Formal Methods for V&V and Certification

Dr. Laura Humphrey
Team Lead, V&V of Complex & Autonomous Systems
Aerospace Systems Directorate, Autonomous Controls Branch (AFRL/RQQA)

October 2019
Outline

• Motivation
• Overview of formal methods
• Formal methods for autonomous systems
• Supplementary & alternative approaches
• Where our group is going
• Successes of formal methods
• Resources
Outline

• Motivation
• Overview of formal methods
• Formal methods for autonomous systems
• Supplementary & alternative approaches
• Where our group is going
• Successes of formal methods
• Resources
• Intersperse answers throughout
How is Assured Autonomy Interpreted in My Area?

- **Autonomy**
  - General word cloud: non-deterministic, reactive, adaptive
  - Unique part of my definition: too complex to V&V through current approaches

- **Assured (formal methods)**
  - Having mathematically rigorous evidence of safety and correctness for critical functionality
  - Potentially less rigorous or thorough evidence for less critical functionality (i.e., commensurate with risk)
Motivation

Rigorous

Systems Engineering

- CONOPS
- Design
  - Requirements
  - Architecture
  - Detailed
- Implementation
- Validation
- Verification

Transition

20.5% faults found
300-1000x est. cost for fault removal

70% faults introduced
3.5% faults found
1x est. cost for fault removal

10% faults introduced
59.5% faults found
20-80x est. cost for fault removal

20% faults introduced
16% faults found
5x est. cost for fault removal

Rigorous design to decrease errors & cost[1,2,3]

Rapid

- Exponential software growth is making V&V through testing unaffordable

- Boeing & Airbus[4]: 27M+ SLOC → $10B+
- Driverless car testing[5]: 11B miles, $6B, 500y

Automated analysis to speed up V&V

Boeing
Airbus
Unaffordable

8M
134M
299M

B777-200: 4M
B737: 470K
A330/340: 2M
B747-200: 370K
A320: 800K
A300FF: 40K
B757-200, B767-300: 190K
A310: 400K
INS: 0.8K

1975
1990
2005
2020

Curve implies SLOC doubles about every 4 years

1.75
3.5
7.8B
$38M
$81M
$290M

1x
5x
dept. cost for fault removal

10%
59.5%
20%
59.5%
520-80x
dept. cost for fault removal

300-1000x
What are Formal Methods?[6,7]

• Mathematically-based techniques for specification, design, and verification of software & hardware
  • Formal logic  • Discrete mathematics  • Computer-readable languages

• Two major activities
  • Modeling – high- & low-level requirements, architecture, source code, object code
  • Analysis – consistency, traceability, compliance, robustness, completeness

• **Formal analysis must be sound** (never say a property is true if it is not)

• Three categories
  • Theorem proving – use axioms & inference rules to prove properties of a model
  • Model checking – exhaustively search a model for a counterexample to a property
  • Abstract interpretation – construct & analyze conservative representations of software
Benefits of Formal Methods[6,7]

• Remove ambiguity (rigorous) → enable semi- or automated analysis (rapid)

• Analysis of general properties such as
  • Compliance: low-level model complies with higher-level model/architecture/requirement
  • Traceability: all low-level model aspects trace to a higher-level model/architecture/requirement & vice versa

• Analysis of software properties such as
  • Freedom from runtime exceptions
  • Freedom from deadlock
  • Non-interference between different levels of criticality
  • Worst case execution time (WCET)
  • Bounds on stack size during execution
  • Freedom from unintended function
  • Correct synchronous or asynchronous behavior

• Consistency: absence of conflicts
• Completeness: output specified for all input conditions & vice versa
• Robustness: nominal & off-nominal conditions

• Traceability: all low-level model aspects trace to a higher-level model/architecture/requirement & vice versa
A Simple Example

• Consider a function that returns \((a + b)/c\) bounded by a threshold

• What could go wrong with this simple function?

```plaintext
function foo(a, b, c, threshold : Float) return Float is
  x : Float;
begin
  x := (a + b) / c;
  if x <= threshold and x >= -threshold then
    return x;
  elsif x > threshold then
    return threshold;
  else
    return -threshold;
  end if;
end;
```
A Simple Example

• Consider a function that returns \((a + b)/c\) bounded by a threshold
• What could go wrong with this simple function?

```function foo(a, b, c, threshold : Float) return Float is
  x : Float;
begin
  x := (a + b) / c;
  if x <= threshold and x >= -threshold then
    return x;
  elsif x > threshold then
    return threshold;
  else
    return -threshold;
  end if;
end;```

• Maybe verifying this function is not so simple after all!
A Simple Example of Functional Verification in SPARK\textsuperscript{[8,9]}

- SPARK is a programming language & associated set of verification tools
- Subset of Ada
- Performs several types of analyses, including functional verification
  - Absence of runtime errors (robustness)
  - Code satisfies user-specified low-level requirements written as contracts, e.g. pre- & post-conditions (compliance)
  - Low-level requirements are consistent
  - Certain types of low-level requirements are complete
- Functional verification achieves full coverage over all specified inputs

**Specification:**
Low-Level Requirements as Pre- & Post-Conditions

```haskell
function foo(a, b, c, threshold : Float) return Float with Pre => a > -Float'Last/2.0 and a < Float'Last/2.0 and b > -Float'Last/2.0 and b < Float'Last/2.0 and c >= 1.0 and threshold > 0.0, Post => foo'Result <= threshold and foo'Result >= -threshold;
```

**Code Implementation**

```haskell
function foo(a, b, c, threshold : Float) return Float is
    x : Float;
    begin
        x := (a + b) / c;
        if x <= threshold and x >= -threshold then
            return x;
        elsif x > threshold then
            return threshold;
        else
            return -threshold;
        end if;
    end;
```

A Simple Example of Dependency Contracts in SPARK\cite{8,9}

- SPARK can also perform analysis of dependency contracts

- Data dependency contracts
  - What global data is read and/or written
  - All read global data is initialized
  - All written global data is used

- Flow dependency contracts
  - Which output variables depend on which input variables

- Contracts help specify the architecture & low-level requirements

- Verifies all data is traceable to the architecture & low-level requirements

**Specification for a Procedure with a Data & Flow Dependency Contract**

```plaintext
procedure Append_To_Stream(Message : in String;
                           Status : out Boolean)
with Global => (In_out => Message_Stream),
     Depends => (Message_Stream => (Message_Stream,
                                      Message)),
     Status => (Message_Stream,
                Message));
```
DO-333 Certification Case Studies\[^{10}\]

- Formal methods applied to flight control system (FCS) with redundant flight guidance systems (FGSs)
  - Theorem Proving
  - Formalized system, high-level requirements (HLR) & architecture for FGS switching logic
    - System requirements: (1) Only one FGS active, (2) At least one FGS active, (3) Pressing transfer switch switches FGS ...
    - Architecture: Communications bus, clocks, switch, FGS
    - HLR: initial values & state update equations for all components
  - Proved completeness & consistency of HLR & architecture, compliance with system requirements
  - Abstract Interpretation
    - Analyzed for runtime errors (robustness) & unreachable code (traceability) in C code generated by Simulink model of Heading Control Law

- Formal methods applied to flight control system (FCS) with redundant flight guidance systems (FGSs)
  - Model Checking
    - Formalized HLR and low-level requirements (LLR) for FGS logic for selecting lateral & vertical mode
      - HLR: (1) At least one vertical mode active, (2) At most one vertical mode active, (3) Vertical Approach cannot be active until Lateral Approach has become active ...
      - LLR: Simulink/Stateflow models of FGS mode logic
    - Proved compliance of LLR with HLR
  - Model Checking
    - Formalized HLR and low-level requirements (LLR) for FGS logic for selecting lateral & vertical mode
      - HLR: (1) At least one vertical mode active, (2) At most one vertical mode active, (3) Vertical Approach cannot be active until Lateral Approach has become active ...
      - LLR: Simulink/Stateflow models of FGS mode logic
    - Proved compliance of LLR with HLR

- Models & longer report publicly available from NASA

\[^{10}\] Models & longer report publicly available from NASA
Formal Methods for Autonomous Systems

• **Need automation to help build autonomy!**

• Current work has focused on source code & low-level requirements, but still need to
  • Integrate these methods into existing workflows better
  • Increase education & provide good case studies

• Autonomy requirements & architecture will be more complex, so we & others have
  • Developed tools for architectures, LLR & HLR\textsuperscript{[11,12]}
  • Used tools to find errors in an autonomous protocol for controlling teams of unmanned air vehicles\textsuperscript{[13]}
  • Developed tools to automate design from LLR, HLR & architectures, e.g. synthesize code skeletons or source code\textsuperscript{[14, 15, 16]}
What are Key Challenges your Field Thinks About?

• Formalizing to enable automated rigorous analysis
  • Richer requirements: more focused on safety & high-level goals, possibly re-evaluated by system online
  • Flexible architectures: aid decomposition, enable "plug & play" composability, yet still provide strong evidence

• Handling complexity
  • Horizontally (compositional verification)
  • Vertically (end-to-end verification)
What Advances are Needed?

• Scalability
  • Through the right decompositions (art)
  • Practical ways to use human intuition to guide proof
  • Using machine learning to speed up analysis

• Usability
  • Incorporating formal methods into practical workflows
  • Enabling iterative processes
  • Increasing education
Supplementary & Alternative Approaches

Hybrid Systems

- Hybrid systems have both discrete logic & continuous dynamics
- “Autonomy in motion”
- Tools to check whether hybrid system can reach unsafe state from initial conditions[^17]

Runtime Assurance (RTA)[^20]

- Some systems too complex to verify with sufficient confidence (formally or informally)
- Develop simpler but verifiable “safe controller” as fallback
- Requires “monitor” to know when to switch
- Caveats:
  - Interactions between RTA controllers can result in unexpected actions
  - Developing correct monitors can be challenging

Assurance Cases[^18, 19]

- No “one size fits all” approach to TEV&V of autonomous systems
- Assurance cases: “Structured arguments supported by evidence that a system is safe in a given environment”
- Approach & tools to allow designers to build custom but rigorous TEV&V arguments

Where our Group is Going

Unmanned Systems Autonomy Services (UxAS)\(^{[21]}\)
- Service-oriented framework for mission-level autonomy for teams of unmanned air vehicles
- Use as case study for formal methods
- Re-coding & verifying services in SPARK (LLR)
- Want to formalize HLR & architecture, then develop workflow to automate parts of design
- Want to address cyber security concerns through crypto libraries, taint analysis with SPARK

Verified Algorithm Implementations
- Formalize requirements for core algorithms used in autonomy: branch & bound, search trees, etc.
- Develop verified algorithm implementations, similar to certified machine learning algorithm in \([22]\)
- Automatically synthesize verified implementations from requirements as an extension to \([15]\)

Air & Space Collision Avoidance
- Formalize requirements for air & space collision avoidance
- Develop faster approaches for hybrid systems analysis
- Use a mix of formal methods & more traditional methods like modeling & simulation to design collision avoidance systems
- Consider cyber security concerns

Assurance Cases
- Develop tools to graphically represent assurance cases & organize evidence electronically
- Apply to problems of interest
Successes of Formal Methods

Airbus A380 & A400M software modules[23]
- Determined WCET & max stack usage (required minimal expertise)
- Proved functional correctness (more expertise & manual effort)

Dassault-Aviation flight control software[7]
- Replaced software robustness testing with deductive verification & abstract interpretation
- Saved person-month of effort per software release

Amazon Web Services[25,26,27]
- Developed cloud computing tools to
  - Verify user access control polices meet user security requirements
  - Determine if computers are reachable from outside world
  - Automated continuous formal verification of s2n TLS library

DARPA High Assurance Cyber Military Systems (HACMS)[28]
- Proved cyber security guarantees with “safe” programming languages, formalized architecture & seL4
- Secured Boeing’s Unmanned Little Bird (ULB) helicopter

seL4 microkernel[29]
- Proved security properties like integrity & confidentiality
- Verified functional correctness of kernel capabilities in source & object code

Collins Aerospace[30]
- Aircraft systems status display
  - Formalized requirements
  - Auto-generated source code from requirements
  - Verified compliance of source code with requirements
  - Saved 10-15 weeks of labor
- Crew alerting system
  - Auto-generated test cases for coverage from requirements

Microsoft[24]
- Developed Static Driver Verifier (SDV) tool to verify third-party drivers
- Verifying HTTPS ecosystem
References

Publications


