SERC DOCTORAL STUDENT FORUM 2023 | NOVEMBER 14, 2023

Behavioral verification via Model-Checking: A Practical Approach for System Models

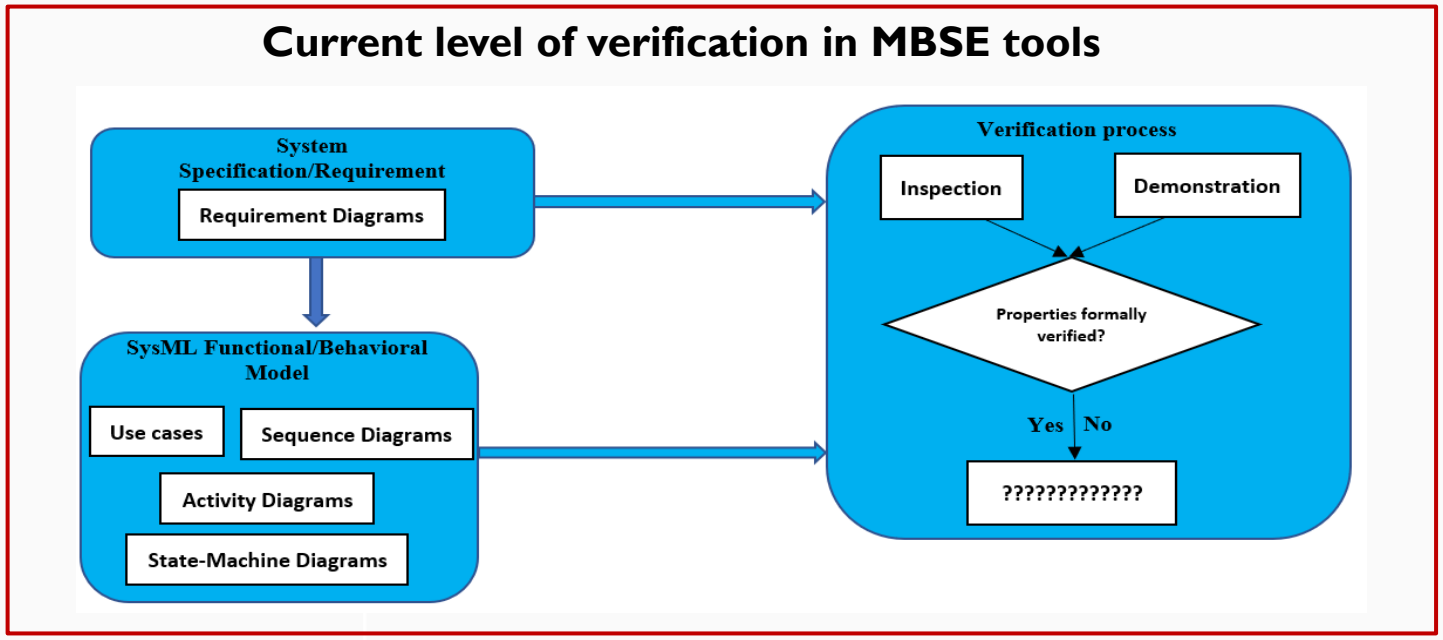
Ibukun Phillips, Purdue University

PhD Advisor: Dr. Robert Kenley

PURDUE UNIVERSITY

Background

- Prevalence of large, complex systems in current engineering applications and the digital transformation reality
  - More design opportunities for the application of Model-based Systems Engineering (MBSE) - a subset of Digital Engineering (DE).
- However, existing verification methods in MBSE have limited capability to provide system-level assertions since:
  - Common modeling languages (SysML & UML) utilized in MBSE tools are semi-formal modeling approaches and rely on mostly inspection and demonstration verification methods
  - Lack mathematically rigorous, high-level formal languages for rigorously evaluating these systems against specifications
  - Limited scalability to large, complex models

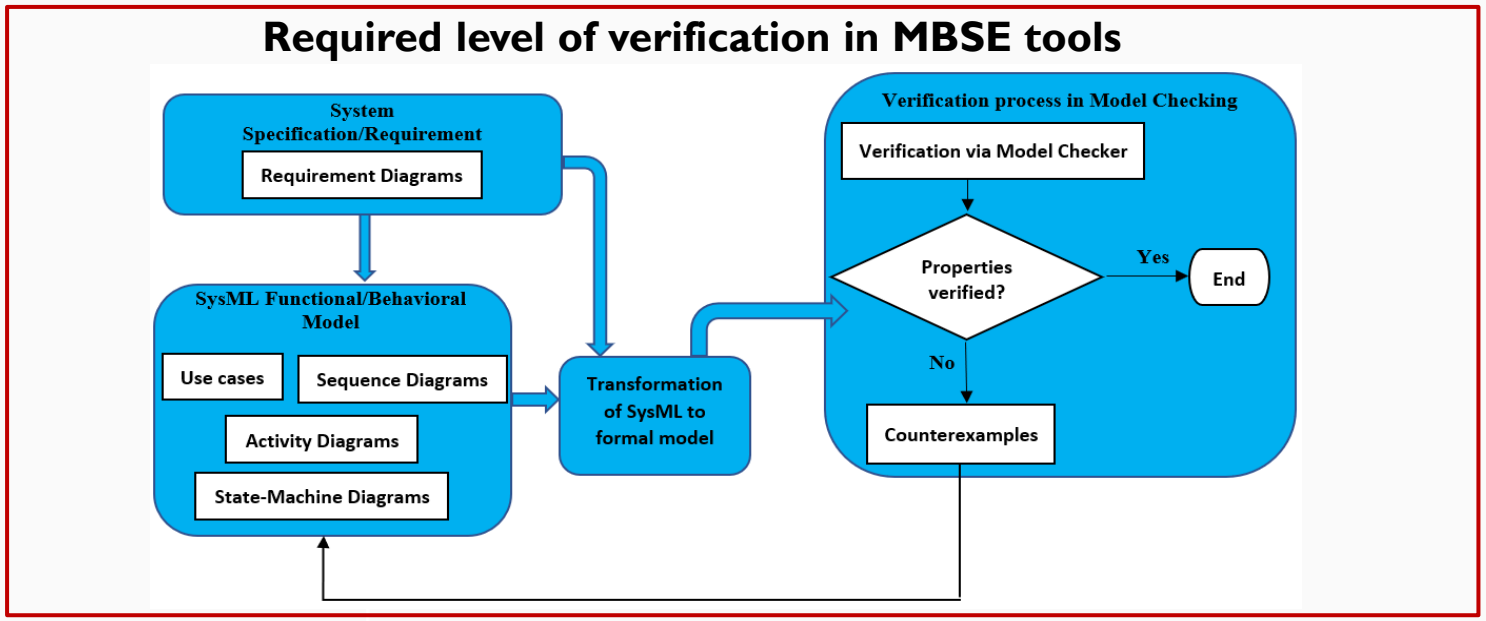


Research: Existing Gap and Desired Outcome

- There is a mismatch in Systems Engineering practice between:
  - Department of Defense (DoD)'s DE strategy of ensuring systems model correctness by being "*accurate, complete, and trusted…*", and:
  - Current MBSE activities using SysML models especially for ascertaining systems are being built right (verification) [Hecht & Chen, 2021: *Verification and Validation of SysML Models*]
- Hence, a more robust verification approach in MBSE is needed to ensure correctness of system models to attain the modeling goal of DoD's Digital Engineering strategy
  - Ensuring design assurance and reducing cost overruns and delays
- In addressing this gap, this research demonstrated a novel but pragmatic formal verification approach for MBSE behavioral models by:
  - Applying the model checking technique to verify an autonomous differential drive robot (DDR) modeled using as a state machine diagram against a set of requirements specified for the system
  - Mitigating attendant state-space explosion issue with the model checking method by abstraction on a safety relay system case study

Formal Methods: Model Checking

- Formal methods are:
  - mathematically rigorous techniques for analyzing and verifying system (hardware and software) models at any part of the system lifecycle
  - most common V&V (verification & validation) method in formal verification literature [Martínez-Fernández, 2022: *Software engineering for AI-based systems: A survey*]
- Formal verification is broadly classified into deductive verification and ***model checking*** [Seger, 1992: *Introduction to Formal Hardware Verification*]
- **Main Argument:** Formal methods + system models (SysML/UML) → Effective formal modeling and specification of systems to ensure the correctness of system model against requirements.



Model Checking: Temporal Logic Language (TLL)

- TLL is the verification language for the formalism for this transformation to be used by model checkers.
- Majorly classified into linear temporal logic (linear time structure) and Computational tree logic (branching time structure)
- Typical Linear temporal logic operators are:
  - `F p` (read “in the future `p`”), stating that a certain condition `p` holds in one of the future time instants;
  - `G p` (read “globally `p`”), stating that a certain condition `p` holds in all future time instants;
  - `p U q` (read “`p` until `q`”), stating that condition `p` holds until a state is reached where condition `q` holds;
  - `X p` (read “next `p`”), stating that condition `p` is true in the next state.

NuSMV Model Checker

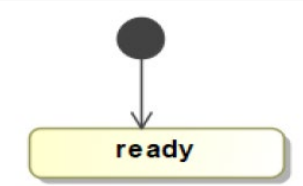
- New Symbolic Model Verifier (NuSMV)
  - used to analyze temporal logic specifications of various systems
  - abstract notation of Kripke structure
  - permits Linear Temporal Logic (LTL), and Computational Tree Logic (CTL)

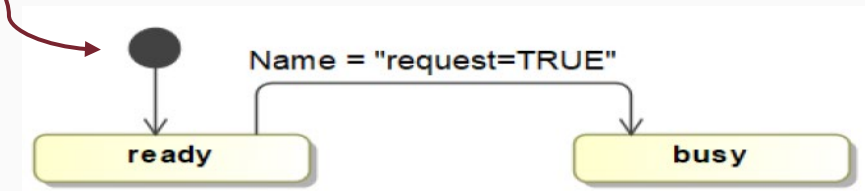
```
1 MODULE main
2
3    VAR
4    request : boolean; state : {ready,busy};
5
6    ASSIGN
7    init(state) := ready;
8    next(state) := case
9                   state = ready & request : busy;
10                                  TRUE : {ready,busy};
11                   esac;
```

variable declaration

variable initialization

variable transition

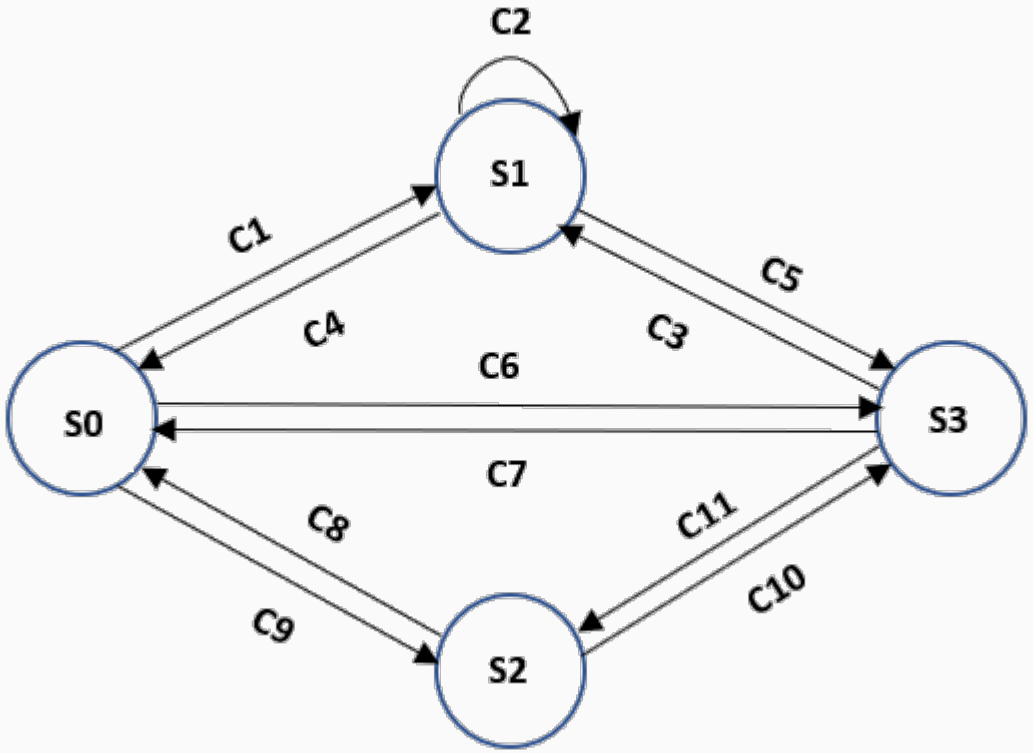




A simple NuSMV Module declaration

Kripke Structure in NuSMV from SysML Transformation

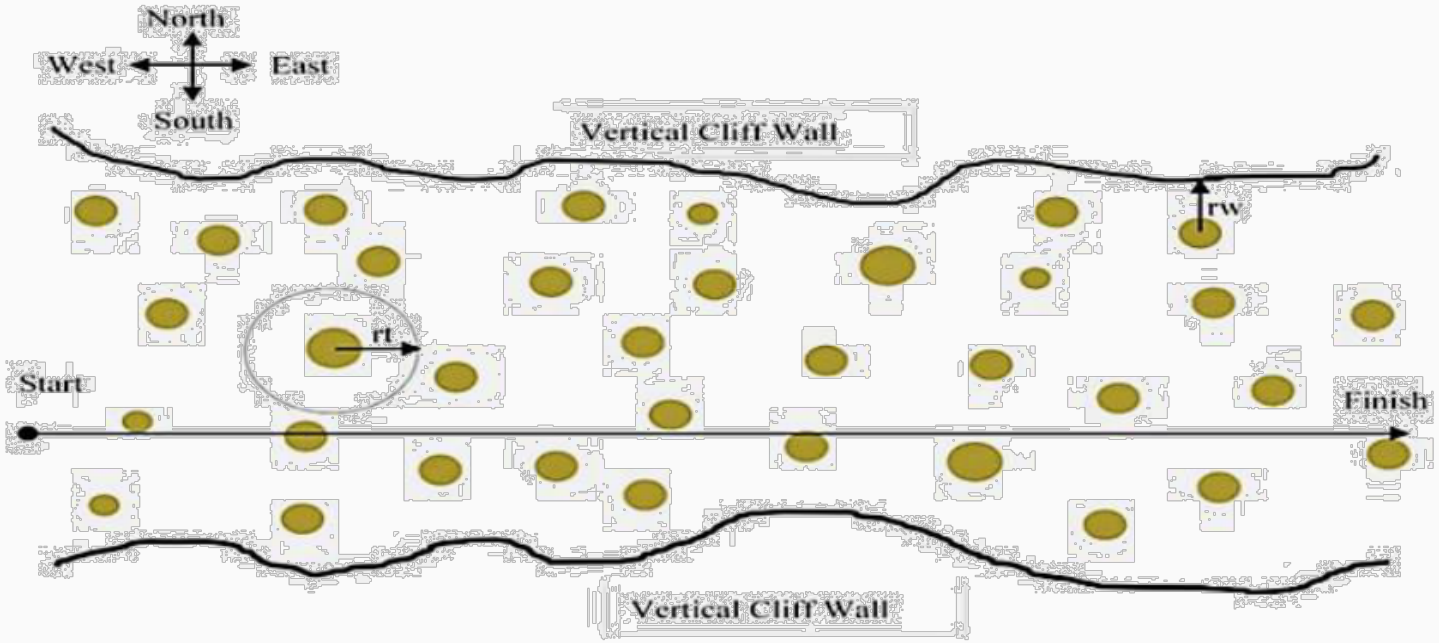
- Underlying abstraction of the model checking process based on the concept of Finite State Machines (FSMs)



- defined as a quadruple $M = (S, R, S_0, L)$ where S is the set of states with $S = \{S0, S1, S2, S3\}$
- $S_0 \subseteq S$ is the set of initial states
- $R \subseteq S \; X \; S$ is the transition relation with $C = \{C1, \ldots, C11\}$ being the transition set
- and $L{:}S \rightarrow P(A)$ is the labeling function, where A is the set of atomic propositions, and $P(A)$ is the powerset over A

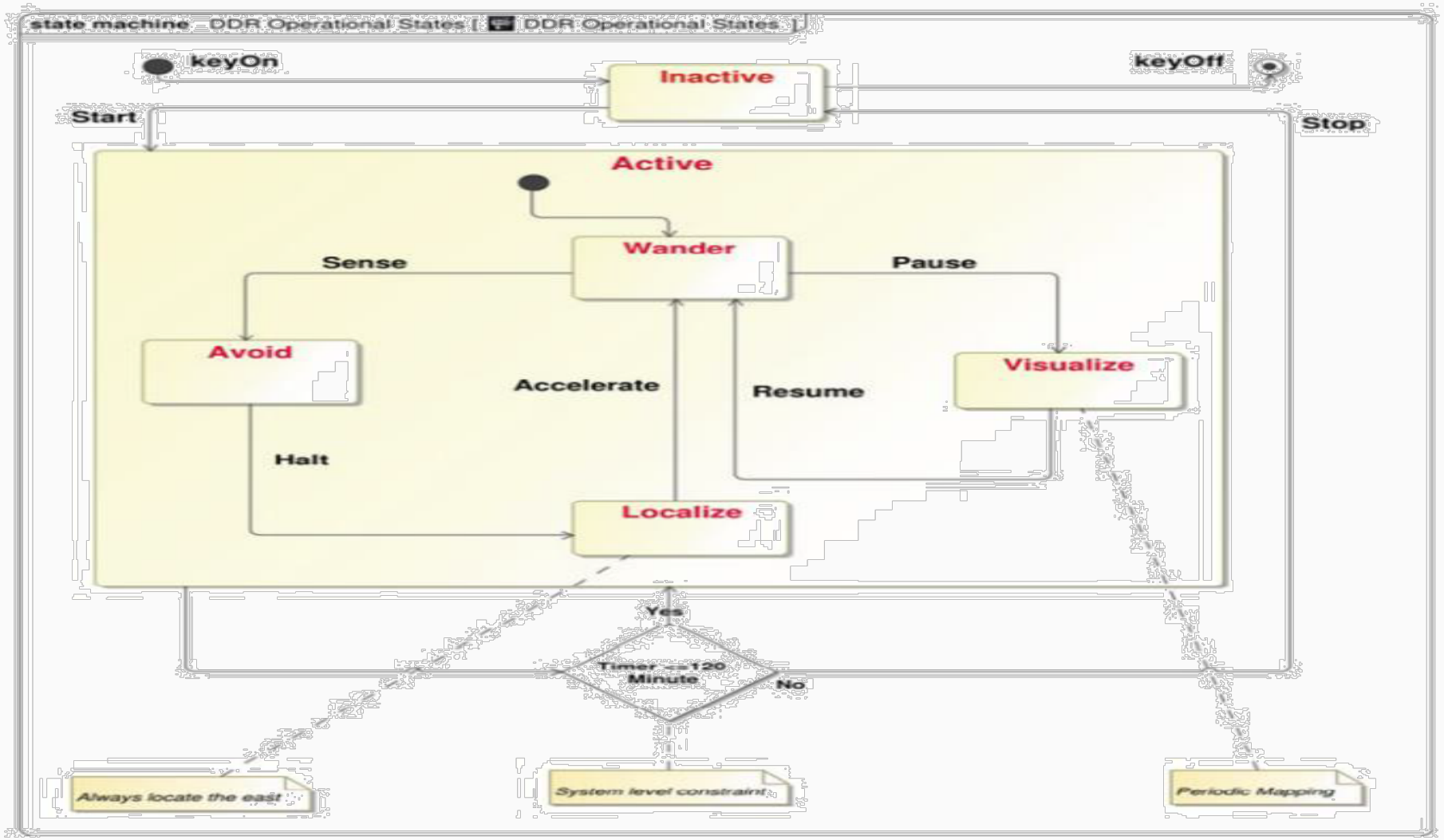
Case study: Autonomous multi differential drive robot (DDR)

- DDR system is a CPS of autonomous mobile robots (differential drive robots)
  - ❑ developed to identify and retrieve a target inside a forest with an unknown path plan
- Goal: utilize model checking to verify some system models already available for this DDR system that can be described in terms of the forest environment, operational specifications, and differential robot specifications.



System Environment of the DDR

Model Transformation: DDR Operational States

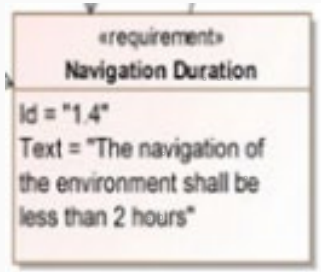


STM Diagram for the DDR System Operational States

Model Transformation: DDR Operational States

```
MODULE active
        VAR
                sense           : boolean;
                pause           : boolean;
                halt            : boolean;
                accelerate      : boolean;
                resume          : boolean;
                active_state    : {wonder, avoid, localize, visualize};
        ASSIGN
                init (active_state) := wander;
                next (active_state) := case
                                active_state = wander & sense                   : avoid;
                                active_state = wander & pause                   : visualize;
                                active_state = avoid & halt                     : localize;
                                active_state = localize & accelerate            : wander;
                                active_state = visualize & resume               : wander;
                                TRUE                                            : active_state;
                                esac;
MODULE main
        VAR
           keyOff: boolean;
           start: boolean;
           state: {inactive, active, off};
           timer: 1..120;
        ASSIGN
                init (state) := inactive;
                init (timer) := 1;
                next (state) := case
                                state = inactive & start                                : active;
                                (state = active) & (timer <= 120)                       : active;
                                (state = active) & (timer > 120)                        : inactive;
                                state = inactive & keyOff                               : off;
                                TRUE                                                    : state;
                                esac;
```

NuSMV model for the DDR Operational States system model



Requirement:

- LTLSPEC -p "G (state = active & timer <= 120)"

Verification result

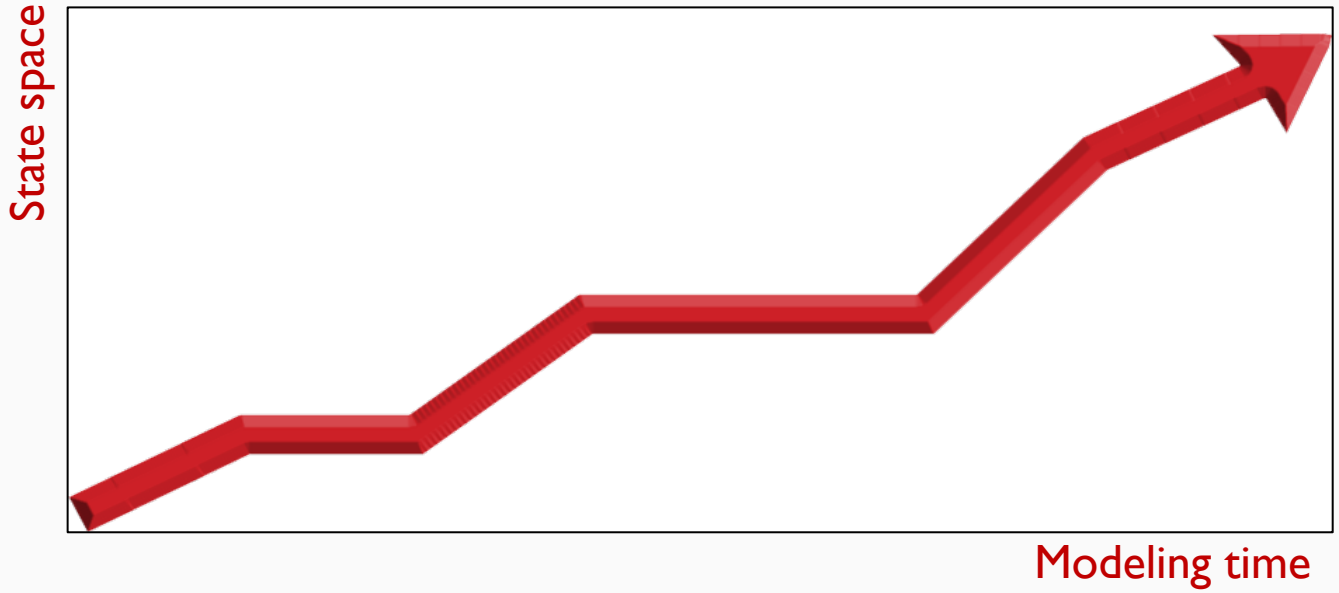
```
NuSMV > check_ltlspec -p "G(state=active & timer<=120)"
-- specification  G (state = active & timer <= 120)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
 -> State: 1.1 <-
   keyOff = FALSE
   start = TRUE
   state = inactive
   timer = 1
 -- Loop starts here
 -> State: 1.2 <-
   start = FALSE
   state = active
 -> State: 1.3 <-
```

NuSMV verification result for the DDR System

- Navigation duration specification is false, which means that the NuSMV system model does not satisfy the requirement.
- Conversely, a counterexample is provided to detail the execution sequence that breaks the LTL specification formula.
- An implication of the false specification for the DDR navigation requirement in State 1.1 is that the DDR could be in an inactive phase with the timer variable being simultaneously at most 120 minutes
- The model checking process helped uncover the unrealistic navigation requirement for the DDR

State Space Explosion: Mitigation through Abstraction

- What happens when SysML behavioral model experiences an exponential growth in the number of states in a system as the model size increases?



- Abstraction techniques:
  - **State-based reductions** e.g., symmetry reduction, live variable reduction, cone of influence reductions etc.
  - Path-based reductions e.g., transition merging, partial order reduction, τ-confluence
  - Compositional methods e.g., compositional generation of the state space and assume-guarantee approach

Cone of Influence Reduction

- The Cone of influence C of V' is the minimal set of variables such that
  - $V'(variables\ of\ interest\ with\ respect\ to\ specifications) \subseteq C$
  - If for some $v_i \in C$ its f_i (boolean function) depends on v_j, then $v_j \in C$.

Correctness Preservation

Let $V = \{v_1, \dots, v_n\}$ be a set of boolean variables and let $\boldsymbol{M(Original\ Model)} = (S, R, S_0, L)$ be the model of a synchronous circuit defined over V where,

$S = \{0,1\}^n$ is the set of all valuations of V ; $R = \bigwedge_{i=1}^{n}[v_i' = f_i(V)]$; $L(s) = \{v_i | s(v_i) = 1\ for\ 1 \le i \le n\}$

$S_0 \subseteq S$.

Suppose we reduce the circuit with respect to the cone of influence $C = \{v_1, \dots, v_k\}$ for some $k \le n$. The ***reduced model*** $\widetilde{\boldsymbol{M}} = (\tilde{S}, \tilde{R}, \widetilde{S_0}, \tilde{L})$ is defined by

$\tilde{S} = \{0,1\}^k$ is the set of all valuations of $C = \{v_1, \dots, v_k\}$; $\tilde{R} = \bigwedge_{i=1}^{n}[v_i' = f_i(V)]$; $\tilde{L}(\tilde{s}) = \{v_i | \tilde{s}(v_i) = 1\ for\ 1 \le i \le k\}$

$\widetilde{S_0} = \{(\widetilde{d_1}, \dots, \widetilde{d_k}) |$ there is a state $(d_1, \dots, d_n) \in S_0$ such that $\widetilde{d_1} = d_1 \wedge \dots \wedge \widetilde{d_k} = d_k\}$.

Conclusion

B is a bi-simulation relation between $\boldsymbol{M}$((original NuSMV system model consisting of variables influencing the requirement variables and redundant variables that don't) and $\widetilde{\boldsymbol{M}}$ (reduced NUSMV system model consisting of only variables that influence the requirement variables). Thus, $\boldsymbol{M} \equiv \widetilde{\boldsymbol{M}}$.

- Case study: Verify the behavioral system model of a safety relay system (electrical)
  - For the safety relay system, the number of state variables gets reduced from **24** to **6**

Cone of Influence Reduction

- Abstraction approach: aims to reduce the transition graph's size by focusing on the system variables indicated in the specifications

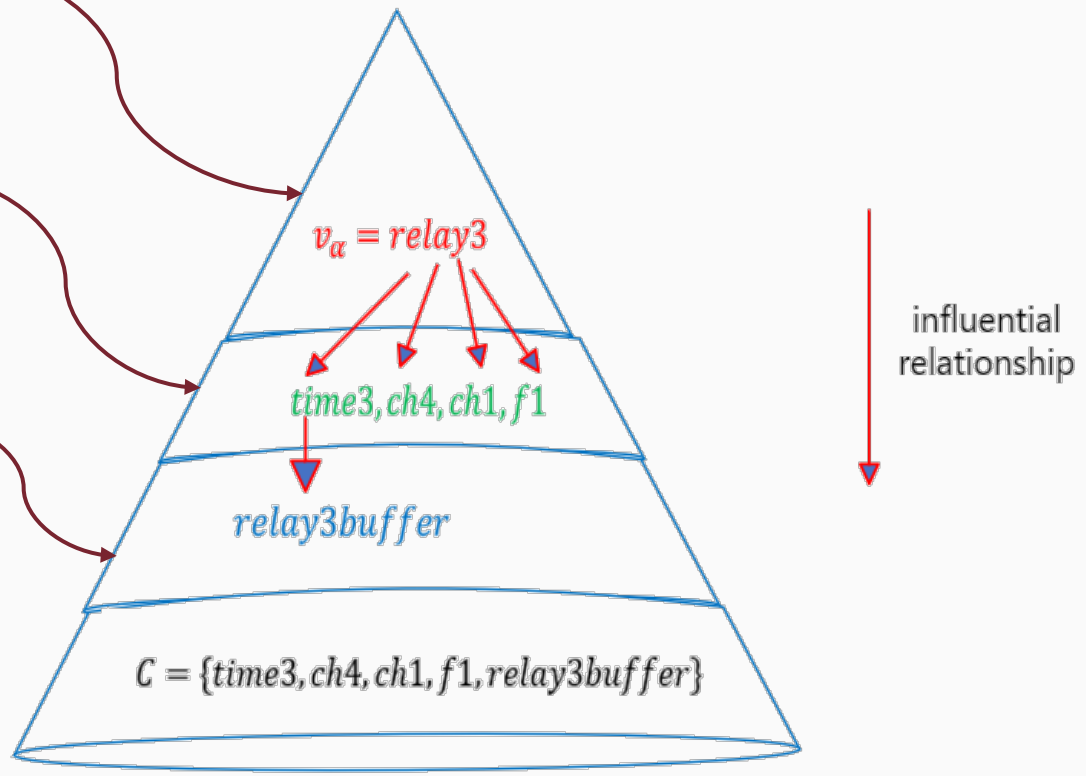
Requirement:

-LTLSPEC -p "G(relay3 = alarm)"

```
next (relay3) := case
((time3 = 30) & ((ch4 & f1) | (ch1 & ch4))) : alarm;
TRUE : OK;
esac;
```

```
next (time3) := case
(relay3buffer = alarm & time3 < 30): time3 + 1;
(relay3buffer = alarm & time3 = 30): 30;
TRUE : 0;
esac;
```

```
next(relay3buffer) := case
((ch4 & f1) | (ch1 & ch4)) : alarm;
TRUE : OK;
esac;
```



Verification Results

```
NuSMV > check_ltlspec -p "G(relay3=alarm)"
-- specification  G relay3 = alarm  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    ch1 = FALSE
    ch2 = FALSE
    ch3 = FALSE
    ch4 = FALSE
    f1 = FALSE
    ARCFail = FALSE
    triac1 = OK
    triac2 = OK
    triac3 = OK
    triac4 = OK
    relay1 = OK
    relay2 = OK
    relay3 = OK
    relay4 = OK
    relay5 = OK
    relay6 = OK
    relay1buffer = OK
    relay2buffer = OK
    relay3buffer = OK
    relay4buffer = OK
    time1 = 0
    time2 = 0
    time3 = 0
    time4 = 0
  -> State: 1.2 <-
```

Variables: 24

```
NuSMV > check_ltlspec -p "G(relay3=alarm)"
-- specification  G relay3 = alarm  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    ch1 = FALSE
    ch4 = FALSE
    f1 = FALSE
    relay3 = OK
    relay3buffer = OK
    time3 = 0
  -> State: 1.2 <-
```

Variables: 6

NuSMV verification results with counterexamples. Left (Original Model $\boldsymbol{M}$); Right (Reduced Model $\widetilde{\boldsymbol{M}}$)

Conclusion and Future Work

- A model checking approach ensures model correctness in MBSE towards DE goals:
  - Novel Applications:
    - An integrated approach of model checking and abstraction techniques for pragmatic verification of system models.
    - Showcases versatility of verification method with emergency surveillance and electrical component case studies.
  - Contributions
    - Demonstrates how model checking can be used in verifying behavioral system models in SysML, such as state machine diagrams.
    - Demonstrates a method for addressing an enduring limitation of model checking: state-space explosion.
    - Explores the formal mathematical representations of a SysML behavior diagram that lends it to formal expression in the verification language of a model checker.

- Future Work Opportunities:
  - Utilizing other behavioral SysML diagrams e.g. activity, use-case etc., as system models to be verified against requirements
  - Automation of the model checking and abstraction processes can potentially add formal verification to existing MBSE modeling tools' functionalities.
  - Mitigating state-space explosion by exploring techniques focused on giving up requirement on completeness and exploring only part of the state space.

Thank you

Stay connected with SERC Online:

Email the presenter: Ibukun Phillips

poluwase@purdue.edu

Email the advisor: Dr. Robert Kenley

kenley@purdue.edu