Explicit-State and Symbolic Model Checking of Nuclear I&C Systems: A Comparison

Igor Buzhinsky, Antti Pakonen, Valeriy Vyatkin

igor.buzhinskii@aalto.fi

IECON 2017, Beijing, China

October 31, 2017
Industrial automation systems require a high level of reliability

**Model checking** is a formal verification technique which exhaustively explores the state space of the system’s model

**Open-loop** (controller-only) and **closed-loop** (plant-and-controller) modeling and model checking

In Finnish nuclear industry, model checking is used to check correctness of nuclear I&C (instrumentation and control) systems

Previously, we worked with these systems using the framework based on symbolic model checker **NuSMV**

In this work we:

1. Propose a way to efficiently model I&C systems using Promela, the language of explicit-state model checker **SPIN**

2. Explore when model checking in **SPIN** is beneficial
Nuclear I&C applications: basic blocks

- A typical I&C system is a system of interconnected basic blocks, where each basic block has I/O variables, and (optionally) state.
- Example of a basic block modeled in NuSMV:

```plaintext
MODULE UnitDelay(INPUT) -- input variable 'INPUT'
VAR
    last : boolean; -- state variable 'last'
DEFINE
    OUTPUT := last; -- output variables 'OUTPUT'
ASSIGN
    init(last) := FALSE; -- internal logic
    next(last) := INPUT; -- internal logic
```
Nuclear I&C applications: block networks

- PlusMinus example: returns either the sum or the difference of arguments depending on the value of a Boolean switch.
The same block network in NuSMV (automatically generated using MODCHK tool)

MODULE PlusMinus(IN0, IN0_FAULT, IN0_CONNECTED, IN1, IN1_FAULT, IN1_CONNECTED, SWITCH, SWITCH_FAULT, SWITCH_CONNECTED)

VAR

    SW : BINARY_SWITCH(SWITCH, SWITCH_FAULT, TRUE, PLUS.ADDER_OUTPUT_SIGN, PLUS.ADDER_OUTPUT_SIGN_FAULT, TRUE, MINUS.ADDER_OUTPUT_SIGN, MINUS.ADDER_OUTPUT_SIGN_FAULT, TRUE, FALSE);

PLUS : ADDER_3(0, FALSE, FALSE, IN0, IN0_FAULT, TRUE, IN1, IN1_FAULT, TRUE, 1, 1, 1);

MINUS : ADDER_3(0, FALSE, FALSE, IN0, IN0_FAULT, TRUE, IN1, IN1_FAULT, TRUE, 1, 1, -1);

DEFINE

    OUT := SW.BSWITCH_OUTP_SIGN;
    OUT_FAULT := SW.BSWITCH_OUTP_SIGN_FAULT;

How to represent the same in Promela (SPIN)? And why do we need this? Let’s see
Model checking

- Requires a **formal model** of the system
- Formal languages allow specifying such models in a convenient way
- In particular, model checkers **NuSMV** and **SPIN** process textually specified models

- Similarly, **temporal logics** (e.g. LTL, CTL) are formal languages which specify properties to be verified
- These properties often examine infinite model behaviors
- Example of an LTL property: $\mathbf{G}(p \rightarrow \mathbf{F} q)$ – “always, if $p$ is true, eventually $q$ will be true”
Explicit-state and symbolic model checking

Explicit-state model checking
- Each state of the model’s state space is processed explicitly
- SPIN model checker
- Shortcoming: state space can easily become too large to be processed (state space explosion)

Symbolic model checking
- State subsets are encoded implicitly with Boolean formulas and processed using heuristics
- Often a solution to the state space explosion problem
- NuSMV model checker
Why do we need explicit-state model checking? Isn’t it a step backwards?

- Symbolic model checking can still be infeasible or slow for sufficiently complex models
- Also, its execution time is highly unpredictable

- In closed-loop model checking, the presence of the plant model reduces the state space of the system to be verified
- This has direct positive impact on computational complexity of explicit-state model checking
- As for symbolic model checking, it has not
- Instead, the overall model becomes more complex and hence model checking time increases
## Expectations

- **Assumption:** the number of states in the plant model is small (not more than several thousand)
- **In our study:** this is achieved by constructing plant models automatically from behavior traces

<table>
<thead>
<tr>
<th></th>
<th><strong>NuSMV</strong></th>
<th><strong>SPIN</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-loop model checking</td>
<td>Known to be usually fast (seconds)</td>
<td>Expected to be slow/infeasible due to state space explosion</td>
</tr>
<tr>
<td>Closed-loop model checking</td>
<td>Known to be slow (minutes, hours or infeasible)</td>
<td><strong>Expected to be faster than in NuSMV</strong></td>
</tr>
</tbody>
</table>

- **Justification:** SPIN will allow achieving the benefits of state space reduction
bool INPUT, OUTPUT, last = 0;

proctype UnitDelay(chan ret) {
    OUTPUT = last;
    last = INPUT;
    ret ! false;
}

- In SPIN, models are represented as sequentially executed code
- A process call can be used to execute a basic block
- But actually, instead of this technique a more efficient one is used
Basic blocks are modeled manually in SPIN, using their NuSMV implementations.

NuSMV block network models are parsed using AntLR.

Then, Promela (SPIN) models are generated.
The goal is to make Promela (SPIN) models equivalent to NuSMV ones and also fast to process. Thus, model execution is organized as an infinite loop, where first the plant executes, and then the controller.

- Trick 1: execution of basic block instances is inlined. Process calls are omitted. Thus, faster processing.
- Trick 2: consequent deterministic statements are grouped into `d_step` statements. This also speeds up processing.
- Trick 3: the body of the loop is grouped into an atomic statement. This makes intermediate states invisible in model checking.
How the resultant code looks

// <Variable declarations:
// - controller inputs and outputs
// - outputs of each basic block>

init { do :: atomic {
    // <Selection of controller inputs:
    // - Open-loop case: nondeterministic selection
    // - Closed-loop case: based on the plant model>
    d_step {
        // <Block network execution:
        // - in the order of a topological sort to resolve
        //   dependencies
        // - unit delays break cyclic dependencies>
    }
} od }
Case study: generic PWR model

Generic PWR model overview:

<table>
<thead>
<tr>
<th>Process networks</th>
<th>Automation networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary circuit</td>
<td>Reactor control</td>
</tr>
<tr>
<td>Pressure vessel</td>
<td>Plant and turbine power control</td>
</tr>
<tr>
<td>Emergency system</td>
<td>Reactor and turbine trip</td>
</tr>
<tr>
<td>Steam generators</td>
<td>Protection networks</td>
</tr>
<tr>
<td>Etc.</td>
<td>Etc.</td>
</tr>
</tbody>
</table>

- Model of a nuclear power plant (NPP) with a pressurized water reactor (PWR)
- From Fortum Power and Heat Oy (Finland)
- Implemented in Apros continuous process simulator
Example of an I&C block network of one of the subsystems

- This control subsystem is responsible for pressurizer pressure control
- Details are hidden due to non-disclosure reasons
- Plant model I/Os correspond to control subsystem I/Os
Experiment setup

- 8 I&C subsystems of the generic PWR model were selected

- Plant models were generated based on traces as explicit finite-state machines (the next presentation will explain this approach in detail)

- 3000 simulation traces were collected, each with 240 elements, corresponding to 4 simulation minutes

- Up to 4906 states in the largest plant model, but three of them have 100 states or less

- Then, model checking was run on equivalent NuSMV and SPIN models
Obtained model checking time in minutes

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td># temporal specs</td>
<td>9</td>
<td>24</td>
<td>26</td>
<td>15</td>
<td>10</td>
<td>18</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Open-loop</td>
<td>SPIN</td>
<td>54</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>TL</td>
<td>3</td>
</tr>
<tr>
<td>MC time (s)</td>
<td>NuSMV</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>1</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>Closed-loop</td>
<td>SPIN</td>
<td>3</td>
<td>44</td>
<td>277</td>
<td>98</td>
<td>256</td>
<td>148</td>
<td>3</td>
</tr>
<tr>
<td>MC time (s)</td>
<td>NuSMV</td>
<td>2611</td>
<td>137</td>
<td>769</td>
<td>TL</td>
<td>718</td>
<td>1104</td>
<td>268</td>
</tr>
</tbody>
</table>

- Time limit (TL) = 10 minutes × the number of temporal specifications
- Open-loop model checking is faster in NuSMV
- Closed-loop model checking is faster in SPIN
Conclusions

- Comparison of explicit-state and symbolic model checking techniques on the problem of nuclear I&C application verification
- To enable this comparison, an efficient approach to represent such systems in Promela (SPIN) has been developed
- Block networks are used beyond the nuclear domain, e.g. in IEC 61131 FBD, IEC 61499
- Particular attentions to the case of closed-loop verification

- Key finding: in closed-loop model checking, when the number of states of the plant model is limited to several thousand, explicit-state model checking outperforms symbolic one
Future work

- In model checking, consider temporal requirements which require the plant model to be present (such as: “maintain the particular plant parameter within a defined range”)

- Investigate whether the findings are the same for manually prepared plant models

- Apply closed-loop model checking in SPIN to verify models of both I&C software and hardware
Acknowledgments

- SAUNA project, funded by the Finnish Nuclear Waste Management Fund VYR as a part of research program SAFIR2018

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