Formal Methods Applied to Safety-Critical Systems

Alwyn Goodloe
a.goodloe@nasa.gov
NASA Langley Research Center

NASA R&D in Formal Methods

- NASA Langley Research Center (LARC) - Safety Critical Avionics Branch
- NASA Ames Research Center(ARC)– Robust Software Engineering Group
- Jet Propulsion Laboratory (JPL/FFRDC) – Laboratory for Reliable Software
- NASA Marshall Spaceflight Center, NASA Kennedy Spaceflight Center, and NASA Johnson Spaceflight Center have efforts applying model checking to small projects, but I don’t discuss these

Focus of Talk

- LaRC efforts in formal methods will be focus of today's talk
- A brief overview of JPL efforts that may be of interest to SDP
- ARC's work was presented at recent SDP meeting so I will mainly highlight collaborative efforts
- LaRC has historically targeted unltra-reliable safety-critical systems in aerospace
  - Heavily regulated
  - Very long development times
  - Safety trumps cost/time to deliver

NASA Langley

- LaRC created in 1917 as the first National Advisory Committee for Aeronautics (NACA) research facility
  - Located in Hampton, Virginia
- LaRC became a NASA lab in 1958
  - The Mercury program begin at LaRC
- Research areas of concentrations: Aeronautics, Atmospheric Sciences, and Exploration
- Formal methods research at LaRC is conducted in the Safety-Critical Avionics Systems Brach of the Research and Technology Directorate (RTD)

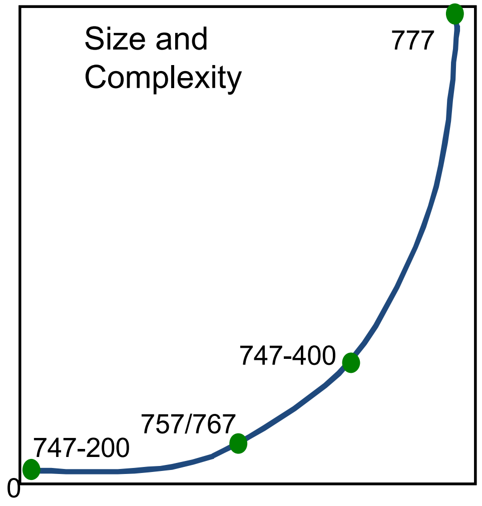
Ultra-Reliability is Hard

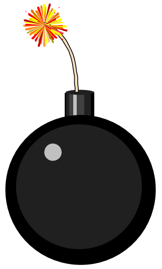
We are very good at building complex software systems that work 95% of the time---but, we do not know how to build complex software systems that are ultra-reliably safe.

What then has saved us in the past?

- –minimal amount of software that is safety-critical
- –simple designs
- –enormously expensive verification and certification processes
- – backups that are not software, e.g.:
  - ° hardware interlocks
  - ° human intervention

All sectors of aerospace are increasingly relying on software to perform safety-critical functions





Branch Mission

Safety-Criticial Avionics Systems:

Research, create, and demonstrate new methodologies and tools for designing, verifying, validating, and assuring high confidence software-intensive systems to improve safety, reliability, and capacity in mission- or life-critical aerospace systems

Analyzing Designs and Algorithms

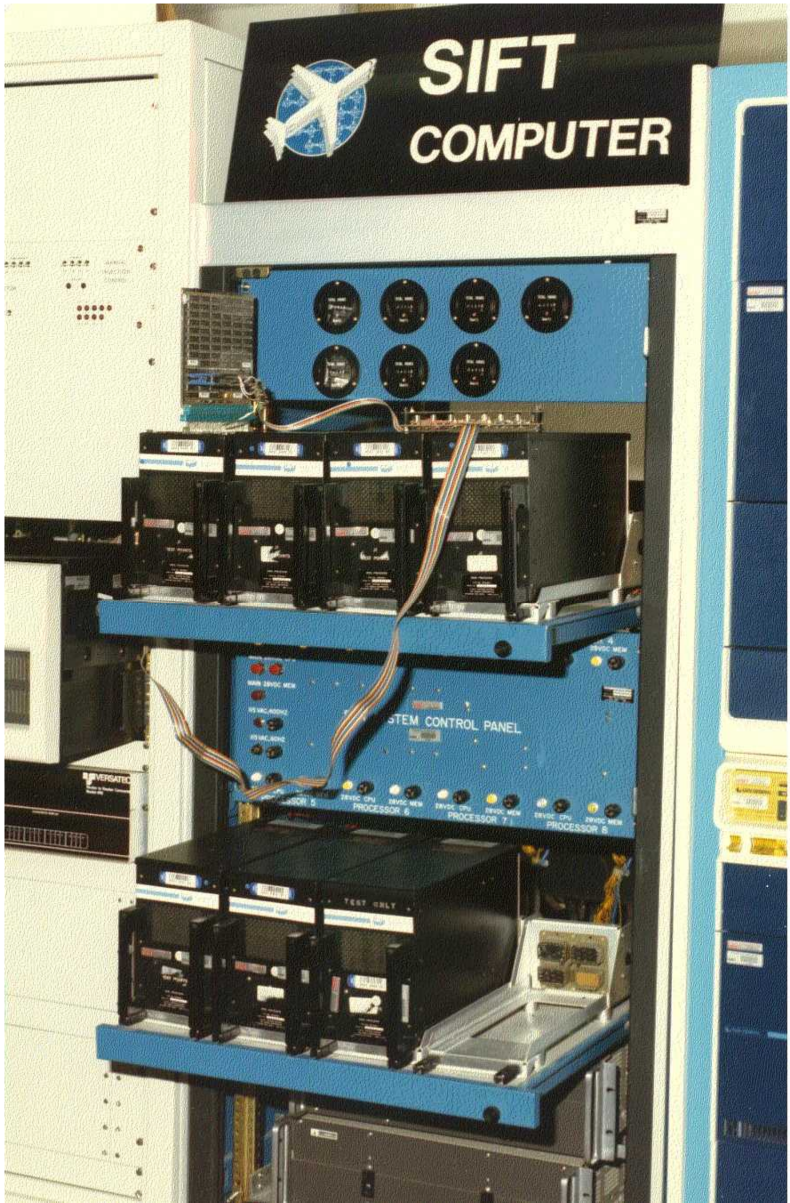
- Avionics code is very conservative and testing far exceeds almost any other software
  - Buffer overflows are not the problem here
- Problems often stem from the physically possible, but logically unanticipated
  - How does software respond to unanticipated hardware failures
- LaRC has traditionally focused on design and algorithm analysis rather than code
  - More code analysis recently
- Many models involve continuous math
  - Cannot just abstract this away
  - Interactive theorem proving is often the only formal tool we can use
    - SMT solvers and model checkers used when appropriate
    - Developing new decision procedures for nonlinear arithmetic

LaRC Early Pioneer

- Historically LaRC focus has been on formal methods for analyzing avionics
- Safety-critical distributed systems
- In late 1970s there was a contract in place with SRI International and Bendix to build a fault-tolerant computer named SIFT: Software Implemented Fault Tolerance
- And a second contract with SRI to formally prove the SIFT operating system correct

SIFT Computer

- Reliability goal: 10^{-9}
- 6 processors
- Fully-connected topology
- Fault-tolerant clock synchronization
- Byzantine agreement algorithm
- Delivered to NASA Langley in 1981
- Contributers include: Jack Goldberg, Chuck Weinstock, Karl Levitt, Michael Melliar-Smith, Richard Schwartz, Rob Shostak, Bob Boyer, J. Moore, John Wensley, Leslie Lamport



Landmark Accomplishments

- Although the verification of the entire OS was overly ambitious
- Some landmark accomplishments had been made:
  - Fault-tolerant clock synchronization
  - Byzantine Agreement
  - An insightful problem decomposition:
    - Prob[enough hardware] via Markov analysis
    - Enough hardware → good answers
    - Hierarchical decomposition
  - Shostak decision procedures → EHDM prover → PVS

Later Recognized Successes:

- Rockwell Collins/SRI Verification of AAMP5/AAMP-FV μPs (Srivas, Miller)
- Proved microcode of one instruction in each instruction class of their new AAMP5
- Errors found:
  - Discovered two errors during specification
  - Proofs systematically uncovered two ``seeded'' errors
- There were four engineers at Collins that were skilled in formal methods
- In fall 1996 Rockwell Collins hired a formal methods expert whose full-time job is to integrate the use of formal methods into their product lines

Honeywell Technology Center (Minneapolis) with SRI International

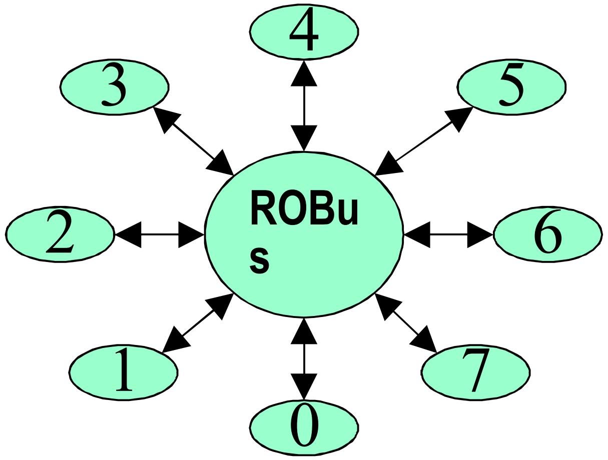
GOAL: Develop and implement verification techniques for demonstrating safety of IMA software using the DEOS operating system as the test subject

Primus Epic

- DEOS is a partitioned real-time operating system used in Honeywell’s Primus Epic developed to DO-178B Level A certification standards
- In parallel with the research tasks, the verification integrated into the DEOS certification process

SPIDER

- SPIDER: Scalable Processor-Independent Design for Electromagnetic Resilience
- Built upon 20 years of fault tolerance research at LaRC
- Co-funded by FAA and NASA Langley
- Inhouse project
- GOALS: Develop fault-tolerant computer architecture in accordance with RTCA DO-254 guidelines:



  – demonstrate feasibility of formal methods as means of certification
  – develop training materials for FAA
  – develop advanced fault-tolerant computer architecture platform for in-house analysis and experimentation

SPIDER'S ELECTROMAGNETIC RESILIENCE

- Recovery from normal transients or permanent faults is guaranteed by the formal design verification
- Lab testing confirms that the *assumptions* used in the design and proofs are valid
- Recovery from massive upset is not guaranteed mathematically, but
  - PE's can be restarted once the SPIDER ROBUS has recovered
  - SPIDER ROBUS can be internally protected with shielding (small size can help reduce weight)

Honeywell Engines and Systems with TTTech and SRI International

GOAL: Develop Fault Tolerant Integrated Modular Architecture design, validation, and implementation technologies for deployment in next-generation engine controls for commercial aircraft

APPROACH: Use TTTech's Time Triggered Architecture (TTA) developed in Europe for the automotive industry and formal verification methods (SRI) to develop a FTIMA architecture. Targeted application is Full Authority Digital Engine Control (FADEC)

Time-Triggered Technology has been developed over the past fifteen years at Vienna University of Technology. It was refined in co-operation with leading industrial pacesetters. Provides:

Composability
Predictable temporal behavior
Diagnosability and Testing
Reusability of Components
Fault-tolerance

Formal Models of Distributed Avionics

- Integrated analysis of TTEthernet using SAL model checker and PVS (SRI)
- Architecture Analysis and Design Language (AADL) models of synchronous and asynchronous systems (Honeywell and WWTechnologies)
  - Can we establish a basis for comparison
- Model based testing of distributed avionics systems (Honeywell)

DO-178C Formal Methods Supplement

- FAA must certify aircraft before they are allowed to fly
- RTCA standard DO-178C governs software
- New formal methods supplement allows the use of formal methods in place of some, but not all, testing
  - Approved by committee as DO-333
- LaRC engineers have played a critical role in getting this approved

Expanding Portfolio

- In recent years we have added new people to the group with new skill sets
  - Model checking, SMT solving, static analysis, etc.
- Expanded the targeted application areas to include
  - Airspace management
    - Traditionally done by simulation
    - FM and simulation people now working together
  - Static code analysis
  - New decision procedures

Generating Java Code From PVS

- LaRC has designed and proven correct a considerable number of algorithms using SRI’s Prototype Verification System (PVS)
- Customers often want executable prototypes
- LaRC has an ongoing effort to build a system that translates a subset of PVS into Java
  - Removes tail recursion
  - Semantic attachments can replace PVS functions with Java library calls
  - Produces JML assertions and invariants from PVS spec that can be used to verify the generated code
  - Collaborative effort with ARC to generate test cases

Software Change Management Research

- Develop novel techniques to preserve and improve the integrity of software as it changes over time
  - Change impact analysis techniques generally estimate program differences based on source level differences
    - Results may over-estimate or under-estimate the *effect* of changes because there is insufficient information to accurately compute the impact of the change

What are the ***effects*** of changing this code…

```
public boolean detection(Vect3 s, Vect3 vo, Vect3 vi,
      double D, double H, double B, double T) {
    t_in  = 0;
    t_out = 0;
    if (T >= 0 && B >= T) return false;
    Vect2 s2  = s.vect2();
    Vect2 vo2 = vo.vect2();
    Vect2 vi2 = vi.vect2();
    double vz = vo.z-vi.z;
    if (vo2.almostEquals(vi2) && Horizontal.almost_horizontal_los(s2,D)) {
      if (!Util.almost_equals(vo.z,vi.z)) {
        t_in  = T < 0 ? Math.max(Vertical.Theta_H(s.z,vz,Entry,H),B) :
          Math.min(Math.max(Vertical.Theta_H(s.z,vz,Entry,H),B),T);
        t_out = T < 0 ? Math.max(Vertical.Theta_H(s.z,vz,Exit,H),B) :
          Math.max(Math.min(Vertical.Theta_H(s.z,vz,Exit,H),T),B);
      } else if (Vertical.almost_vertical_los(s.z,H)) {
        t_in  = B;
        t_out = T;
      }
    } else {
      Vect2 v2 = vo2.Sub(vi2);
      if (Horizontal.Delta(s2,v2,D) > 0) {
        double td1 = Horizontal.Theta_D(s2,v2,Entry,D);
        double td2 = Horizontal.Theta_D(s2,v2,Exit,D);
        if (!Util.almost_equals(vo.z,vi.z)) {
          double tin  = Math.max(td1,Vertical.Theta_H(s.z,vz,Entry,H));
          double tout = Math.min(td2,Vertical.Theta_H(s.z,vz,Exit,H));
          t_in  = T < 0 ? Math.max(tin,B)  : Math.min(Math.max(tin,B),T);
          t_out = T < 0 ? Math.max(tout,B) : Math.max(Math.min(tout,T),B);
        } else if (Vertical.almost_vertical_los(s.z,H) ) {
          t_in  = T < 0 ? Math.max(td1,B) : Math.min(Math.max(td1,B),T);
          t_out = T < 0 ? Math.max(td2,B) : Math.max(Math.min(td2,T),B);
        }
      }
    }
    return t_out < 0 || t_in < t_out;
  }
```

$\Rightarrow$

…on how this operates?

Software Change Management Research

- Our approach: Use the results of inexpensive source code differencing techniques to guide more precise techniques to explore and characterize the impact of changes
  - Goal: Avoid exploring unchanged program execution behaviors to control analysis cost
    - Differential Symbolic Execution (DSE): Use over-approximating summaries of unchanged sections of code when applying more precise techniques
    - Directed Incremental Symbolic Execution (DiSE): “Prune” the (symbolic) execution space when it does not contain affected behaviors

Software Change Management Research

- Both techniques compute a summary of the affected program behaviors
  - Symbolic summaries characterize program behaviors in terms of constraints on the program inputs
- Use decision procedures to analyze and compare summaries
- Use summaries to direct more expensive software testing and verification techniques to analyze the parts of the program affected by the changes

Software Health Management

- Complexity of fielded systems means that it may not be possible to exhaustively test and verify all software
- Runtime verification (RV) - is a computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors satisfying or violating certain properties
  - Properties often expressed in past-time temporal logic
  - Very exciting area of research for formal methods community

NASA Support for RV

- ARC has been a pioneer in the area
- JPL (Havelund) – Applying RV to robotic missions
- Research grants to support work in RV applied to avionics
  - UIUC (G. Rousu) – Monitoring-Oriented Programming
  - SRI (J. Rushby) – Reliability via possibility perfect monitors
  - Galois (L. Pike) – Sampling approach targeting hard real-time
    - Copilot Haskell EDSL
  - RICAS (J. Shuman) – Baysian networks

NASA PVS Libraries

- LaRC maintains and develops an extensive library of PVS theories
- Representative Examples:
  - Basic Mathematics: algebra and trigonometry
  - Not So Basic: logarithms, exponentials and hyperbolic
  - Calculus: Series, Integration
  - Discrete structures: arrays, sequences
  - Probability
  - Linear Algebra
- Aimed mainly at verification of safety-critical cyber-physical systems
  - Driven more by engineering applications than computer science problems

Numerical Software Verification

- Floating point numbers are not real
  - Approximation creates well-known anomalies
  - Safety-critical numerical software needs to be built carefully
- Deductive verification of numerical software
  - In some cases, can prove absence of errors
  - Otherwise, want to prove errors fall within bounds
  - Verification often possible but usually difficult
- Research goals:
  - Apply Bernstein polynomial techniques
  - Develop tools and techniques to verify properties of floating point computations
  - Aim for high degree of automation

Non-Linear Arithmetic

- Heart Dipole Problem:
- $P(x_1,\ldots,x_8) = -x_1x_6^3+3x_1x_6x_7^2-x_3x_7^3+3x_3x_7x_6^2-x_2x_5^3+3x_2x_5x_8^2-x_4x_8^3+3x_4x_8x_5^2-0.9563453$
- $x_1\in[-0.1,0.4], x_2\in[0.4,1], x_3\in[-0.7,-0.4], x_4\in[-0.7,-0.4], x_5\in[0.1,0.2], x_6\in[-0.1,0.2], x_7\in[-0.3,1.1], x_8\in[-1.1,-0.3]$
- Theorem: $\forall x: p(x_1, \ldots, x_8) \geq -1.7435$
- Theorem: $\exists x: p(x_1,\ldots,x_8) \leq -1.7434$

Motivating Better Tools

- Inability to handle nonlinear arithmetic is a serious issue with many formal methods tools (SMT solvers, hybrid model-checking)
- Automatic verification of algorithms that compute with real numbers
- Code-level verification of algorithms that compute with floating-point numbers
- Verifying reliability and stability in control systems

Existing Approaches

- Existing approaches for verification of non-linear arithmetic:
  - Quantifier elimination
  - Sum of squares
  - Numerical approximation
- None of these can solve Heart-dipole problem
  - Some not really practical efficiency wise

Bernstein

- A formal library in PVS for reasoning about (multivariate polynomials)
- Based on Bernstein polynomials
- Numeric constants are operated on using infinite-precision rational arithmetic
  - All results produced are free from numerical representation errors
  - Bernstein's results carry the weight of rigorously proved mathematical theorems.
- Proof strategies in PVS for automatically solving inequalities
- User friendly tools for formally solving global optimization

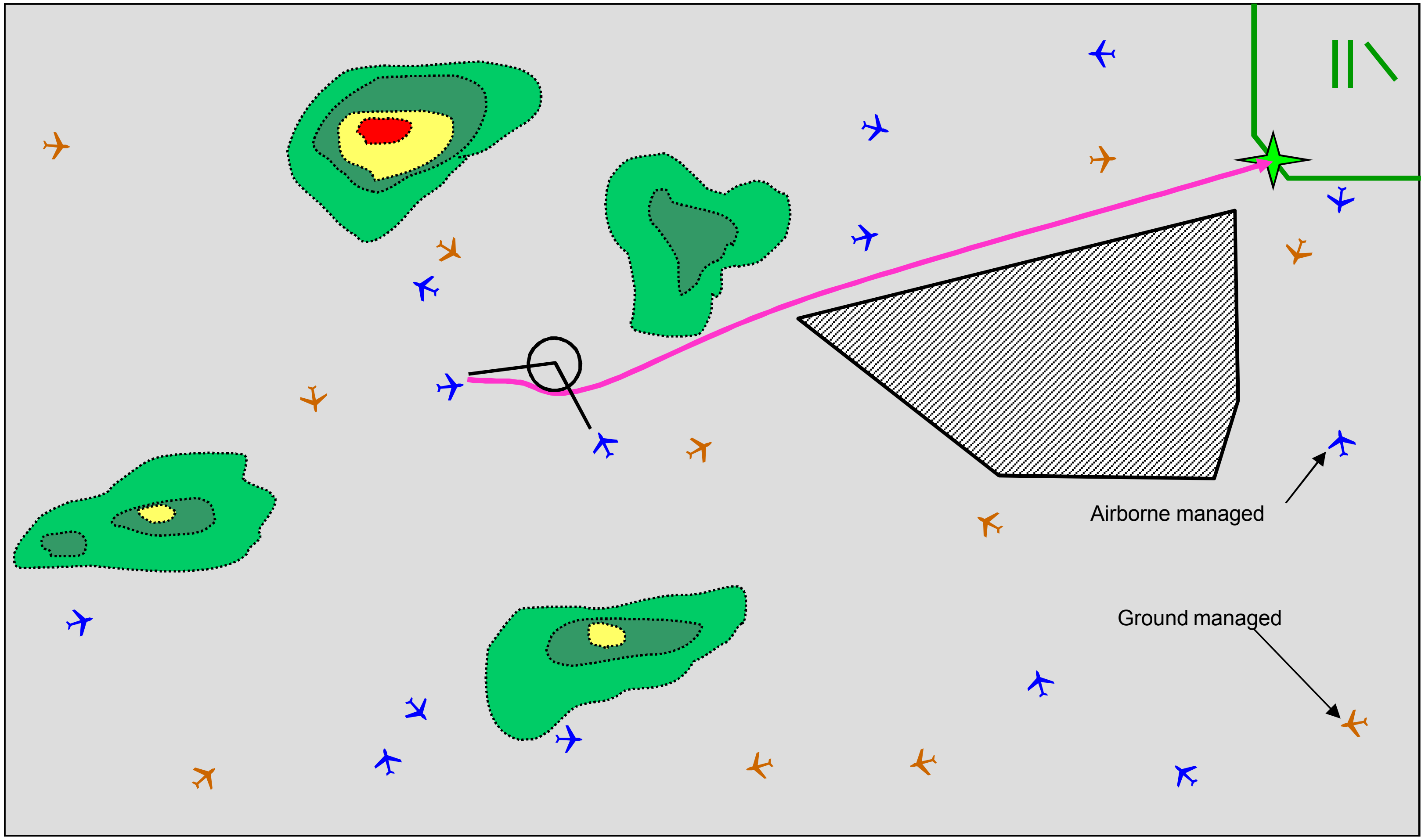
Kodiak

- A C++ library that implements Bernstein polynomial using an infinite precision arithmetic library (GiNaC/GMP)
- Intended for use in SMT solvers
- We are looking for for collaborators who wish to use library
  - May want to implement their own version

Aircraft Separation

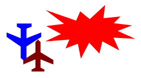
- As part of congressional mandate, as part of Joint Planning Development Office (JPDO) organization, NASA is responsible for looking at futuristic ATM concepts
- NASA is looking at a variety of air traffic management concepts to look at increasing capacity, efficiency, flexibility, etc.
- More controllers will not be able to achieve big gains in these parameters
  - Everything that NASA is looking at has a significant role for automation
    - Often new uses for automation
- More automation doesn't remove safety issues, but simply shifts the risk from people to automation
- NASA is interested in new ways to analyze the safety of air traffic automation

Self Separation Concept

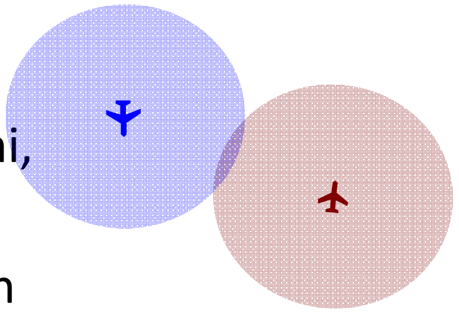


Separation and Automation

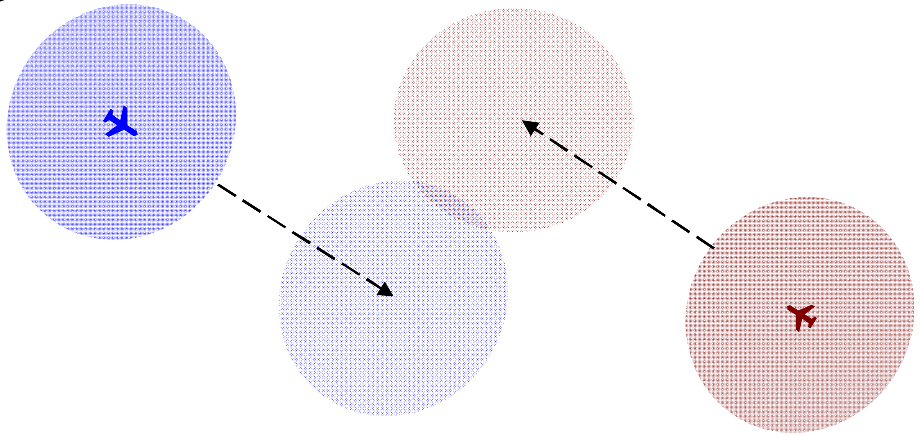
- Collision
  - Scrape paint
  - Avoid through pilot, controller, and TCAS



- Loss of Separation
  - Separation standards are violated (5nmi, 1000ft)
  - Avoid through human and/or automation decisions



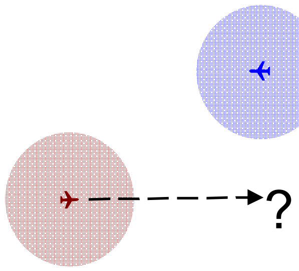
- Conflict
  - Predicted loss of separation



Separation Algorithms

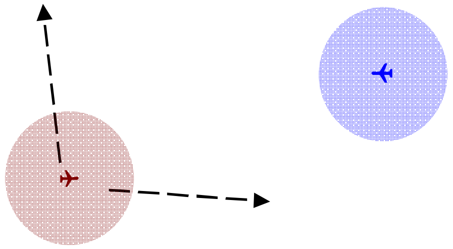
Conflict Detection

– Detect future loss of separation



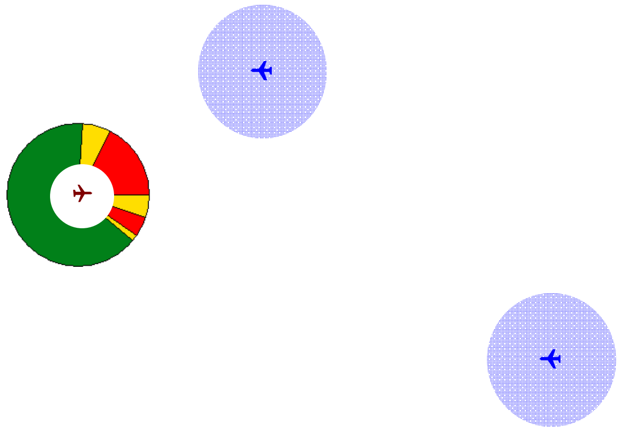
Conflict Resolution

– Suggest maneuvers to avoid a conflict



Conflict Prevention

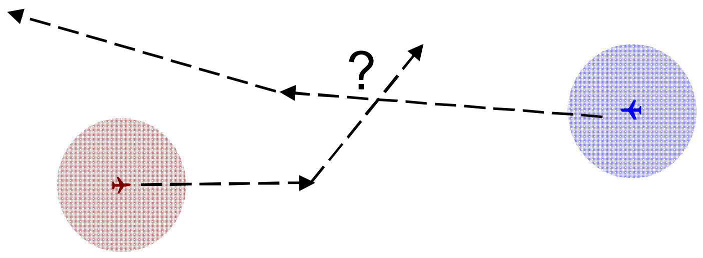
– Provide conflict-free maneuvers



Trajectory Algorithms

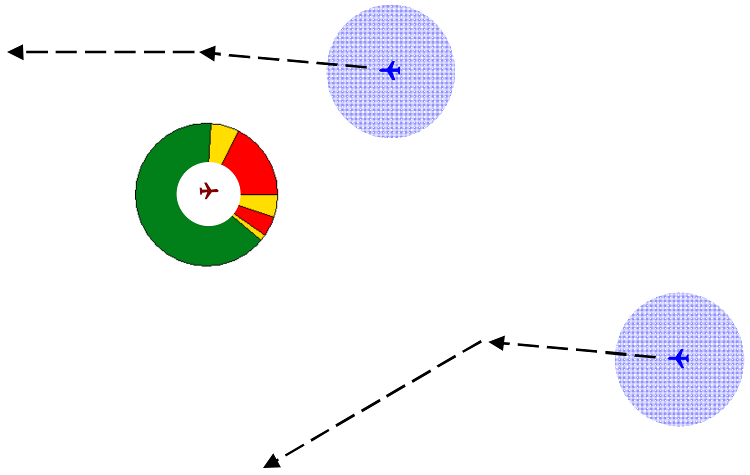
Conflict Detection

– Detect future loss of separation



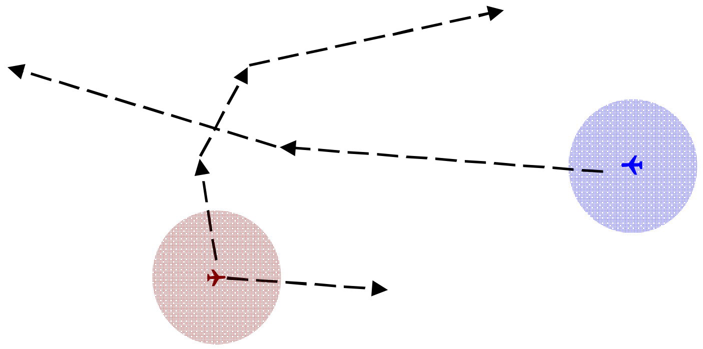
Conflict Prevention

– Provide conflict-free maneuvers



Conflict Resolution

– Suggest maneuvers to avoid a conflict



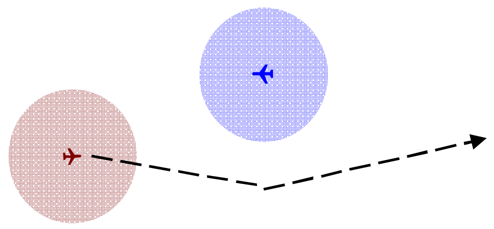
Recovery Algorithms

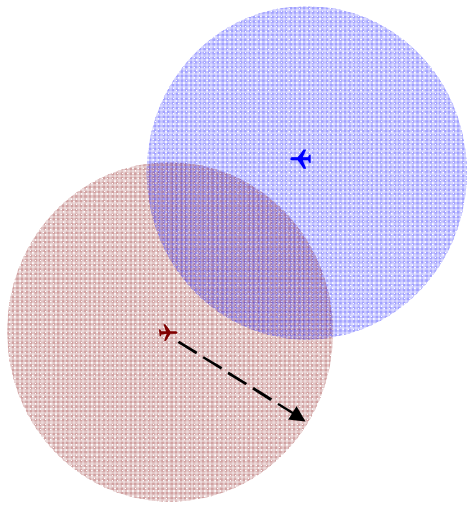
Conflict Recovery

- Suggest maneuvers to regain desired path

Loss of Separation Recovery

- For a variety of reasons separation may be lost
- Suggest a maneuver to regain separation



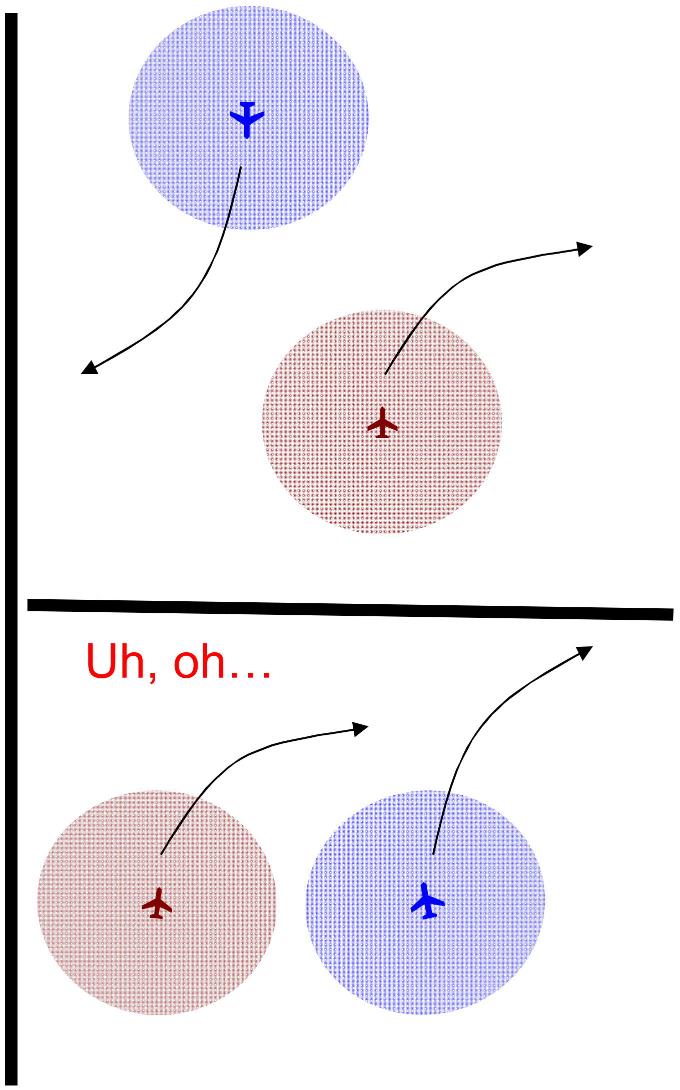


Research Goal

Develop a general formal framework for analysis of safety properties of these algorithms

Conflict Resolution

- Each aircraft determines its own set of six maneuvers
  - Go right/left, Speed up/slow down, Go up/down
- Properties
  - Independence: free of conflicts if one aircraft maneuvers
  - Coordination: free of conflicts if both aircraft maneuver
- Requirements
  - No specific comm between aircraft
  - No unfair rules: lower aircraft ID goes first, etc.



Formal Statement of Properties

```
independent: THEOREM
    precondition_ind?(s(a), s(b), v(a), v(b)) AND
    (nva = cr3d_vertical_speed(a,b) OR
     nva = cr3d_ground_speed(a,b) OR
     nva = cr3d_heading(a,b)) AND
   IMPLIES
     NOT conflict?(s(a),s(b),nva-v(b))

coordinated: THEOREM
    precondition_coord?(s(a), s(b), v(a), v(b)) AND
    (nva = cr3d_vertical_speed(a,b) OR
     nva = cr3d_ground_speed(a,b) OR
     nva = cr3d_heading(a,b)) AND
    (nvb = cr3d_vertical_speed(b,a) OR
     nvb = cr3d_ground_speed(b,a) OR
     nvb = cr3d_heading(b,a))
    IMPLIES
     NOT conflict?(s(a),s(b),nva-nvb)
```

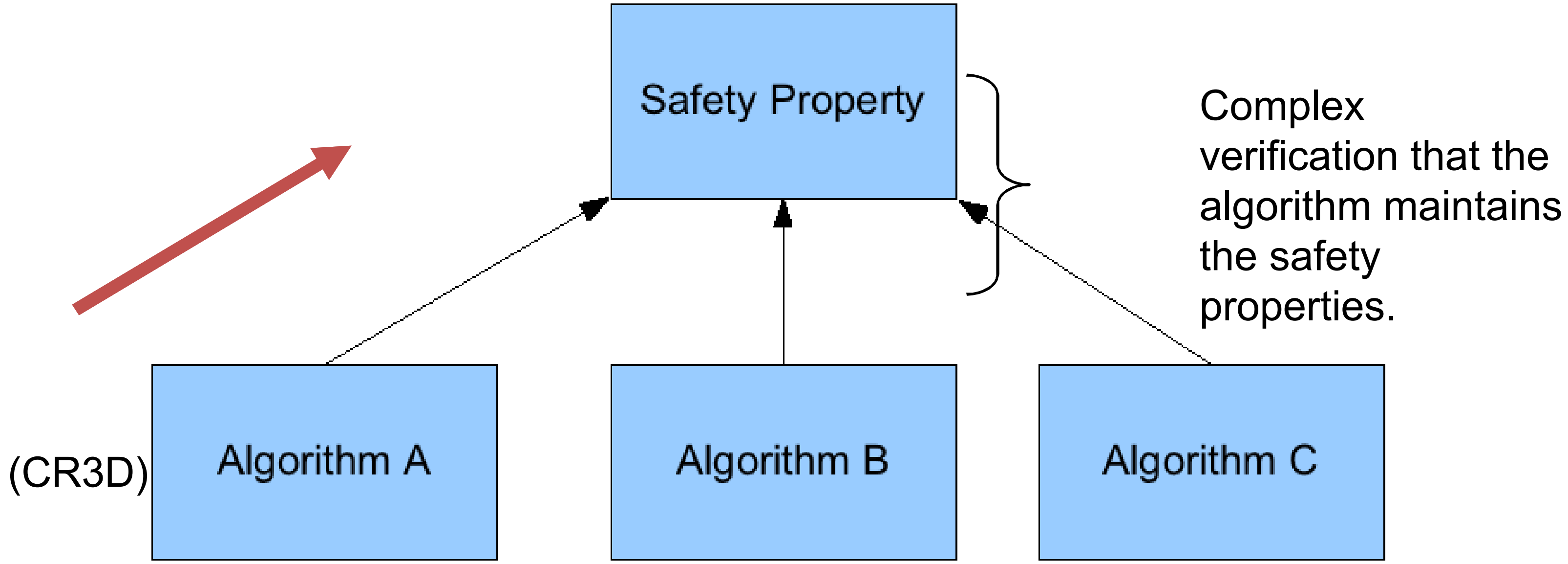
Formal Verification of Coordination

Begin by splitting the problem into nine cases…

		Aircraft B		
		Vertical	Ground	Track
Aircraft A	Vertical	Tricky	Easy	Easy
	Ground	Easy	Tricky	Tricky
	Track	Easy	Tricky	Tricky

… then prove each one, for all encounter geometries.

Algorithm Verification

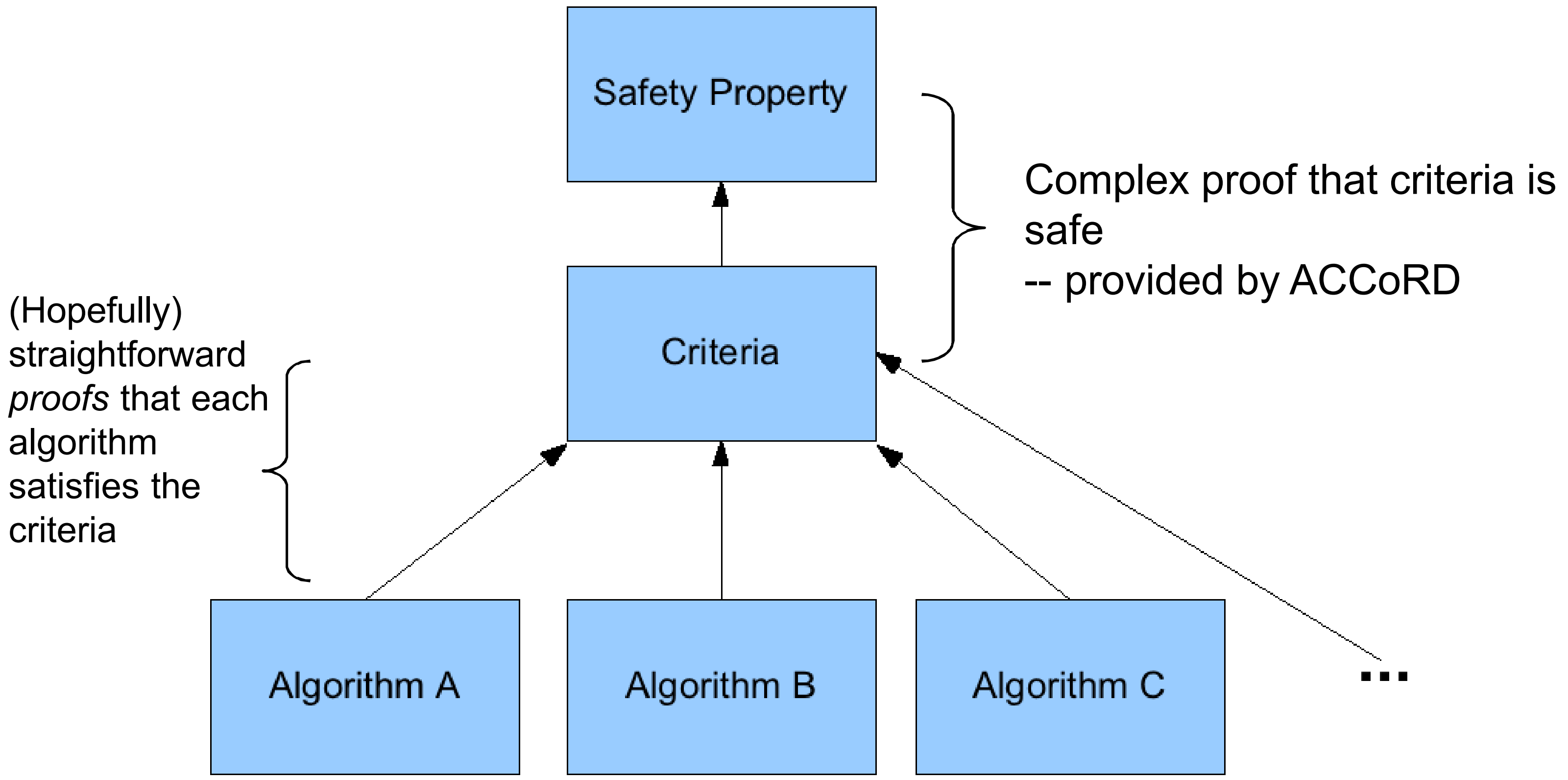


Complex verification that the algorithm maintains the safety properties.

How can we reuse these arguments?

ACCoRD Framework

Solution: ACCoRD – a verification framework for *classes* of separation algorithms


Safety Property
Criteria
Algorithm A
Algorithm B
Algorithm C
...
Complex proof that criteria is safe
-- provided by ACCoRD
(Hopefully) straightforward *proofs* that each algorithm satisfies the criteria

Criteria is Very General

- The criteria was developed to aid the verification process
- Criteria allows combinations of ground speed and vertical speed.
  - We have never looked at these algorithms before!
- But even more, if different algorithms satisfy the criteria, then they will be coordinated with each other
  - Self-separation does not rely on everyone running the same algorithm!

Using the Criteria

- Enables different airlines to fly different algorithms
  - and algorithms can evolve over time
- Requires an international agreement
  - Criteria embodies “rules of the road”
- Verification of individual algorithms easier
  - Hard work has been done in criteria framework
  - Need only prove that an algorithm satisfies the criteria

JPL Laboratory for Reliable Software

- JPL the engineers behind deep space robotic missions
  - Historically software built by domain experts
- Software bugs have caused a number of well publicised incidents resulting in either a loss of mission or near loss of mission
- LRS works to improve software engineering practices used on critical mission functions
- Composed of researchers in formal methods and software engineering

JPL Process

- A lab-wide coding standard focused on risk-related rules
  - Automated compliance verification
- A software developer certification process
  - Courses focused on SE principles and risk reduction
- A senior managers course, focused on software risk
- An emphasis on tool-based analysis (and not just people-based)
  - Including tool-based code review
  - Based on strong static source code analysis
  - Daily checks for coding-rule compliance
- Routine logic model checking for safety-critical parts of the design

Power of 10 Rules

- Restrict to simple control flow constructs
- Do not use recursion and give all loops a fixed upper-bound
- Do not use dynamic memory allocation after initialization
- Limit functions to no more than ~60 lines of text
- Use minimally two assertions per function on average
- Declare data objects at the smallest possible level of scope
- Check the return value of non-void functions; check the validity of parameters
- Limit the use of the preprocessor to file inclusion and simple macros
- Limit the use of pointers
- Compile with all warnings enabled, and use source code analyzers

Conclusion

- The safety-critical nature of aerospace systems make them a natural target for FM
- NASA Langley has been a pioneer in this area
  - Early research on fault-tolerance now in standard textbooks
  - Spurred use of formal methods by aerospace industry
- NASA Langley current focus on avionics and air traffic management
  - Areas where "good enough" is not good enough
  - Heavy-weight formal methods often needed when dealing with continuous math
    - But new decision procedures can help us make progress

URL Pointers

- http://shemesh.larc.nasa.gov/fm/index.html
- Look under the research page for topics and you should see pointers to papers
  - The fault-tolerance and separation assurance sections on the research page point to papers on those subjects
  - For the work on Bernstein polynomials see César Muñoz's page
  - For code difference papers see Suzette Person's page

Questions?