Integration of Formal Methods into Design and Implementation of Aerospace Systems

Kristin Yvonne Rozier
Rice University

December 11, 2014
Formal Methods Have Greatly Impacted Aerospace Engineering

- Expected design-time component
- Recommended in DO-178B standard for certification
- Successfully applied in many aerospace contexts...
Successes: Full-Scale and Real-Life

Explicit Model Checking


Successes: Full-Scale and Real-Life

Symbolic Model Checking


Successes: Full-Scale and Real-Life

Probabilistic Model Checking


Successes: Full-Scale and Real-Life, By Project

Theorem Proving

- **ACAS-X** (Airborne Collision Avoidance System X)

- **ACCoRD** (state-based conflict detection & resolution algorithms)

- **Chorus** (tactical conflict & loss of separation detection & resolution)

- **Stratway** (strategic separation)

- **KB3D** (CD&R)
Successes: After the Design Phase . . .

Static Analysis, Dynamic Analysis, and Symbolic Execution


- **SymbolicPathFinder** (symbolic analysis)
Successes: Mission Time

Runtime Monitoring

- **rt-R2U2** (system & safety health management)

- **Copilot**

Runtime Monitoring faces fewer challenges that design-time verification:
- less formal
- only specs needed
- specs inherited from design time
Still not often adapted to flight-certifiable!
**Progress**

- Impactful results
- Efficiency of analysis
- Coverage of analysis
- Adaptability to specific problems
- Scalability
- Recognition of the need for formal methods in aerospace system design and runtime

In the *design stage*, where changes are cheapest, easiest, and most impactful is where we face the biggest bottlenecks...
A Goal Aerospace System Design Process

System Design

Model Check

Model Validation via Model Checking

Model Validation Specification

SPEC DEBUGGING

REVISE

ERROR NO

Build Prototype

Testing and Simulation

ERROR NO

YES

USE SPECIFICATIONS FOR RUNTIME MONITORING

SPEC DEBUGGING

Kristin Yvonne Rozier
Integration of FM into Design & Implementation of Aero Sys
Bottlenecks

- Creating a system model/complete formalizable design
Bottlenecks

- Creating a **system model**/complete formalizable design
- Writing **formal specifications**/getting precise requirements
<table>
<thead>
<tr>
<th>Bottlenecks</th>
<th>Need for a New Direction</th>
<th>Future Challenges</th>
</tr>
</thead>
</table>

- Creating a **system model**/complete formalizable design
- Writing **formal specifications**/getting precise requirements
- Artifacts analyzable by one tool do not **translate** to any other
  - Need to know from the beginning **all features**/**expressability** that will be needed and choose the right tool from the start
  - Cannot change direction, translate between tools, or start again
Bottlenecks

- Creating a **system model**/complete formalizable design
- Writing **formal specifications**/getting precise requirements
- Artifacts analyzable by one tool do not translate to any other
  - Need to know from the beginning **all features**/**expressability** that will be needed and choose the right tool from the start
  - Cannot change direction, translate between tools, or start again
- Constantly have to re-explain the model/specification **context**
  - continuous vs discrete **time**
  - level of **abstraction**
  - **types** of system that can be reasoned about
Bottlenecks

- Creating a **system model**/complete formalizable design
- Writing **formal specifications**/getting precise requirements
- Artifacts analyzable by one tool do not **translate** to any other
  - Need to know from the beginning **all features**/expressability that will be needed and **choose the right tool** from the start
  - Cannot change direction, translate between tools, or start again
- Constantly have to re-explain the model/specification **context**
  - continuous vs discrete **time**
  - level of **abstraction**
  - **types** of system that can be reasoned about
- Outputs also require **human translation** (e.g. counterexamples)

Each project requires a **very active middleman** engaging for a **long** time!
The Bottom Line:

**Bottom Line:** INPUTS to formal analysis are the BIGGEST challenge.
Proposed Solutions

- code-level analysis
- NLP → LTL
- structured inputs
- training system designers in formal methods

Each may be *part of* a solution
None of these solves the problem
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using no coding standards
**Code-Level Analysis Does Not Solve the Modeling Problem**

- Mostly written by untrained engineers using **no coding standards**
- **No organization/spaghetti code**

---

**Kristin Yvonne Rozier**

Integration of FM into Design & Implementation of Aero Sys
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using no coding standards
- No organization/spaghetti code
- Often very large or intractable, full of dead code
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using **no coding standards**
- **No organization/spaghetti code**
- Often very large or **intractable**, full of dead code
- **Not an actual model** of the intended system
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using no coding standards
- No organization/spaghetti code
- Often very large or intractable, full of dead code
- Not an actual model of the intended system
- People inherently write bad code
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using **no coding standards**
- **No organization/spaghetti code**
- Often very large or **intractable**, full of dead code
- Not an actual model of the intended system
- People inherently write bad code
- Does not solve the problems of **specifications**
  - Automatic specifications check code quality, not **design**
  - Development mismatch: code is a final system; **too late to check design-time specifications**
Code-Level Analysis Does Not Solve the Modeling Problem

- Mostly written by untrained engineers using **no coding standards**
- **No organization/spaghetti code**
- Often very large or **intractable**, full of dead code
- **Not an actual model** of the intended system
- People inherently write bad code
- Does not solve the problems of **specifications**
  - Automatic specifications **check code quality, not design**
  - Development mismatch: code is a final system; **too late to check design-time specifications**

  **Design ≠ code**

  Need to analyze the **design**

  Code needs to be auto-generated; people should not write the code

  **Still useful for re-use of trusted components, analysis of final composed code, emergency verification**
NLP Does Not Solve the Specification Problem

- Can extract some types: maybe LTL but not theorems
- Can automatically run specification debuggers

Remaining Challenges:
- Incomplete requirements
- Matching variables to a model
- Maintaining level of abstraction
- Context
- Output organization and usability
Solving the Problem: The View from 10K Feet

No more individual tools!

Need a *toolset*!

- Need to **combine multiple formal methods tools** into a unified suite
- Must be **standardized** in some way
- Must be **sold by a company**
  - Need a start-up or business to *market* it widely at a reasonable price
- Must be **taught in aerospace engineering departments at universities**
Solving the Problem: Need for a Standard Toolset

- **Need a professional, intuitive GUI**
  - Restrictions on context serving as built-in reminders
  - Needs to do the job of today’s middleman!
- **Needs to organize, index, and cross-reference** specifications and modeling components
- **Needs to allow for:**
  - compositional modeling,
  - component/specification re-use,
  - automatic documentation and structure visualization
- **Cannot let people write their own code!**
Solving the Problem: Need for a Standard Toolset

- Must be **compatible** with other V&V methods
- Both able to:
  - Translate between analysis techniques
  - Provide **artifacts** for testing and simulation
  - Test case generation
  - Produce **simulation** models
  - Automate visualization (i.e. counterexamples)
- Needs a **unified choice of graphical displays** for outputs (counterexamples, fault trees, dependencies, etc.)
- Replace middleman with customer service department
  - System designers can **pay for support** or call in with individual questions

*We don’t have any tools for some of these yet!*
For Our Future

- Need **models/formal system descriptions**
- Need **specifications**
- Need **output visualization**
- Need **unified toolset** for widespread adaptation
- Need to **overcome stigmas**: learning curve, usefulness, etc.
For Our Future

- Need models/formal system descriptions
- Need specifications
- Need output visualization
- Need unified toolset for widespread adaptation
- Need to overcome stigmas: learning curve, usefulness, etc.

... So how do we do that?