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

Igor Buzhinsky, Antti Pakonen, Valeriy Vyatkin

igor.buzhinskii@aalto.fi

Aalto University
School of Electrical Engineering

ITMO UNIVERSITY

VTT

LULEÅ UNIVERSITY OF TECHNOLOGY

IECON 2017, Beijing, China

October 31, 2017

Introduction

- 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:

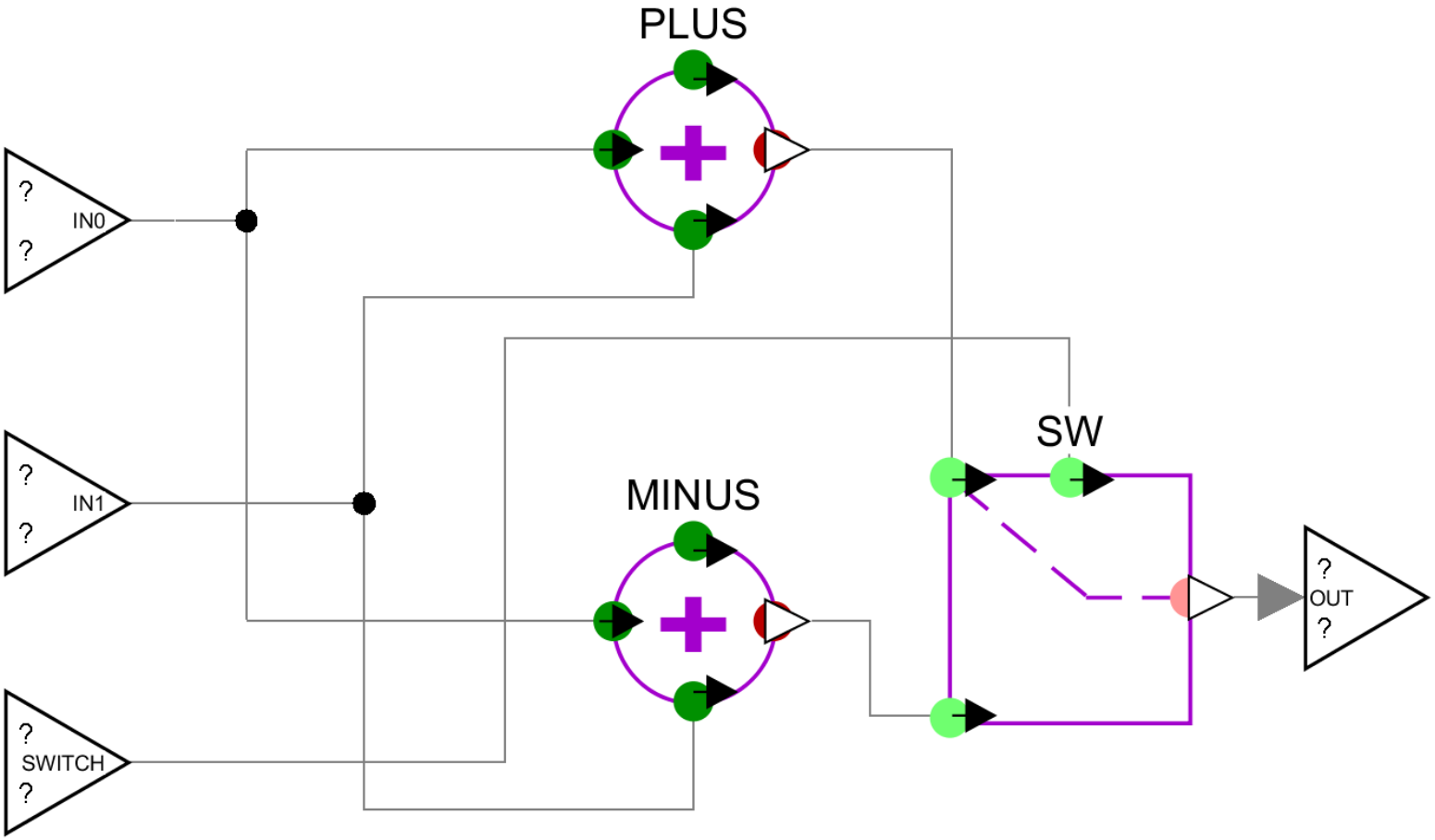
1. Propose a way to efficiently model I&C systems using Promela, the language of explicit-state model checker **SPIN**
2. Explore when model checking in **SPIN** is beneficial

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:

```
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

	NuSMV	SPIN
Open-loop model checking	Known to be usually fast (seconds)	Expected to be slow/infeasible due to state space explosion
Closed-loop model checking	Known to be slow (minutes, hours or infeasible)	**Expected to be faster than in NuSMV**

- Justification: SPIN will allow achieving the benefits of state space reduction

How to model basic blocks in SPIN

```
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

I&C system modeling approach for Promela (1)

- 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

I&C system modeling approach for Promela (2)

- 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 }
```

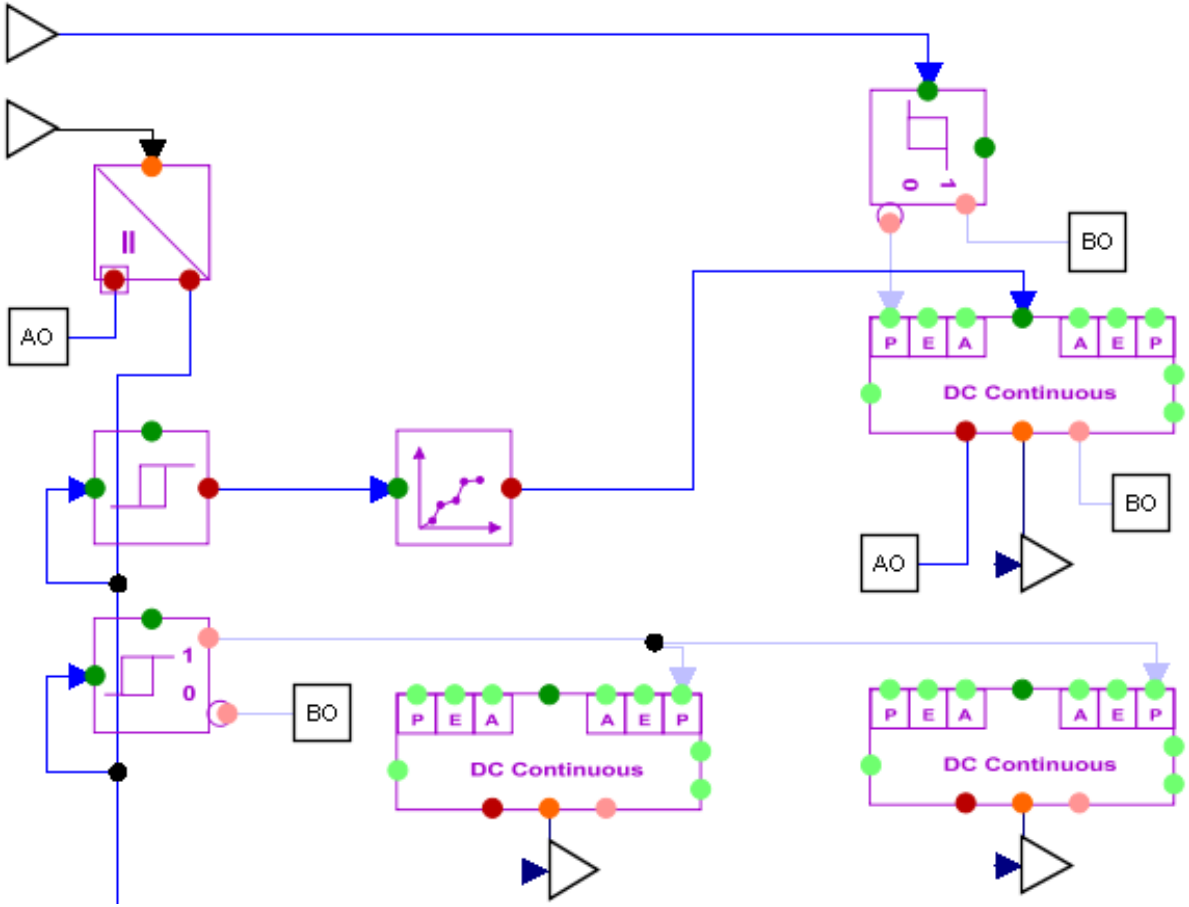
Case study: generic PWR model

Generic PWR model overview:

Process networks	Automation networks
Primary circuit	Reactor control
Pressure vessel	Plant and turbine power control
Emergency system	Reactor and turbine trip
Steam generators	Protection networks
Etc.	Etc.

- 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

Subsystem		S1	S2	S3	S4	S5	S6	S7	S8
# temporal specs		9	24	26	15	10	18	11	8
Open-loop	SPIN	54	TL	TL	TL	TL	TL	3	8
MC time (s)	NuSMV	5	1	11	11	1	21	1	2
Closed-loop	SPIN	3	44	277	98	256	148	3	3
MC time (s)	NuSMV	2611	137	769	TL	718	1104	268	8

- Time limit (TL) = 10 minutes $\times$ 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