

Formal Analysis of a Railway Signaling Block designed in AIDA

Case Study

Roberto Cavada Alessandro Cimatti Alberto Griggio Christian Lidström **Gianluca Redondi** Matteo Tessi Dylan Trenti

¹Fondazione Bruno Kessler, Trento, Italy

²Reti Ferroviarie Italiane, Roma, Italy

RSSRail2025, Pisa



Case Study Context

- Application of formal methods to a component designed by RFI
- Focus on a **railway signaling component** within AIDA framework
- High-level comparison of two verification approaches

Case Study Context

- Application of formal methods to a component designed by RFI
- Focus on a **railway signaling component** within AIDA framework
- High-level comparison of two verification approaches

Deductive Approach

- Dafny verification
- Implementation-level reasoning
- Semi-automatic proofs

Model Checking

- nuXmv model checker
- (Abstract) Model-level reasoning
- Automatic proofs (or counterexamples)

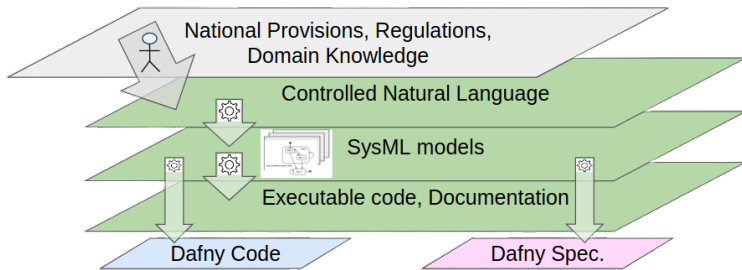
AIDA: Model-Based Design of Railway Logics

Inputs

- DSL (domain specific language)
- Logic specification

Outputs

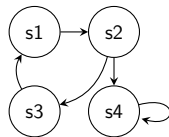
- Documentation
- C for execution
- Python for debugging
- **Dafny for verification**



AIDA Component Structure

- **Generic** components defined extended FSMs
- **FSM Elements:**
 - Variables
 - Lists of 'connected entities'
 - Transitions with guards and effects
- The generic component is reusable across different stations
- **Execution:** given a configuration, a scheduler governs execution

Generic component



instance1



instance2



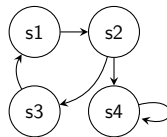
...

more instances

AIDA Component Structure

- **Generic** components defined extended FSMs
- **FSM Elements:**
 - Variables
 - Lists of 'connected entities'
 - Transitions with guards and effects
- The generic component is reusable across different stations
- **Execution:** given a configuration, a scheduler governs execution

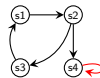
Generic component



instance1



instance2



...

more instances

Current Status

AIDA is being used to implement a new software-based interlocking logic, with intermediate logic currently in design and testing phases.

The Signal Component



Component Structure

- **Role:** manages signal color aspects
- 34 local variables
- 12 transitions
- 5 outputs (CH1–CH5)

The Signal Component



Component Structure

- **Role:** manages signal color aspects
- 34 local variables
- 12 transitions
- 5 outputs (CH1–CH5)

Aspect	CH1	CH2	CH3	CH4	CH5
None	off	off	off	off	off
●	on	off	off	off	off
●	off	on	off	off	off
●	off	phase0	off	off	off
●	off	off	on	off	off
● ●	off	on	off	off	on
● ●	off	phase0	off	off	phase0
● ●	off	phase0	off	off	phase180
● ●	on	off	off	on	off
● ●	on	phase0	off	off	off
● ●	on	off	off	off	on
● ●	off	on	off	on	off

The Signal Component



Component Structure

- **Role:** manages signal color aspects
- 34 local variables
- 12 transitions
- 5 outputs (CH1–CH5)

Aspect	CH1	CH2	CH3	CH4	CH5
None	off	off	off	off	off
●	on	off	off	off	off
●	off	on	off	off	off
●	off	phase0	off	off	off
●	off	off	on	off	off
● ●	off	on	off	off	on
● ●	off	phase0	off	off	phase0
● ●	off	phase0	off	off	phase180
● ●	on	off	off	on	off
● ●	on	phase0	off	off	off
● ●	on	off	off	off	on
● ●	off	on	off	on	off

Safety Property ϕ

The signal component should always output one valid aspect.

Signal Behavior: Scheduler Phases

single scheduler execution cycle



Signal Behavior: Scheduler Phases

single scheduler execution cycle

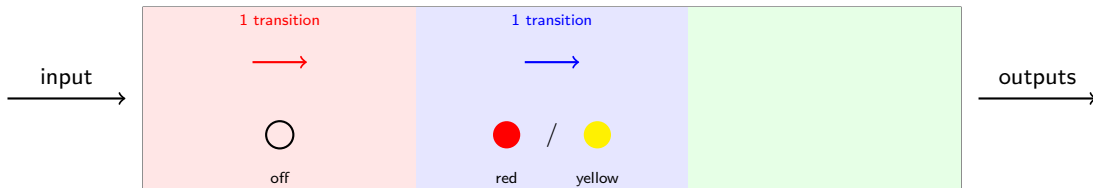


Signal Component Behavior

- **Initialization:** All channels → "off"

Signal Behavior: Scheduler Phases

single scheduler execution cycle

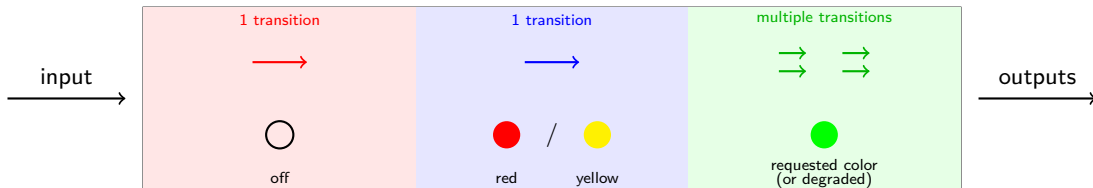


Signal Component Behavior

- **Initialization:** All channels → "off"
- **Second phase:** Set default color (red or yellow)

Signal Behavior: Scheduler Phases

single scheduler execution cycle



Signal Component Behavior

- **Initialization:** All channels → “off”
- **Second phase:** Set default color (red or yellow)
- **Third phase:** Handle requests from other components:
 - Validation checks → display requested color
 - Failed checks → degradation (green → yellow)

Dafny Verification: Auto-Generated Code

Dafny Programming Language

- **Deductive verification:** Programming language with built-in verification capabilities
- **AIDA integration:** Auto-generates Dafny code with contracts for transitions

Dafny Verification: Auto-Generated Code

Dafny Programming Language

- **Deductive verification:** Programming language with built-in verification capabilities
- **AIDA integration:** Auto-generates Dafny code with contracts for transitions

Auto-Generated Guard Example: Green Signal Request

```
method green_signal_transition_guard() returns (b : bool)
  ensures (green_request &&
     $\forall \text{diag} :: \text{diag} \text{ in } \text{diagnoses} \Rightarrow \text{diag.inactive}() \iff b$ ) {
  b := green_request;
  if (b) {
    for i := 0 to |diagnoses|
      invariant (green_request &&
         $\forall \text{diag} :: \text{diag} \text{ in } \text{diagnoses}[\dots i] \Rightarrow \text{diag.inactive}() \iff b$ ) {
        if (!diagnoses[i].inactive()) {
          b := false; break;
        } } } }
```

Dafny Deductive Verification Strategy

Phase-by-Phase Inductive Proof

- **Goal:** Use auto-generated contracts to show inductive steps
- **Inductive structure:** Each phase preserves safety property ϕ
- **Compositional reasoning:** Combine individual phase proofs

Dafny Deductive Verification Strategy

Phase-by-Phase Inductive Proof

- **Goal:** Use auto-generated contracts to show inductive steps
- **Inductive structure:** Each phase preserves safety property ϕ
- **Compositional reasoning:** Combine individual phase proofs

Phase 1

```
method first_phase_proof()  
  requires initialization_guard()  
  ensures all_channels_off()  
  ensures safety_phi()  
{  
  first_phase_effects();  
}
```

✓ **Success**

Dafny Deductive Verification Strategy

Phase-by-Phase Inductive Proof

- **Goal:** Use auto-generated contracts to show inductive steps
- **Inductive structure:** Each phase preserves safety property ϕ
- **Compositional reasoning:** Combine individual phase proofs

Phase 1

```
method first_phase_proof()  
  requires initialization_guard()  
  ensures all_channels_off()  
  ensures safety_phi()  
{  
  first_phase_effects();  
}
```

✓ Success

Phase 2

```
method second_phase_proof()  
  requires all_channels_off()  
  requires default_color_guard()  
  ensures default_color()  
  ensures safety_phi()  
{  
  second_phase_effects();  
}
```

✓ Success

Dafny Deductive Verification Strategy

Phase-by-Phase Inductive Proof

- **Goal:** Use auto-generated contracts to show inductive steps
- **Inductive structure:** Each phase preserves safety property ϕ
- **Compositional reasoning:** Combine individual phase proofs

Phase 1

```
method first_phase_proof()  
  requires initialization_guard()  
  ensures all_channels_off()  
  ensures safety_phi()  
{  
  first_phase_effects();  
}
```

✓ **Success**

Phase 2

```
method second_phase_proof()  
  requires all_channels_off()  
  requires default_color_guard()  
  ensures default_color()  
  ensures safety_phi()  
{  
  second_phase_effects();  
}
```

✓ **Success**

Phase 3

```
method third_phase_proof()  
  requires safety_phi()  
  requires complex_guard()  
  ensures safety_phi()  
{  
  third_phase_effects();  
  // FAILS verification!  
}
```

✗ **Failed**

Dafny Failure: Bug Discovery

First Dafny Failure

- Failed to prove: individual transitions in third phase preserve ϕ
- **Counterexample:** Pre/post-state generated by Dafny
- **Challenge:** No concrete execution trace provided

Dafny Failure: Bug Discovery

First Dafny Failure

- Failed to prove: individual transitions in third phase preserve ϕ
- **Counterexample:** Pre/post-state generated by Dafny
- **Challenge:** No concrete execution trace provided

Manual Investigation Reveals Bug

- **Root cause:** Corner case in handling degradation
- **Scenario:** Color request + internal validation failure: trying to display yellow with red already active.
- **Result:** **CH1** and **CH2** active: invalid signal aspect.

Dafny Failure: Bug Discovery

First Dafny Failure

- Failed to prove: individual transitions in third phase preserve ϕ
- **Counterexample:** Pre/post-state generated by Dafny
- **Challenge:** No concrete execution trace provided

Manual Investigation Reveals Bug

- **Root cause:** Corner case in handling degradation
- **Scenario:** Color request + internal validation failure: trying to display yellow with red already active.
- **Result:** **CH1** and **CH2** active: invalid signal aspect.

Bug Confirmation & Fix

- Confirmed using AIDA logic simulator
- Subsequent fix by signaling experts

Dafny Second Attempt: Remaining Challenges

After Bug Fix

- ✓ Individual transitions in the third phase preserve ϕ
- **But...** still incomplete verification

Dafny Second Attempt: Remaining Challenges

After Bug Fix

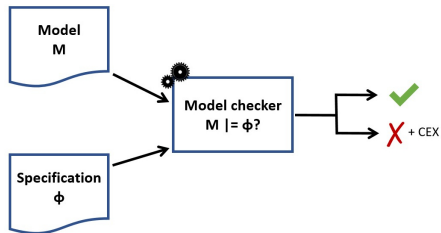
- ✓ Individual transitions in the third phase preserve ϕ
- **But...** still incomplete verification

Remaining Issue

Multiple transitions in the third phase!

- Dafny failed to prove: arbitrary sequence of transitions preserves ϕ
- **Ambiguous failure:** Real bug or missing lemmas?

Model Checking: Alternative Approach



Why Model Checking?

- **Model checking:** fully automated, model-level reasoning, limited scalability, provides concrete counterexamples.

Motivation: Applied model checking to the signal component to resolve ambiguity in Dafny's failure. Abstraction was required to manage system complexity.

Model Checking with nuXmv

Abstraction Strategy

- **Localization reduction**
- Faithful modeling: signal + 2 direct neighbors
- Other components: nondeterministic environment
- Sound over-approximation

Model Structure

- Single scheduler cycle
- Scheduling order preserved
- Environment transitions between phases

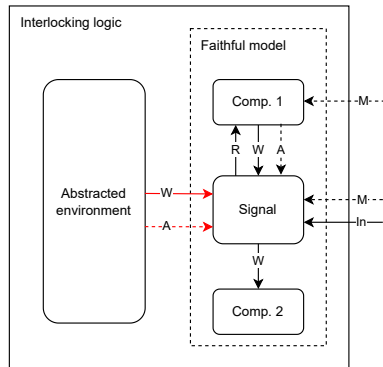


Figure: Abstract model diagram

Model Checking Results: Second Bug Found

First nuXmv Counterexample

- **11-step execution trace** violating ϕ
- **Scenario:** Two conflicting color requests in same cycle
 - ① Request from environment component
 - ② Request from diagnostic component
- **Result:** Incompatible output channels active simultaneously

Model Checking Results: Second Bug Found

First nuXmv Counterexample

- **11-step execution trace** violating ϕ
- **Scenario:** Two conflicting color requests in same cycle
 - ① Request from environment component
 - ② Request from diagnostic component
- **Result:** Incompatible output channels active simultaneously

Bug Confirmation

- Trace shared with railway engineers
- Confirmed on railway yard simulator
- **Root cause:** Missing exclusion check in nominal transition
- Signal should have been blocked from service, one request should have been ignored

Model Checking: Environmental Assumptions

After Second Fix

- Logic updated with proper exclusion check
- nuXmv re-executed...
- **Another counterexample found!**

Model Checking: Environmental Assumptions

After Second Fix

- Logic updated with proper exclusion check
- nuXmv re-executed...
- **Another counterexample found!**

Third Counterexample Analysis

- **Scenario:** Two external requests from environment in same cycle
- **Spurious check:** Other components cannot actually send both requests
- **Issue identified:** Signal lacks **defensive design**

Model Checking: Environmental Assumptions

After Second Fix

- Logic updated with proper exclusion check
- nuXmv re-executed...
- **Another counterexample found!**

Third Counterexample Analysis

- **Scenario:** Two external requests from environment in same cycle
- **Spurious check:** Other components cannot actually send both requests
- **Issue identified:** Signal lacks **defensive design**

Resolution & Final Verification

- **Environmental constraint:** At most one environment request per cycle
- With this assumption: ϕ successfully verified ✓
- **Design insight:** Signal should be more robust

Comparison: Deductive vs Model Checking

Dafny - Deductive Verification

Strengths:

- Direct reasoning on implementation
- Automatic contract generation

Limitations:

- Manual invariant engineering
- Ambiguous failure diagnosis
- Complex inductive proofs

nuXmv - Model Checking

Strengths:

- Fully automated analysis
- Concrete counterexamples
- Global state exploration

Limitations:

- Requires abstraction
- Need environment assumptions

Comparison: Deductive vs Model Checking

Dafny - Deductive Verification

Strengths:

- Direct reasoning on implementation
- Automatic contract generation

Limitations:

- Manual invariant engineering
- Ambiguous failure diagnosis
- Complex inductive proofs

nuXmv - Model Checking

Strengths:

- Fully automated analysis
- Concrete counterexamples
- Global state exploration

Limitations:

- Requires abstraction
- Need environment assumptions

Insight

Complementary approaches: Dafny for checking local properties, nuXmv for global analysis and debugging. Both completed proofs in less than a minute.

Conclusions

Formal Methods in Industrial Railway Context

- **Real-world application** of verification techniques to RFI component
- **Successful bug discovery** with both deductive and model checking approaches

Conclusions

Formal Methods in Industrial Railway Context

- **Real-world application** of verification techniques to RFI component
- **Successful bug discovery** with both deductive and model checking approaches

Importance of Understandable Counterexamples

- **Dafny limitation:** Complex "failures to prove" not informative for engineers
- **nuXmv advantage:** Concrete 11-step execution traces
- **Key insight:** Railway experts need actionable diagnostic information

Conclusions

Formal Methods in Industrial Railway Context

- **Real-world application** of verification techniques to RFI component
- **Successful bug discovery** with both deductive and model checking approaches

Importance of Understandable Counterexamples

- **Dafny limitation:** Complex "failures to prove" not informative for engineers
- **nuXmv advantage:** Concrete 11-step execution traces
- **Key insight:** Railway experts need actionable diagnostic information

Future Work

- **Generalize nuXmv model construction** from Dafny failures
- **Parameterized components:** Beyond current nuXmv scope
- **Systematic integration** in AIDA verification workflow

Thank You

Questions?

