Testing Automation Systems by Means of Model Checkings

Igor Buzhinsky, Valeriy Vyatkin

igor.buzhinskii@aalto.fi

Aalto University
School of Electrical Engineering

ITMO UNIVERSITY

LULEÅ UNIVERSITY OF TECHNOLOGY

ETFA 2017, Limassol, Cyprus

September 14, 2017
Industrial automation systems require a high level of reliability

**Testing:** fast, popular in industry, not exhaustive

**Formal verification:** often slow and requires specific expertise, thus unpopular in industry, but more reliable

**Model checking:** formal verification technique based on exhaustive state space exploration

This work searches for synergies between testing and model checking by viewing test case execution as a model checking run
Model checking

- Requires a **formal model** of the system
- Mathematically, this model is often represented as a Kripke structure, a variant of a nondeterministic state machine where states are labeled and transitions are not
- However, there are formal languages to specify such models in a convenient way
- In particular, verifiers **NuSMV** and **SPIN** process textually specified models

- Similarly, **temporal logics** are formal languages which specify properties to be verified
- These properties often examine **infinite model behaviors**
Linear temporal logic (LTL)

- Formal language which extends the usual propositional Boolean logic
- Variables: *atomic propositions*, e.g. $p$ and $q$
- Usual Boolean operators are allowed, e.g. $p \rightarrow q$ is an LTL formula, but it refers to the **first element** of an infinite sequence

**Temporal operators**, such as:

- **G**: globally (always), e.g. $G(p \rightarrow q)$ means “in each element of the sequence, $p \rightarrow q$ holds”
- **F**: in the future, e.g. $F(p \rightarrow q)$ means “for some element of the sequence, $p \rightarrow q$ holds”
- **X**: on the next step, e.g. $X(p \rightarrow q)$ means “$p \rightarrow q$ holds for the second element of the sequence”
Why what is proposed is proposed

- Represent the testing process as model checking and examine the consequences

**Why?**

1. More convenient analysis of errors: a test case is failed, model checker will return a **counterexample**, which can be examined in detail without a need of logging
2. Time is modeled formally, and thus any **delays** in the tested system are omitted
3. If **multiple controllers** are specified in a single model, they can be checked **simultaneously**, which is potentially faster
4. In addition to test case success criteria, additional **temporal properties** (specifications) can be formulated independently from test cases and checked on them
What is proposed, in short

- Obtain the formal model of the tested system (controller) – preferably, using automatic approaches, e.g. by translating PLC code to NuSMV or SPIN (e.g. see works by D. Darvas et al.)
- Optionally, either manually or automatically, obtain a (nondeterministic) model of the plant
- Obtain test cases (e.g. using model-based testing, or MBT) and convert them to deterministic formal models
- Test success criteria can be formulated in temporal logics – a particular state of the test case must be reached
- Optionally, formulate additional temporal specifications
- Run model checking and examine counterexamples (i.e. error traces)
Entities which are modeled formally

- Test suite
  - Test case 1
  - Test case 2
- Controller under Test
- Plant
- Specification

- Solid lines: mandatory within the proposed framework
- Dashed lines: optional
Traffic lights case study: overview

- Fictitious industrial system
- A traffic lights device controls traffic in both directions on a crossroads
- 12 (6 pairs) lights in total
- The device is controlled by a PLC, which, in addition to automatic control, also has a manual mode
Traffic lights case study: PLC implementation (CODESYS)
Abstract specification as a state machine

- Two traffic lights operate in rounds of color changes
- Color durations are influenced by traffic intensities
- Model-based testing: test cases were generated automatically using the `graphwalker` tool
- In addition, long test cases (250 rounds / 12500 PLC cycles / 208 minutes) were prepared
How to represent test cases?

- Our previous work: input-output sequences modeled in net condition/event systems (NCES)
- Input-output sequences seem natural for test cases obtained from simulations or counterexamples generated by a model checker
- However, such sequences require **concrete outputs** from the controller on each step
- Thus, partial specifications, which permit various outputs, become problematic to check

- Solution: **deterministic state machines**
- In each state of the test case, multiple outputs are potentially possible, and they may lead to different states of the test case
- An accepting state must be reached to consider the test case passed
Test cases are represented as Moore state machines

- Inputs for the controller \((i_1, i_2)\) are presented in states
- Transitions are guarded by controller outputs \((o_1, o_2)\)
- In model checking: \(F_{s_a}\) (LTL) or \(AF_{s_a}\) (CTL) check whether the test case is passed
- \(s_f\) is provided for convenience and indicates a “test failure” state from which the accepting state \(s_a\) is unreachable
Abstract test cases were concretized and implemented in NuSMV

MODULE TEST_SUITE(expectedT1red, expectedT1yellow, expectedT1green, expectedT2red, expectedT2yellow, expectedT2green)
VAR
    state: {pass, fail, green_red, red_red1, red_green, man, green_red2};
    delay: 0..63;
    actual_delay1: 0..63;
ASSIGN
    init(state) := green_red;
    next(state) := case
        state in {pass, fail}: state;
        state = green_red & _T1_GREEN & _T2_RED : state;
        state = green_red & _T1_YELLOW & _T2_RED : red_red1;
        state = red_red1 : !( _T1_RED & _T2_RED ) ? fail : red_green;
        state = red_green & _T1_RED & _T2_GREEN : state;
        state = red_green & _T1_RED & _T2_YELLOW : DELAY_CONDITION ? fail : man;
        state = man : !MAN_MATCH ? fail : green_red2;
        state = green_red2 : !( _T1_GREEN & _T2_RED ) ? fail : pass;
    esac;
    init(delay) := 0;
    next(delay) := case
        state = green_red & _T1_GREEN & _T2_RED & delay < 63 : delay + 1;
        state = green_red & _T1_YELLOW & _T2_RED : 0;
... and in SPIN (Promela)

do
:: c_plant ? dummy_var;
    check_specs = false;

    mtype old_state = state;
    if
        :: state == pass || state == fail
        -> ;
        :: state == green_red && _T1_GREEN && _T2_RED
        -> delay++;
        :: state == green_red && _T1_YELLOW && _T2_RED
        -> actual_delay1 = delay; delay = 0;
        state = red_red1;
        :: state == red_red1
        -> state = (!(_T1_RED && _T2_RED)) ->
        fail : red_green);
        :: state == red_green && _T1_RED && _T2_GREEN
        -> delay++;
        :: state == red_green && _T1_RED && _T2_YELLOW
        -> actual_delay2 = delay; delay = 0;
        state = (DELAY_CONDITION -> fail : man);
        :: state == man
        -> state = (!MAN_MATCH -> fail :
        green_red2);
        :: state == green_red2
        -> fail : pass);
    :: else
        -> state = fail;
    fi

    DEFINES

c_controller ! false;
od
Experiments: test case execution

- Total number of test cases: 24
- 20 short test cases (1–2 color change rounds) generated using the finite-state model of requirements
- 4 manually specified long test cases (250 rounds = 12500 PLC cycles)

- SPIN: 1 s for short tests, 2.6 s for long tests – quite fast
- These results are explained by the fact that the state space is just a path for deterministic models
- In conventional testing, executing a long test case would take 208 minutes due to delays

- NuSMV: the median execution time was 18.4 minutes, 8 tests did not finish in one hour
- Symbolic model checking does not benefit from determinism!
Other experiments: checking temporal requirements on test cases

- In addition to the default temporal requirement “the test case reaches its accepting state” ($F_{s_a}$), more can be formulated.
- E.g. that only one color is on.
- E.g. that in the manual mode traffic lights show manually specified colors.
- Up to 4 seconds in SPIN to check each such property.
- Minutes or hours in NuSMV – again, since symbolic model checking does not benefit from determinism.

- Note that runtime verification offers similar opportunities, but some temporal properties (e.g. liveness ones) cannot be checked since verification is performed not for formal models.
Other experiments: test case validation

- Checking whether test cases comply with the plant model, i.e. specify permitted plant behaviors
- Implemented only in NuSMV – takes minutes or hours
- Potentially possible to implement in SPIN as a much faster process
Other experiments: simultaneous testing of multiple controllers

- Suppose that multiple controllers which are similar to each other need to be checked – e.g. a product line
- To test them simultaneously, they need to be represented as a single formal model
- In our case, we allowed some controller parameters (concrete delay values) to range over certain intervals
- Implemented in NuSMV – takes minutes or hours
- Can be implemented in SPIN, but testing will just become slower proportionally to the number of distinct controllers to be checked
Future work

- To make the proposed ideas applicable in industry, a user-friendly tool must be developed.

- The approach can be extended to support the UPPAAL model checker for timed automata.

- Ongoing project: test case synthesis by means of model checking and subsequent execution of these test cases.
Conclusions

- Model checking can be used for the purpose of testing and, surprisingly, it can be quite fast (SPIN: 12500 PLC cycles in 2.6 seconds)

- Testing with NuSMV (symbolic model checker) is obviously slower, but the effect of test case length and the number of tested controllers on the execution time is small

- Several benefits of testing by means of model checking have been shown
Acknowledgments

- Engineering Rulez project funded by Tekes – the Finnish Funding Agency for Innovation (Decision 3095/31/2016)

- Supported by the Ministry of Education and Science of the Russian Federation, project RFMEFI58716X0032