Specification: The Biggest Bottleneck in Formal Methods and Autonomy\(^1\)

Kristin Yvonne Rozier
Iowa State University

February 13, 2017

Design-Time Verification!

- Expected design-time component
- Recommended in DO-178B/C, D0-254 standards for
- Successfully applied in many aerospace contexts...
Runtime Verification and System Health Management!

- Required for **Autonomy**
- New: **Intelligent Interfaces**
- Hot topic: **UTM, Mars, NextGen, ...**
A Recent Motivation...

Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
A Recent Motivation...

Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
A Recent Motivation... 
Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
A Recent Motivation... 

Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)
A Recent Motivation...

Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)
- Navigation system calculated a negative altitude
  - premature release of parachute & backshell
  - firing of braking thrusters
  - activation of on-ground systems at 2 miles (3.7 km) altitude
A Recent Motivation...  
Crash of ESA’s ExoMars Schiaparelli Lander

- October 19, 2016
- parachute deployed at:
  - altitude of 7.5 miles (12 km)
  - speed of 1,1075 mph (1,730 km/h)
- heat shield ejected at altitude of 4.85 miles (7.8 km)
- IMU miscalculated saturation-maximum period (by 1 sec)
- Navigation system calculated a negative altitude
  - premature release of parachute & backshell
  - firing of braking thrusters
  - activation of on-ground systems at 2 miles (3.7 km) altitude
- Crash at 185 mph (300 km/h)
A Recent Motivation...

Crash of ESA’s ExoMars Schiaparelli Lander

Sanity Checks

Relevant to this Mission:

- The altitude cannot be negative.
- The rate of change of descent can’t be faster than gravity.
- The $\delta$ altitude must be within nominal parameters; it cannot change from 2 miles to a negative value in one time step.
- The saturation-maximum has an a priori known temporal bound.

These *sanity checks* could have prevented the crash.

Capability of such observations is *required for autonomy.*
Enabling Autonomy

What do the humans do?

1. Pilot/control the system (on-board or remotely)
2. Provide self-awareness
3. Respond to off-nominal conditions
4. Make tough judgment calls
Enabling Autonomy

**What do the humans do?**
**And how do we automate that?**

1. Pilot/control the system (on-board or remotely)
   - Autopilot

2. Provide self-awareness
   - Runtime System Health Management (SHM)

3. Respond to off-nominal conditions
   - Automated replanning and learning

4. Make tough judgment calls
   - Algorithms like TCAS beat humans
   - Ethical decisions are an open problem . . .

**Analysis from design-time and runtime is required for autonomy.**
Specifications:
Required for Formal Methods and Autonomy!

**Formal Methodology**

1. specification language
2. repertoire of proof methods
   - make early precise decisions about major functionalities
   - remove ambiguities from the description of expected behavior

---

Specifications: Required for Formal Methods and Autonomy!

Formal Methodology

1. specification language
   - Linear Temporal Logic

2. repertoire of proof methods
   - make early precise decisions about major functionalities
   - remove ambiguities from the description of expected behavior

---

A Goal Aerospace System Design Process

System Design → Model Check → ERROR → NO

Model Validation Specification → ERROR → NO

Model Checking via Model Validation → SPEC DEBUGGING

SPEC DEBUGGING → YES

USE SPECIFICATIONS FOR RUNTIME MONITORING

YES

A Goal Aerospace System Design Process

System Design → Model Check → Model Verification → SPEC DEBUGGING → Model Validation via Model Checking → SPEC DEBUGGING → Build Prototype → Testing and Simulation → ERROR → NO → USE SPECIFICATIONS FOR RUNTIME MONITORING → YES → ... Garbage in, garbage out!

Bottom Line: INPUTS to formal analysis are the BIGGEST challenge.
Synthesis!

Model checking: check $M \models \phi$

Problems:\(^4\)

- Designing $M$ is hard and expensive
- Re-designing $M$ when $M \not\models \phi$ is hard and expensive
- Synthesis: start from $\phi$, design $M$ such that $M \models \phi$
  - Simplifies verification
  - No re-design
  - For algorithmic derivations: no design!

---

Synthesis!

Model checking: check $M \models \phi$

Problems:

- Designing $M$ is hard and expensive
- Re-designing $M$ when $M \not\models \phi$ is hard and expensive
- Synthesis: start from $\phi$, design $M$ such that $M \models \phi$
  - Simplifies verification
  - No re-design
  - For algorithmic derivations: no design!

What about $\phi$?

---

Synthesis!

Model checking: check $M \models \phi$

Problems:\(^4\)

- Designing $M$ is hard and expensive
- Re-designing $M$ when $M \not\models \phi$ is hard and expensive
- Synthesis: start from $\phi$, design $M$ such that $M \models \phi$
  - Simplifies verification
  - No re-design
  - For algorithmic derivations: no design!

What about $\phi$?

We need LTL Genesis!

Some critical systems are designed without formal requirements.

Some design processes don’t formally define requirements until the testing phase.

Early specifications often mix many different objectives:
- levels of detail/abstraction
- how the system is defined vs how the system should behave
- legal-speak on why the system satisfies rules
- desires/opinions of designers

---

Specification Origins

Specification Extraction Strategies

- **Human Authorship:**
  - Train system designers to write formal specifications first
  - Pair designers with formal methods team to write specifications

- **Natural Language Processing:** extract formal specifications from English Operational Concepts
  - Highly input-dependent: assumptions, implied/arbitrary functions
  - Hard to measure correctness, completeness

- **Specification Mining:** extract behaviors from existing systems

- **Static Analysis:** map all paths of a program
  - Hard to differentiate normal usage from exceptions

- **Learning/Dynamic Invariants:** analyze actual executions; observe use-cases

- **Specification Wizard:** Semi-automated exploration of system facets, guided by human input

---

Specifications for Free?\(^7\)

- Combine *specification mining*, *test-case generation*, *static analysis*, and *dynamic invariants* to extract specifications automatically!
- Can use specifications mined from code
  - Specification validation == *software defect detection*
  - Promising for *software runtime verification*
  - Still *need code*...
    - What about *early design time*?
    - What about *cyber-physical system* specifications?
- Can use specifications *extracted from last version* for *new designs*
  - Challenges with *specialization/levels of abstraction/relevance*
- Other challenges:
  - Scalability
  - Efficiency
  - Expressiveness

\(^7\) A. Zeller. “Specifications for Free.” NFM 2011.
How should we measure specification quality?⁸

- How can we know when we’re done?
- How good are the specifications?
- How can we measure the completeness, correctness, coverage, or general quality of a set of specifications?

---

Sanity Checks

- Satisfiability
- Vacuity
- Realizability
- Coverage
How do we best use specifications?\(^9\)

- Design lifecycles for different cyber-physical systems?
- How to indoctrinate formal specification into diverse teams of system designers?
- Barriers to adoption:
  - time to write/validate
  - learning curves
  - culture
- Need an end-to-end process for specification extraction, usage

What should our roadmap look like for a future full of well-specified (formally analyzable) critical systems?

Specification Use Strategies

- **Property-Based Design:** from specifications to systems

- **Synthesis:** generate $M$ such that $M \models \phi$
  - For cyber-physical systems?

- **Specification-Based Testing:** use specifications in test-case generation

- **From Design- to Run-Time:** carry specifications through the design cycle
  - Specification design lifecycle?

- **Maintenance:** using specifications in system up-keep
  - Maintenance of specifications?
Specifications: Classes

- Safety
- Response
- Reactivity

Others:
- Safety/Liveness/Guarantee/Obligation
- Fairness/Justice/Compassion

Still too coarse and tied to syntax for practical use; need functional and hierarchical specification . . .

---

Specifications: Formula Patterns

- Leverage experience with design and coding patterns
- Define *specification pattern system*
  - capture *recurring solutions*; generalize across specific/domain problems
  - encourage *re-use*
  - make transparent the *means by which requirements are satisfied*
  - *name, intent, logic* (language), *scope* (time interval), *relationship* to other patterns
- Characterized by:
  - Solves a Specific Problem, e.g. not too abstract
  - Proven Concept effective in practice
  - Not Obvious or direct application of basic principles
  - Describes Relationships, not single components
  - Generative, describes how to construct a solution
- Organized in a *hierarchy based on semantics*

---

Specifications: Automata Patterns

Challenges Remain with Translational Semantics:

- Formula patterns are not compositional
- Need consistency with semantics of informal definitions

Automata-based patterns:

- Compositional: based on compositions of patterns (logic executions) and scopes (time)
- Homogeneous: don’t flatten key patterns/scopes separation
- Extensible: compositional semantics allow adding patterns & scopes
- Generic: can combine any pattern and any scope
- Faithful: formal guarantee that the translated temporal formula is faithful to the intended natural semantics

---

Specifications: Automata Patterns¹²

Challenges Remain with Translational Semantics:

- Formula patterns are **not compositional**
- **Need consistency** with semantics of informal definitions

Automata-based patterns:

- **Compositional**: based on compositions of patterns (logic executions) and scopes (time)
- **Homogeneous**: don’t flatten key patterns/scopes separation
- **Extensible**: compositional semantics allow adding patterns & scopes
- **Generic**: can combine any pattern and any scope
- **Faithful**: formal guarantee that the translated temporal formula is faithful to the intended natural semantics

What about runtime specifications for autonomous systems?

Specifications: Functional Patterns?

- Work on specification patterns focuses mostly on design time.
- Formula patterns are not compositional.
- Automata patterns are not decomposable.
  - Hard for cyber-physical systems during runtime.
  - Sanity checks are more complex.
- What if that is a *functional pattern*?
- Are there different patterns for specification functions, e.g., between design time and runtime?
Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

---

Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

---

Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

---

Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

---

Runtime Functional Specification Patterns

- Rates
- Ranges
- Relationships
- Control Sequences
- Consistency Checks

We need to expand specification patterns to runtime!

R2U2: Runtime Specification Patterns in the Field

1. **TL Observers**: Efficient temporal reasoning
   - **Asynchronous**: output \( \langle t, \{0, 1\} \rangle \)
   - **Synchronous**: output \( \langle t, \{0, 1, ?\} \rangle \)
     - **Logics**: MTL, pt-MTL, Mission-time LTL
     - **Variables**: Booleans (from system bus), sensor filter outputs

2. **Bayes Nets**: Efficient decision making
   - **Variables**: outputs of TL observers, sensor filters, Booleans
   - **Output**: most-likely status + probability

---

R2U2: Runtime Specification Patterns in the Field

1. **TL Observers**: Efficient temporal reasoning
   - **Asynchronous**: output \( \langle t, \{0, 1\} \rangle \)
   - **Synchronous**: output \( \langle t, \{0, 1, ?\} \rangle \)
     - **Logics**: MTL, pt-MTL, Mission-time LTL
     - **Variables**: Booleans (from system bus), sensor filter outputs

2. **Bayes Nets**: Efficient decision making
   - **Variables**: outputs of TL observers, sensor filters, Booleans
   - **Output**: most-likely status + probability

How do we organize R2U2 specifications?

---


---
R2U2: Runtime Specification Patterns in the Field

<table>
<thead>
<tr>
<th>Health Nodes / Failure Modes</th>
<th>magnetometer sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_FG</td>
<td>Receiver underrun</td>
</tr>
<tr>
<td>H_FC_RxUR</td>
<td>Receiver overrun</td>
</tr>
<tr>
<td>H_FC_RxOVR</td>
<td>Transmitter overrun in sensor</td>
</tr>
<tr>
<td>H_FG_TxOVR</td>
<td>Transmitter error in sensor</td>
</tr>
<tr>
<td>H_FG_TxErr</td>
<td></td>
</tr>
</tbody>
</table>

We combine specifications in a way that is:
- hierarchical/structured
- compositional
- cross-language

---

How should we organize specifications?

- How do we store specifications in an accessible way?
  - Allow for automated analysis, including verification?
  - Enable re-use: design-time → runtime → future systems

- How do we pair English and Formal specifications?

- How do we preserve the hierarchical structure, compositionality, and relationships between specifications?

- Can we do this in a performable way?
Specification Organization Strategies

- Scenario Definition Languages
Specification Organization Strategies

- Scenario Definition Languages
  - $M$ vs $\phi$?
Specification Organization Strategies

- **Scenario Definition Languages**
  - $M$ vs $\phi$?

- **Matlab/Simulink**
  - not scalable
  - not backwards-compatible for long specification lifecycles
  - not designed for this (kludgy) . . .
Specification Organization Strategies

- **Scenario Definition Languages**
  - $M$ vs $\phi$?

- **Matlab/Simulink**
  - not scalable
  - not backwards-compatible for long specification lifecycles
  - not designed for this (kludgy) ...

- **Database: SQL**
  - relationships are inherently non-tabular
    - requires flattening the database
    - requires extensive JOINs; non-performable

Laboratory for Temporal Logic
Kristin Yvonne Rozier
Specification: The Biggest Bottleneck in FM & Autonomy
Specification Organization Strategies

- **Scenario Definition Languages**
  - $M$ vs $\phi$?

- **Matlab/Simulink**
  - not scalable
  - not backwards-compatible for long specification lifecycles
  - not designed for this (kludgy) . . .

- **Database: SQL**
  - relationships are inherently non-tabular
    - requires flattening the database
    - requires extensive JOINs; non-performable

None of these solve the organization problem!
If we do it right, specifications are everywhere!

- How do we organize specifications for each subsystem, subcomponent, level of abstraction?
- How do we mine specifications for data, patterns, statistical analysis, coverage?
- How do we search specifications?
- How do we sort specifications?
- How do we integrate specification languages for different purposes?
- How do we make specifications available for reuse?

We have a Big Data of Specifications problem!
A property graph \( G = \{N, P, R\} \) where \( N \) is a set of nodes, \( P \) is a set of properties, \( R \) is a set of relationships,

**Node:** document
- contain sets of properties

**Properties:** key/value pair
- Key: string
- Value: arbitrary data type

**Relationships:** connect & structure nodes
- direction
- label
- start node
- end node
- [properties]

---

\(^{16}\) https://neo4j.com/
Specification Challenges: to Infinity and Beyond!

- Where are we now?
  - Continuously re-assess . . .
- Where will we get specifications from?
- How should we measure specification quality?
- How do we best use specifications?
- How should we organize specifications?
Specification Challenges: to Infinity and Beyond!

- Where are we now?
  - Continuously re-assess . . .
- Where will we get specifications from?
- How should we measure specification quality?
- How do we best use specifications?
- How should we organize specifications?

... in the context of cyber-physical, autonomous systems?