CommChannelInputInterface (command=pos3)
1CommandInputInterface (command=pos3)
1PositionInputMessage (command=pos3, p=3)
1UpdatePosition (p=3, p_old=0, opr=working)
1RespondAck ()
1CommRespOutputInterfaceAResp (cr=ack)
1CommChannelInputInterfaceAResp (cr=ack)
1CommRespInputInterface (cr=ack, c=1)
1MissionReallocation (p=3, c=0)
1SetMission (p=3, p1=0)
1SetMission (p=0, p1=1)
1SetMission (p=1, p1=2)
1DecideToSend (p=2, c=0)
1SendPosCommand (p=2, c=0, command=pos2, c*=1)
1CommChannelInputInterface (command=pos2)
1CommandInputInterface (command=pos2)
1PositionInputMessage (command=pos2, p=2)
--> State predicate: Net1.1.TargetPosition:3=1 &
Net1.1.PositionCommand:2=1 is true here.
1UpdatePosition (p=2, p_old=3, opr=working)
1RespondAck ()
1CommRespOutputInterfaceAResp (cr=ack)
1CommChannelInputInterfaceAResp (cr=ack)
1CommRespInputInterface (cr=ack, c=1)
1SendPosCommand (p=2, c=0, command=pos2, c*=1)
1CommChannelInputInterface (command=pos2)
--> Message received at 1CommChannelOutputInterfaceA is true here.
    
```