
Using julia as a Specification Language for the Next-Generation Airborne Collision Avoidance System (ACAS X)

Robert Moss
JuliaCon | 6/29/2015





Collaborating Organizations



Federal Aviation
Administration

 LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

 JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

Stanford | Intelligent Systems Laboratory

Carnegie Mellon University | Silicon Valley



National Aeronautics
and Space Administration



Collision Avoidance Background

1956 Grand Canyon Mid-Air Collision



Series of mid-air collisions in 1950s led to establishment of FAA in 1958

Need for Onboard Collision Protection Systems



San Diego, CA, 1978
144 fatalities



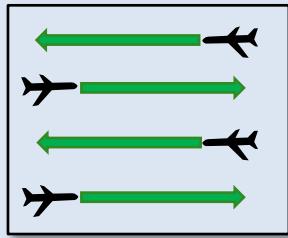
Cerritos, CA, 1986
82 fatalities

Provides independent safety net to protect against failures in:

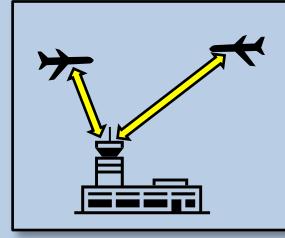
- Air Traffic Control intervention
- Pilot compliance with procedures
- Visual see-and-avoid



Evolving to a Layered Architecture



Strategic Separation
Airspace Design



Tactical Separation
Air Traffic Control

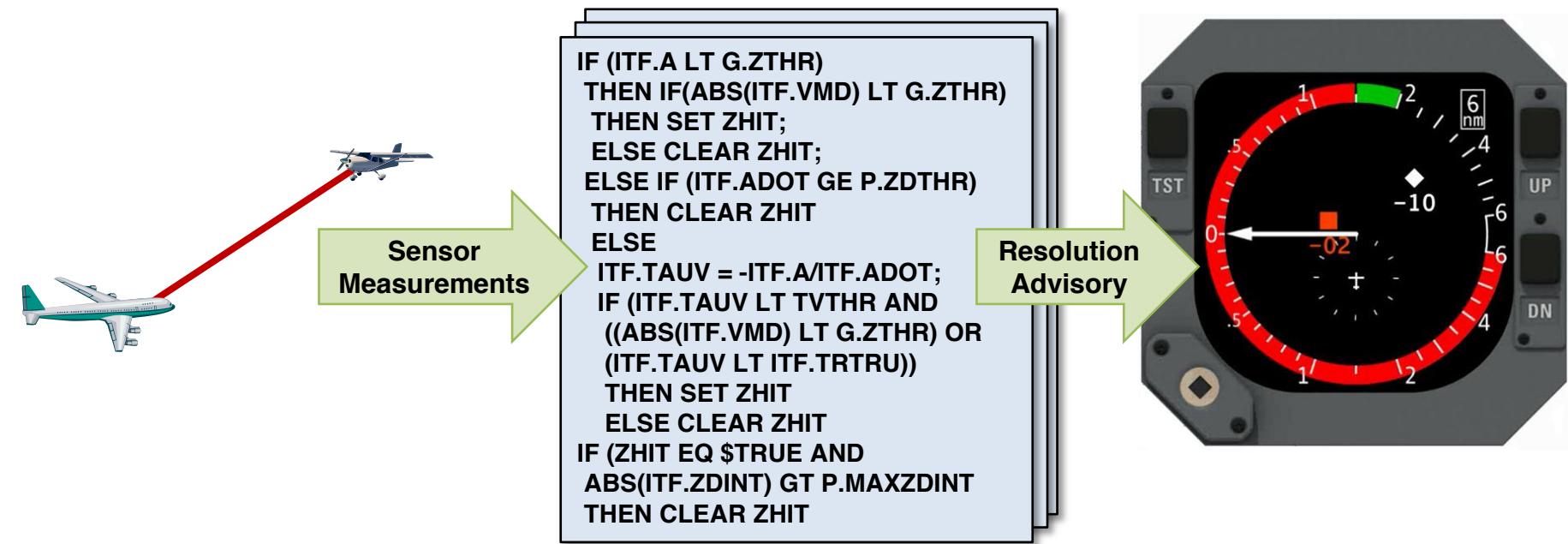
**Onboard
Collision
Avoidance**



Onboard collision avoidance is necessary to meet desired safety level



TCAS* Elements



Surveillance

- Intruder detection
- Position tracking

Advisory Logic

- Alert criteria
- Advisory selection

Display

- Aural annunciation
- Advisory display



TCAS Traffic Advisory



COUNTDOWN TO
NEAREST APPROACH **60** SECONDS





TCAS Resolution Advisory

COUNTDOWN TO
NEAREST APPROACH **46** SECONDS



Traffic... Traffic



TCAS Resolution Advisory

COUNTDOWN TO
NEAREST APPROACH **33** SECONDS





ACAS X Program Overview

Next-Generation Airborne Collision Avoidance System

- **Operational Benefits**

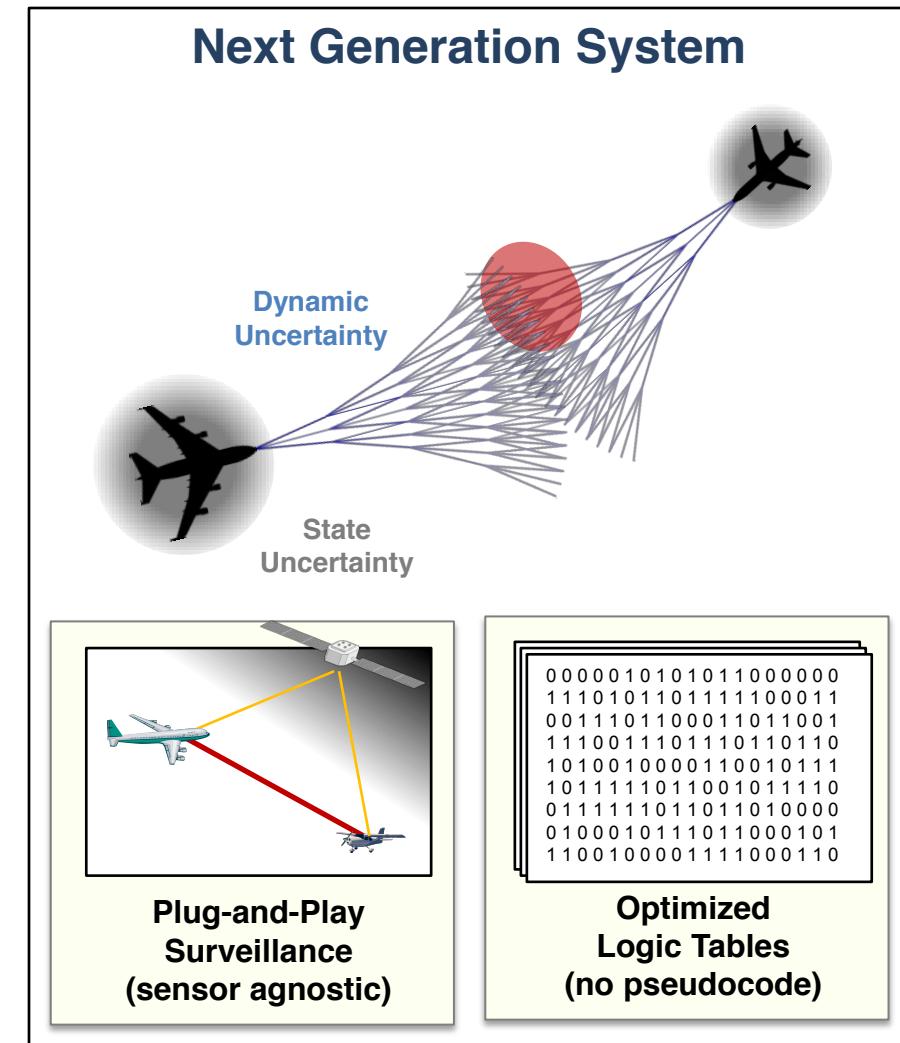
- Enables reduced separation
- Fewer unnecessary alerts
- Extends to all user classes
- Plug-and-play surveillance

- **Developmental Benefits**

- Faster process to update system
- Easier to adapt to changing airspace
- Reduced development and implementation burden

ACAS X supports NextGen airborne collision avoidance requirements

Next Generation System





ACAS X Program Overview

Next-Generation Airborne Collision Avoidance System

- **Operational Benefits**

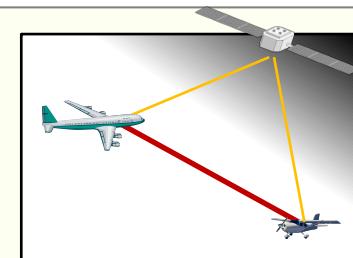
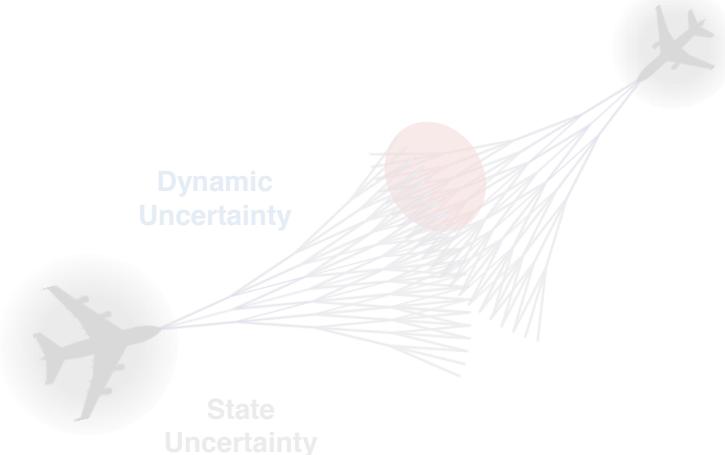
- Enables reduced separation
- Fewer unnecessary alerts
- Extends to all user classes
- Plug-and-play surveillance

- **Developmental Benefits**

- Faster process to update system
- Easier to adapt to changing airspace
- Reduced development and implementation burden

ACAS X supports NextGen airborne collision avoidance requirements

Next Generation System



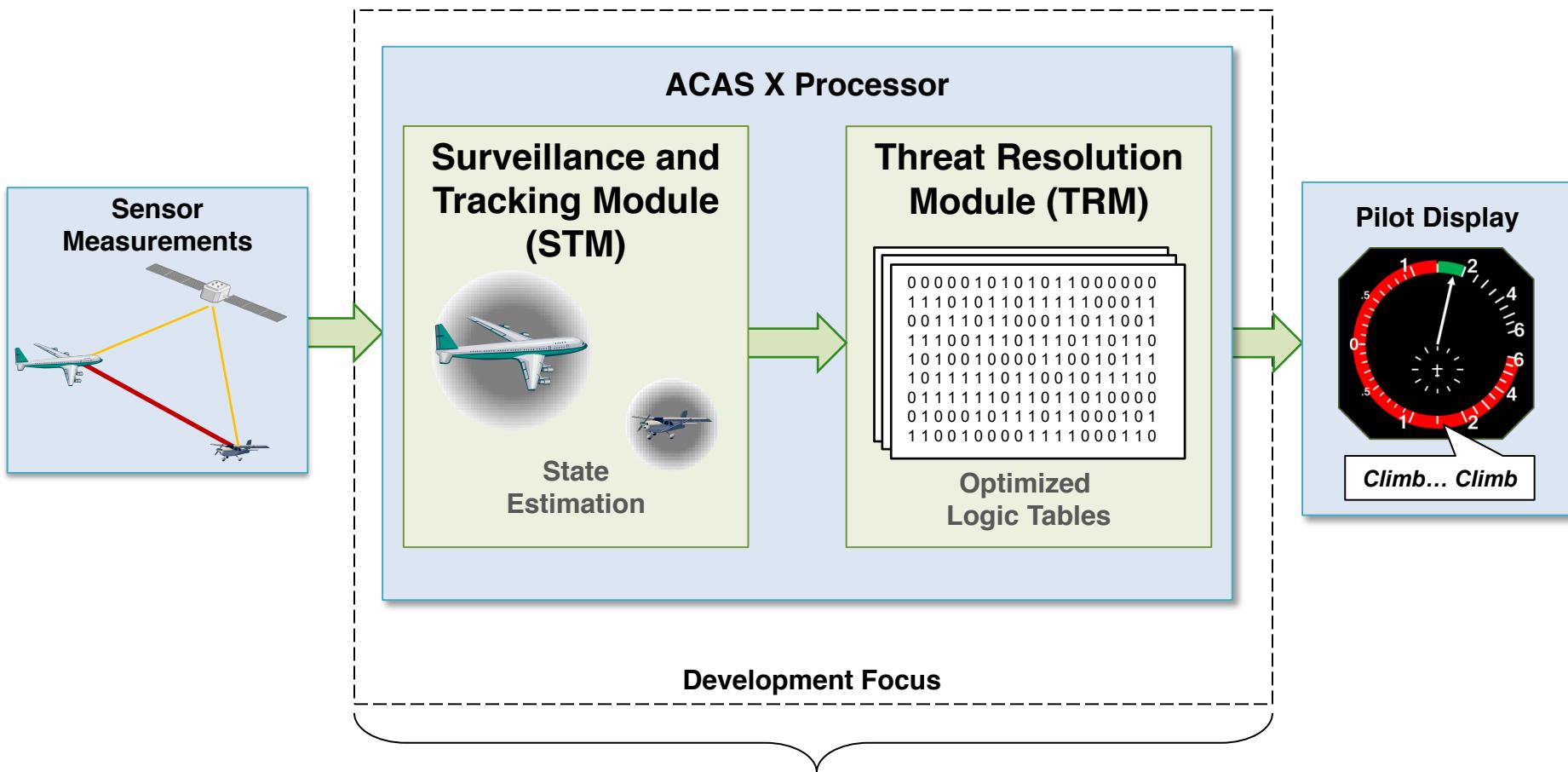
**Plug-and-Play Surveillance
(sensor agnostic)**

0000010101011000000
1110101101111100011
0011101100011011001
1110011101110110110
1010010000110010111
1011111011001011110
0111111011011010000
0100010111011000101
110010001111000110

**Optimized Logic Tables
(no pseudocode)**

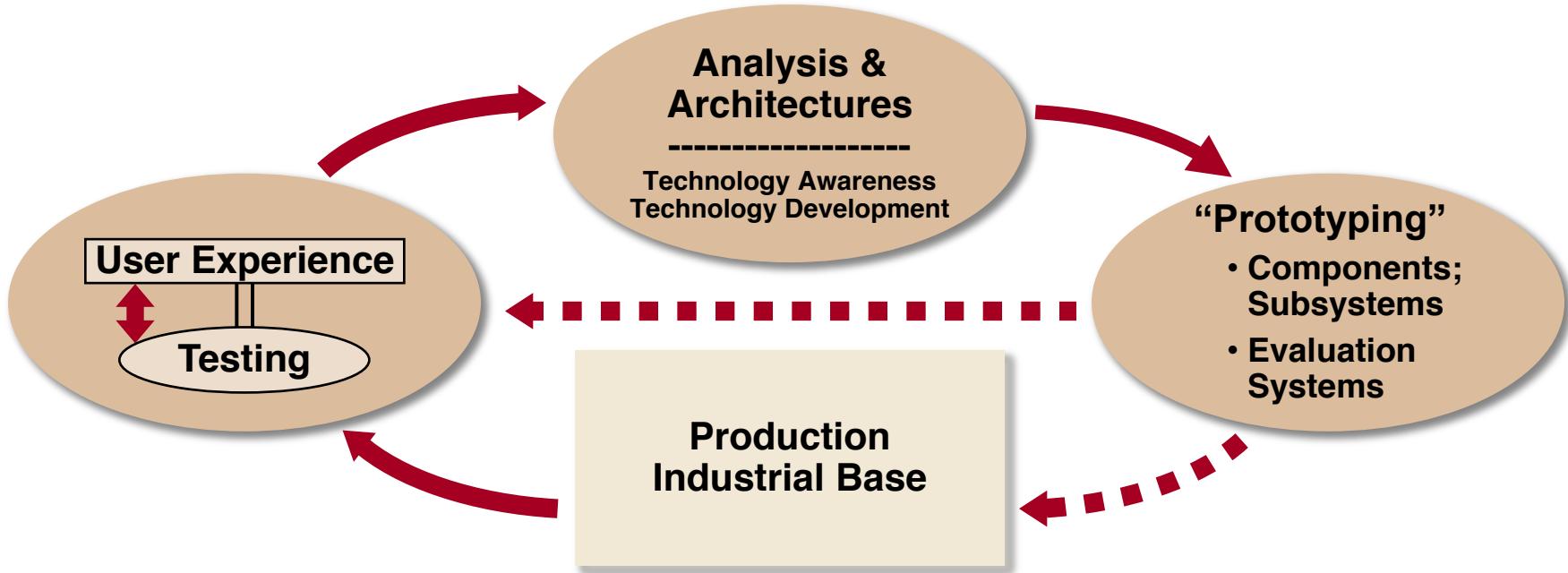


ACAS X Modular Architecture





MIT Lincoln Laboratory in the Development Cycle



- Historically, pseudocode and English used for software specification
- Historic process is costly and error prone
- Promoting Julia as a more cost effective and robust tech transfer path



Overview

- Background on CAS
- **CAS Specification**
 - **TCAS Specification Challenges**
 - **Algorithm Design Description (ADD)**
- Simulation Frameworks
- Verification and Validation Tools in Julia
- Summary

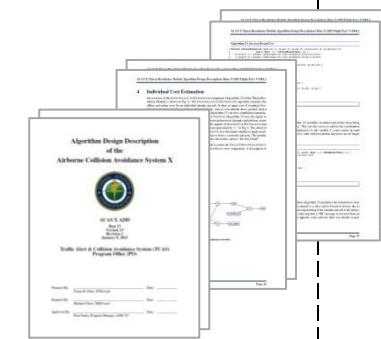


Two Main Specification Problems

1. Writing the spec.

- Improvements...
 - Minimize time taken away from development
 - Easily test and validation document

(LINCOLN LABORATORY)



2. Vendors implementing the spec.

- Improvements...
 - Minimize implementation time
 - Remove confusion of text and algorithms
 - Provide all necessary information

(Honeywell)





TCAS Algorithm Specification: Pseudocode

Variable Based Pseudocode

English Descriptive Pseudocode

```
PROCESS Proximity_test;
IF (range exceeds proximity threshold)
    THEN declare proximity test failed;
ELSEIF (track is non-mode C type)
    THEN IF (own altitude is above non-mode C cutoff)
        <assumes non-Mode-C intruders stay below ceiling allowed by ATC>
        THEN declare proximity test failed;
    ELSE IF ((neither a Traffic nor Proximity Advisory
              has been issued for this track) AND
              (either bearing or range coasted this cycle))
        THEN declare proximity test failed;
    ELSE declare proximity test passed;
OTHERWISE IF (relative altitude is within threshold)
    THEN declare proximity test passed;
    ELSE declare proximity test failed;
END Proximity_test;
```

```
PROCESS Proximity_test;

IF (ITF.R GE P.PROXR)
    THEN CLEAR PRXHITA;
ELSEIF (ITF.MODE EQ $FALSE)
    THEN IF (G.ZOWN GE P.ABOVNM)
        THEN CLEAR PRXHITA;
    ELSE IF ((ITF.TACODE NE $STANMC AND ITF.TACODE NE $PA) AND
              (ITF.BEAROK EQ $FALSE OR ITF.RFLG EQ $FALSE))
        THEN CLEAR PRXHITA;
    ELSE SET PRXHITA;
OTHERWISE IF (ITF.A LT P.PROXA)
    THEN SET PRXHITA;
    ELSE CLEAR PRXHITA;

END Proximity_test;
```

- Original representation of the TCAS logic is in pseudocode
- Within this representation, there are two *different* ways the pseudocode is written



TCAS Algorithm Specification: State Charts

Other Aircraft

Transition(s): Other_Traffic → Proximate_Traffic

Location: Other_Aircraft_{s-157} ▷ Intruder_Status_{s-261}

Trigger Event: Air_Status_Evaluated_Event_{e-C5}

Condition:

Alt_Reportings-193 in state Yes	OR	F	T
Other_Bearing_Valid _{v-173}		T	.
Other_Range_Valid _{v-169}		T	.
Proximate_Traffic_Condition _{m-417}		T	T
Potential_Threat_Condition _{m-415}		F	F
Threat_Condition _{m-415}		.	F

Output Action: Intruder_Status_Evaluated_Event_{e-C3}

Notes: 1. **Description:** Transition to Proximate_Traffic occurs for a non-altitude reporting (or lost) intruder when bearing and range are valid and the proximate traffic classification criteria is satisfied, but the potential threat classification criteria is not. The transition also occurs for an altitude reporting intruder when the proximate traffic classification criteria is satisfied, but neither the potential threat nor the threat classification criteria are satisfied.

3.64 Macro: Proximate_Traffic_Condition

Definition:

AND

Other_Tracked_Range _{f-554} < 6.0 nmi _(PROXR)
Other_Alt_Reportings-165
Own_Tracked_Alt _{f-541} ≥ 15,500 ft _(ABOVNMC)
Current_Vertical_Separation _{f-368} < 1,200 ft _(PROXA)

OR	
T	T
F	T
F	.
.	T

Notes:

- Description:** To be considered as proximate traffic, an intruder must be within a range of 6.0 nmi beyond which visual acquisition is not likely. Additionally, if the intruder is altitude reporting, its relative altitude must be within 1,200 ft. If the intruder is not altitude reporting, then it is considered proximate traffic only if own altitude is below 15,500 ft where own would be in or near the airspace where altitude reporting is not required.
- Pseudocode Reference:** *Proximity_test*.

- Perceived gaps in the pseudocode resulted in an additional representation of the TCAS logic**
- Included supplementary explanatory text via notes**



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



How Julia Solves Our Problems

Improvement Objectives

Vendor Implementation Needs

- Readable
- Unambiguous
- Adaptive to requirements

Test Case Generation

- Generate test case inputs based on desired criteria
- No imprecision or ambiguity

Efficient Development of Software

- Validate system using simulations
- Enable interactive I/O examination of individual parts of the code
- Display simulated results

Justification for Julia

Spec. Advantages

- Natural, convenient syntax
- Technical language
- Highly documented
- Free and open source

Validation Advantages

- Executable
- Platform independent results
- High performance
- Interact w/ code via REPL
- Interpreted
- Graphical plotting libraries



Algorithm Design Description (ADD)

- Includes algorithm representations
 - High-level English explanatory text
 - Low-level Julia algorithms
- Supports vendors in creating real-time implementation of CAS logic
 - Used to implement prototype software that is integrated with avionic hardware
- Provides early verification and feedback
- Includes supporting information
 - Data flow diagrams
 - Parameter file description
 - Notation and algorithmic conventions

**Algorithm Design Description
of the
Airborne Collision Avoidance System X**



ACAS X ADD
Run 13
Version 13
Revision 1
January 8, 2015

Traffic Alert & Collision Avoidance System (TCAS)
Program Office (PO)

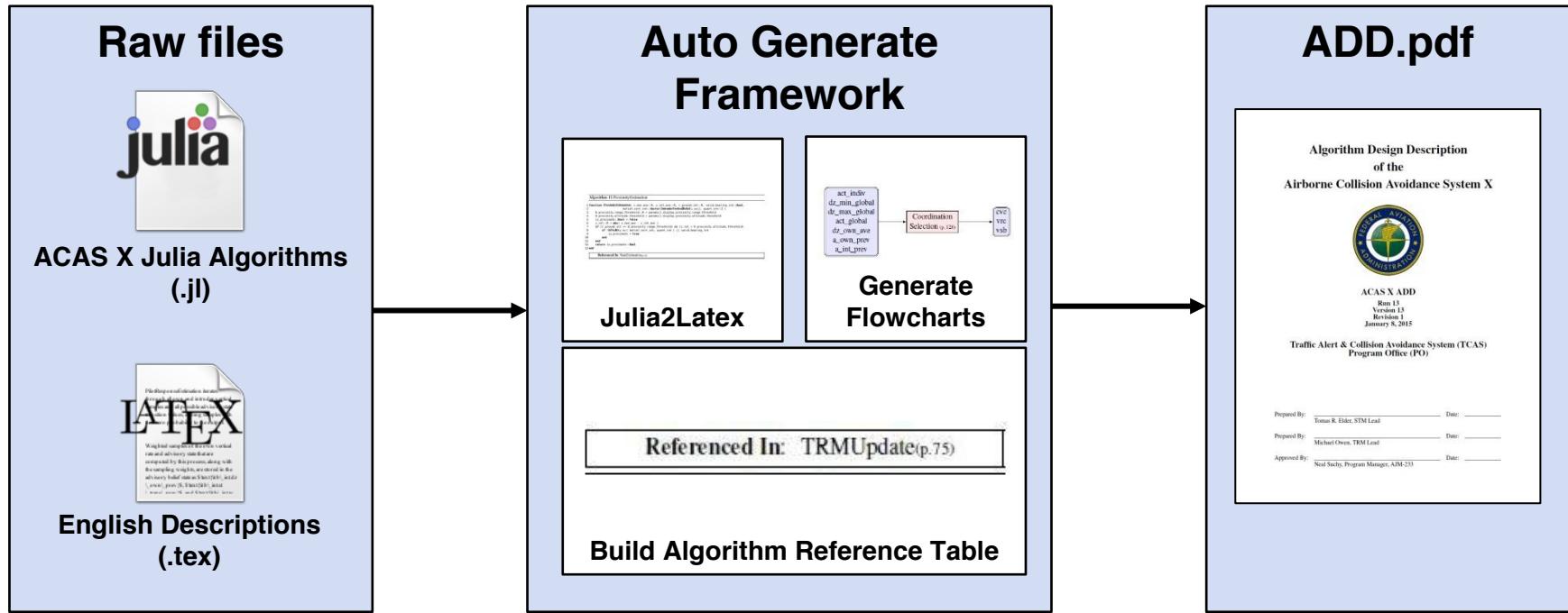
Prepared By: Tomas R. Elder, STM Lead Date: _____

Prepared By: Michael Owen, TRM Lead Date: _____

Approved By: Neal Suchy, Program Manager, AJM-233 Date: _____



ADD Auto Generate Process



- **Auto Generate Framework written purely in Julia**
 - Uses runnable and validated ACAS X Julia algorithms in place of pseudocode
 - Automatically adds hyperlinks to algorithms throughout the document
 - Generates flowcharts of algorithm inputs and outputs



English Description / Julia Algorithm

3.3 Proximity Estimation

The PROXIMITYESTIMATION (Algorithm 11) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line 8). PROXIMITYESTIMATION checks for Non-Altitude Reporting Surveillance (NARS) using IsNARS. It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in TRMUPDATEPREP line 13. Eventually, the STM should filter such tracks out before input to the TRM.

The PROXIMITYESTIMATION functionality will be moved into the STM in a future release of the ACAS X ADD.

Algorithm 11 ProximityEstimation

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,  
2                                belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::Z )  
3     D_proximity_range_threshold::R = params().display.proximity_range_threshold  
4     H_proximity_altitude_threshold = params().display.proximity_altitude_threshold  
5     is_proximate::Bool = false  
6     z_rel::R = abs( z_own_ave - z_int_ave )  
7     if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)  
8         if !IsNARS(p. 95)( belief_vert_int, quant_int ) || valid_bearing_int  
9             is_proximate = true  
10        end  
11    end  
12    return is_proximate::Bool  
13 end
```

Referenced In: StateEstimation(p. 16)



English Description / Julia Algorithm

3.3 Proximity Estimation

The PROXIMITYESTIMATION (Algorithm 11) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line 8). PROXIMITYESTIMATION checks for Non-Altitude Reporting Surveillance (NARS) using IsNARS. It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in TRMUPDATEPREP line 13. Eventually, the STM should filter such tracks out before input to the TRM.

The PROXIMITYESTIMATION functionality will be moved into the STM in a future release of the ACAS X ADD.

Algorithm 11 ProximityEstimation

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
2                               belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::Z )
3     D_proximity_range_threshold::R = params().display.proximity_range_threshold
4     H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
5     is_proximate::Bool = false
6     z_rel::R = abs( z_own_ave - z_int_ave )
7     if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
8         if !IsNARS(p. 95)( belief_vert_int, quant_int ) || valid_bearing_int
9             is_proximate = true
10        end
11    end
12    return is_proximate::Bool
13 end
```

Referenced In: StateEstimation(p.16)



English Description / Julia Algorithm

3.3 Proximity Estimation

The [PROXIMITYESTIMATION](#) ([Algorithm 11](#)) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line 8). [PROXIMITYESTIMATION](#) checks for Non-Altitude Reporting Surveillance (NARS) using [IsNARS](#). It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in [TRMUPDATEPREP](#) line 13. Eventually, the STM should filter such tracks out before input to the TRM.

The [PROXIMITYESTIMATION](#) functionality will be moved into the STM in a future release of the ACAS X ADD.

Algorithm 11 ProximityEstimation

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,  
2           belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::Z )  
3   D_proximity_range_threshold::R = params().display.proximity_range_threshold  
4   H_proximity_altitude_threshold = params().display.proximity_altitude_threshold  
5   is_proximate::Bool = false  
6   z_rel::R = abs( z_own_ave - z_int_ave )  
7   if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)  
8     if !IsNARS(p. 95)( belief_vert_int, quant_int ) || valid_bearing_int  
9       is_proximate = true  
10    end  
11  end  
12  return is_proximate::Bool  
13 end
```

Referenced In: [StateEstimation](#)([p. 16](#))



English Description / Julia Algorithm

3.3 Proximity Estimation

The [PROXIMITYESTIMATION](#) ([Algorithm 11](#)) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line 8). [PROXIMITYESTIMATION](#) checks for Non-Altitude Reporting Surveillance (NARS) using [IsNARS](#). It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in [TRMUPDATEPREP](#) line 13. Eventually, the STM should filter such tracks out before input to the TRM.

The [PROXIMITYESTIMATION](#) functionality will be moved into the STM in a future release of the ACAS X ADD.

Algorithm 11 ProximityEstimation

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,  
2           belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::z )  
3   D_proximity_range_threshold::R = params().display_proximity_range_threshold  
4   H_proximity_altitude_threshold = params().display_proximity_altitude_threshold  
5   is_proximate::Bool = false  
6   z_rel::R = abs( z_own_ave - z_int_ave )  
7   if (r_ground_int <= D_proximity_range_threshold) &  
8     if !IsNARS(p. 95)( belief_vert_int )  
9       is_proximate = true  
10    end  
11  end  
12  return is_proximate  
13 end
```

Referenced In: [StateEstimation](#)(p. 16)

Referenced In: [StateEstimation](#)(p. 16)



In-line Algorithm Descriptions as Comments

- **Block comments of LaTeX formatted code (denoted by `#=*` `*=#`)**
 - Each block comment will be placed before the algorithm in the ADD (therefore removing them from inside the algorithm)

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
2                               belief_vert_int::Vector{IntruderVerticalBelief}, quant_int::Z )
3 # These will be added to the paramsfile for Run13
4 D_proximity_range_threshold::R = params().display.proximity_range_threshold
5 H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
6 is_proximate::Bool = false
7 z_rel::R = abs( z_own_ave - z_int_ave )
8 if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
9     #=* The {\sc \nameref{alg:ProximityEstimation}} (\cref{alg:ProximityEstimation}) algorithm determines whether the intruder is
10    within the range and altitude thresholds for issuing a proximity advisory. Eventually, the STM should filter such tracks out before
11    input to the TRM. *=#
12    if !IsNARS( belief_vert_int, quant_int ) || valid_bearing_int # \label{PE_surv_quality_check}
13        #=*
14        The surveillance must be of sufficient quality to determine proximity (dictated by line~\ref{PE_surv_quality_check}).
15        {\sc \nameref{alg:ProximityEstimation}} checks for Non-Altitude Reporting Surveillance (NARS) using {\sc \nameref{alg:IsNARS}}.
16        It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid
17        range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in {\sc
18        \nameref{alg:TRMUpdatePrep}} line~\ref{TRMUP_z_own_thresh}.
19        *=#
20        is_proximate = true
21    end
22 end
23 #=*
24 The {\sc \nameref{alg:ProximityEstimation}} functionality will be moved into the STM in a future release of the ACAS X ADD.
25 *=#
```



In-line Algorithm Descriptions as Comments

The [PROXIMITYESTIMATION](#) ([Algorithm 11](#)) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line [8](#)). [PROXIMITYESTIMATION](#) checks for Non-Altitude Reporting Surveillance (NARS) using [IsNARS](#). It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in [TRMUPDATEPREP](#) line [13](#). Eventually, the STM should filter such tracks out before input to the TRM.

The [PROXIMITYESTIMATION](#) functionality will be moved into the STM in a future release of the ACAS X ADD.

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
2                               belief_vert_int::Vector{IntruderVerticalBelief}, quant_int::Z )
3 # These will be added to the paramsfile for Run13
4 D_proximity_range_threshold::R = params().display.proximity_range_threshold
5 H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
6 is_proximate::Bool = false
7 z_rel::R = abs( z_own_ave - z_int_ave )
8 if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
9     #=* The {\sc \nameref{alg:ProximityEstimation}} (\cref{alg:ProximityEstimation}) algorithm determines whether the intruder is
10    #=* within the range and altitude thresholds for issuing a proximity advisory. Eventually, the STM should filter such tracks out before
11    #=* input to the TRM. *=
12    if !IsNARS( belief_vert_int, quant_int ) || valid_bearing_int # \label{PE_surv_quality_check}
13        #=*
14        #=* The surveillance must be of sufficient quality to determine proximity (dictated by line~\ref{PE_surv_quality_check}).
15        #=* {\sc \nameref{alg:ProximityEstimation}} checks for Non-Altitude Reporting Surveillance (NARS) using {\sc \nameref{alg:IsNARS}}.
16        #=* It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid
17        #=* range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in {\sc
18        #=* \nameref{alg:TRMUpdatePrep}} line~\ref{TRMUP_z_own_thresh}.
19        #=#
20        is_proximate = true
21    end
22 end
23 #=*
24 #=* The {\sc \nameref{alg:ProximityEstimation}} functionality will be moved into the STM in a future release of the ACAS X ADD.
#*=#
```



In-line Algorithm Descriptions as Comments

The PROXIMITYESTIMATION (Algorithm 11) algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. The surveillance must be of sufficient quality to determine proximity (dictated by line 8). PROXIMITYESTIMATION checks for Non-Altitude Reporting Surveillance (NARS) using ISNARS. It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in TRMUPDATEPREP line 13. Eventually, the STM should filter such tracks out before input to the TRM.

The PROXIMITYESTIMATION functionality will be moved into the STM in a future release of the ACAS X ADD.

```
1 function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
2                               belief_vert_int::Vector{IntruderVerticalBelief}, quant_int::Z )
3 # These will be added to the paramsfile for Run13
4 D_proximity_range_threshold::R = params().display.proximity_range_threshold
5 H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
6 is_proximate::Bool = false
7 z_rel::R = abs( z_own_ave - z_int_ave )
8 if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
9     #*= The {\sc \nameref{alg:ProximityEstimation}} (\cref{alg:ProximityEstimation}) algorithm determines whether the intruder is
10    within the range and altitude thresholds for issuing a proximity advisory. Eventually, the STM should filter such tracks out before
11    input to the TRM. *=
12    if !IsNARS( belief_vert_int, quant_int ) || valid_bearing_int # \label{PE_surv_quality_check}
13        #*
14        The surveillance must be of sufficient quality to determine proximity (dictated by line~\ref{PE_surv_quality_check}).
15        {\sc \nameref{alg:ProximityEstimation}} checks for Non-Altitude Reporting Surveillance (NARS) using {\sc \nameref{alg:ISNARS}}.
16        It does not test for a valid range value because the Surveillance and Tracking Module (STM) filters out tracks with no valid
17        range. It also does not test if own altitude is above the non-Mode C threshold, because such tracks are filtered out in {\sc
18        \nameref{alg:TRMUpdatePrep}} line~\ref{TRMUP_z_own_thresh}.
19        *=#
20        is_proximate = true
21    end
22 end
23 return is_proximate::Bool
24 *=#
25 The {\sc \nameref{alg:ProximityEstimation}} functionality will be moved into the STM in a future release of the ACAS X ADD.
26 *=#
```



Algorithm and Type Descriptions

Algorithm 3 ConvertHorizontal

```
1 function ConvertHorizontal( b_horiz_int::Vector{IntruderHorizontalBelief(p. 103)} ) ← Typed Inputs
2     o::Z = length( b_horiz_int )
3     r_ground::Vector{R} = zeros( R, o )
4     s_ground::Vector{R} = zeros( R, o )
5     phi_rel::Vector{R} = zeros( R, o )
6     w_int_horiz::Vector{R} = [ b.weight for b in b_horiz_int ]
7     for i = 1:o
8         b::IntruderHorizontalBelief(p. 103) = b_horiz_int[i]
9         r_ground[i] = sqrt(b.x_rel^2 + b.y_rel^2 )
10        s_ground[i] = sqrt( b.dx_rel^2 + b.dy_rel^2 )
11        phi_rel[i] = abs( WrapToPi(p. 124)( atan2( b.dy_rel, b.dx_rel ) - atan2( b.y_rel, b.x_rel ) ) )
12    end
13    return (r_ground::Vec{o}, s_ground::Vec{o}, phi_rel::Vec{o}, w_int_horiz::Vec{o}) ← Typed Outputs
14 end
```

Referenced In: StateEstimation(p.16)



Where it's used...
(hyperlink and page ref.)

Type 6 | IntruderHorizontalBelief

```
1 type IntruderHorizontalBelief
2     x_rel::R # E/W component of position relative to own ship (ft)
3     y_rel::R # N/S component of position relative to own ship (ft)
4     dx_rel::R # E/W component of velocity relative to own ship (ft/s)
5     dy_rel::R # N/S component of velocity relative to own ship (ft/s)
6     weight::R # weight of this sample [0-1]
7     IntruderHorizontalBelief( x_rel::R, y_rel::R, dx_rel::R, dy_rel::R, w::R ) = new( x_rel, y_rel,
        dx_rel, dy_rel, w )
8     IntruderHorizontalBelief() = new( 0.0, 0.0, 0.0, 0.0, 0.0 )
9 end
```

Referenced In: ConvertHorizontal(p.17)



Algorithm and Type Descriptions

Algorithm 3 ConvertHorizontal

```
1 function ConvertHorizontal( b_horiz_int::Vector{IntruderHorizontalBelief(p. 103)} )
2     o::Z = length( b_horiz_int )
3     r_ground::Vector{R} = zeros( R, o )
4     s_ground::Vector{R} = zeros( R, o )
5     phi_rel::Vector{R} = zeros( R, o )
6     w_int_horiz::Vector{R} = [ b.weight for b in b_horiz_int ]
7     for i = 1:o
8         b::IntruderHorizontalBelief(p. 103) = b_horiz_int[i]
9         r_ground[i] = sqrt(b.x_rel^2 + b.y_rel^2 )
10        s_ground[i] = sqrt( b.dx_rel^2 + b.dy_rel^2 )
11        phi_rel[i] = abs( WrapToPi(p. 124)( atan2( b.dy_rel, b.dx_rel ) - atan2( b.y_rel, b.x_rel ) ) )
12    end
13    return (r_ground::Vec{o}, s_ground::Vec{o}, phi_rel::Vec{o}, w_int_horiz::Vec{o})
14 end
```

Referenced In: StateEstimation(p. 16)

Type 6 | IntruderHorizontalBelief

Where it's used...
(hyperlink and page ref.)

```
1 type IntruderHorizontalBelief
2     x_rel::R # E/W component of position relative to own ship (ft)
3     y_rel::R # N/S component of position relative to own ship (ft)
4     dx_rel::R # E/W component of velocity relative to own ship (ft/s)
5     dy_rel::R # N/S component of velocity relative to own ship (ft/s)
6     weight::R # weight of this sample [0-1]
7     IntruderHorizontalBelief( x_rel::R, y_rel::R, dx_rel::R, dy_rel::R, w::R ) = new( x_rel, y_rel,
8                                     dx_rel, dy_rel, w )
9     IntruderHorizontalBelief() = new( 0.0, 0.0, 0.0, 0.0, 0.0 )
9 end
```

Typed Data Fields

Referenced In: ConvertHorizontal(p. 17)



Generated Data Flow Diagrams

Algorithm 2 StateEstimation

```

1 function StateEstimation( mode_int::z, a_own_prev::Advisory(p. 68), b_int_prev::AdvisoryBeliefState(p. 68),
2     z_own::Vec{n}, dz_own::Vec{n}, w_own_vert::Vec{n},
3     z_int::Vec{m}, dz_int::Vec{m}, w_int_vert::Vec{m},
4     x_rel::Vec{r}, dx_rel::Vec{r}, y_rel::Vec{r}, dy_rel::Vec{r},
5     w_int_horiz::Vec{r})
6     (r_ground, s_ground, phi_rel, w_int_horiz) = ConvertHorizontal(p. 12)(x_rel, y_rel, dx_rel, dy_rel,
7         w_int_horiz)
8     (tau, w_tau) = TauEstimation(p. 15)(mode_int, r_ground, s_ground, phi_rel, w_int_horiz, z_own, dz_own,
9         w_own_vert, z_int, dz_int, w_int_vert)
10    (z_rel_samp, dz_own_samp, dz_int_samp, s_RA_samp, w_vert_samp) = PilotResponseEstimation(p. 16)(mode_int,
11        z_own, dz_own, w_own_vert, z_int, dz_int, w_int_vert, b_int_prev, a_own_prev)
12    (tau_samp, z_rel_samp, dz_own_samp, dz_int_samp, s_RA_samp, w_samp) = CombineSamples(p. 19)(tau, w_tau,
13        z_rel_samp, dz_own_samp, dz_int_samp, s_RA_samp, w_vert_samp)
14    return (z_rel_samp::Vec{q}, dz_own_samp::Vec{q}, dz_int_samp::Vec{q},
15        s_RA_samp::Arr{q}, tau_samp::Vec{q}, w_samp::Vec{q},
16        tau::Vec{l}, w_tau::Vec{l})
17 end

```

Referenced In: TRMUpdate(p. 9)

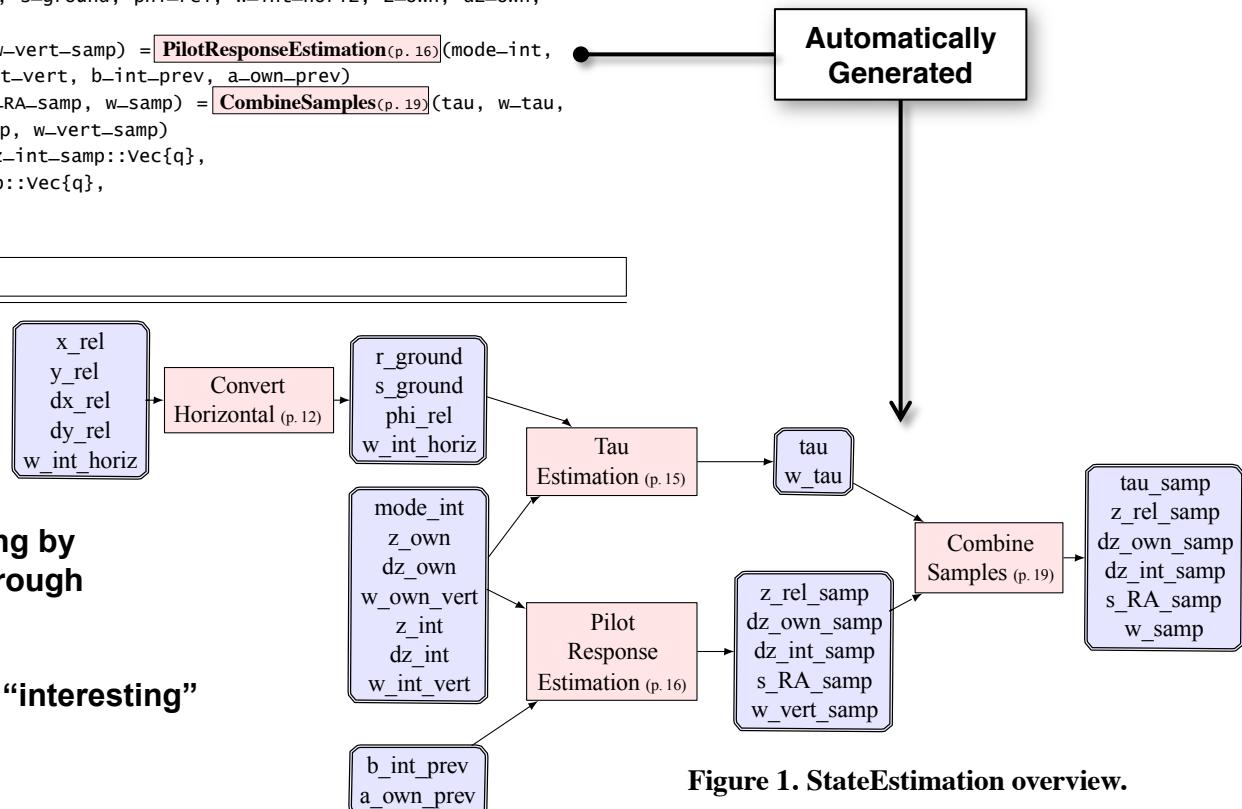


Figure 1. StateEstimation overview.



Overview

- **Background on CAS**
- **CAS Specification**
- **Simulation Frameworks**
 - **Benchmarks**
- **Verification and Validation Tools in Julia**
- **Summary**



Simulation Framework

CSIM



Simulation Framework

Common libacax interface



Development Logic



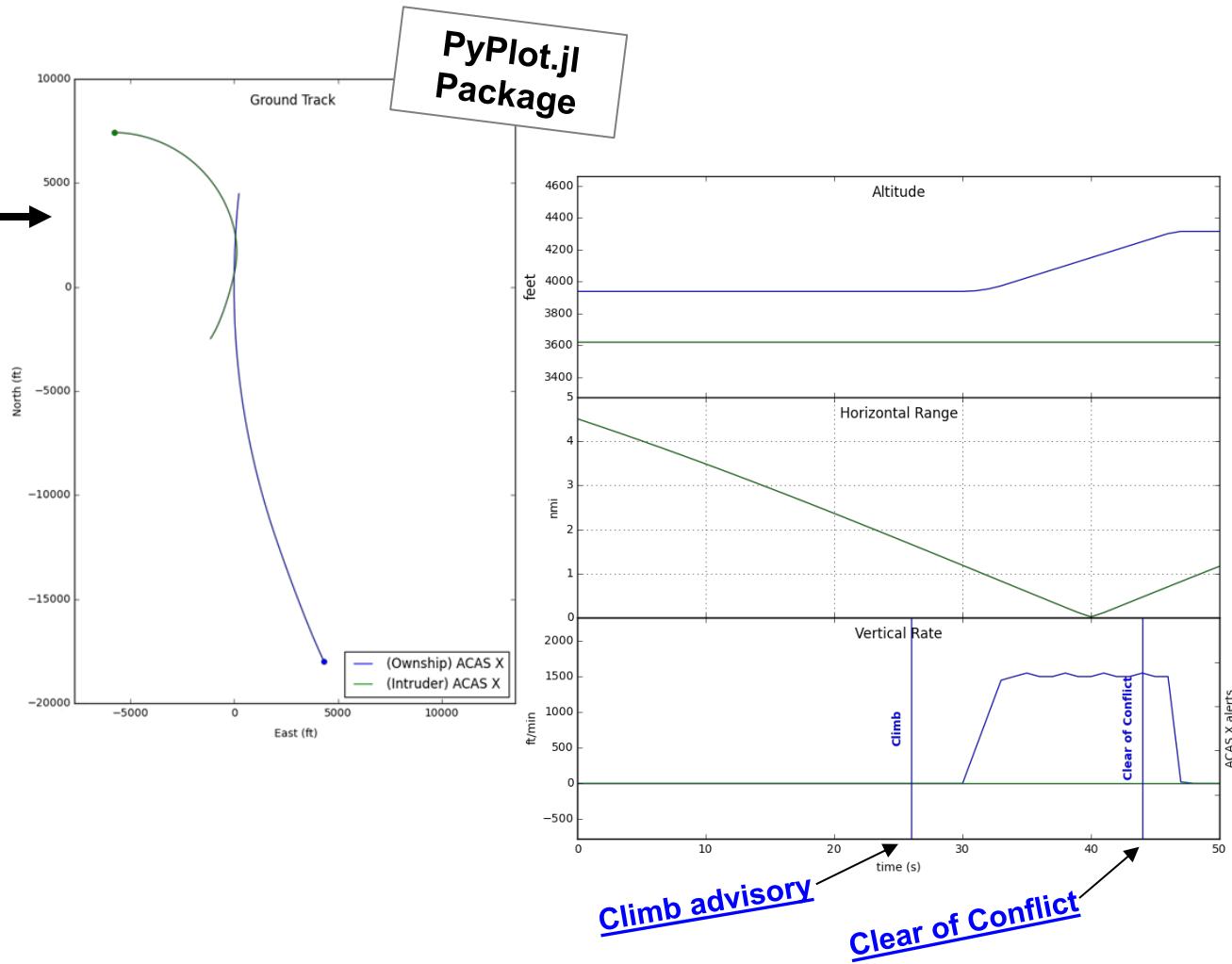
or

ADD Logic

Parameter File



Lookup Tables





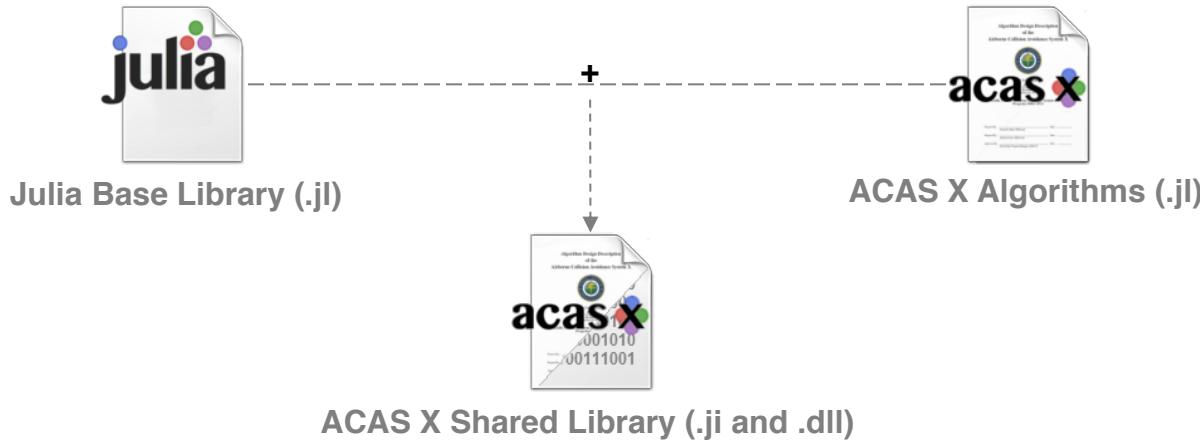
Precompiled Julia Binary

sys.ji

- Julia's system library
- Includes the entire Base module
- Compiled during Julia build process

acasx.ji

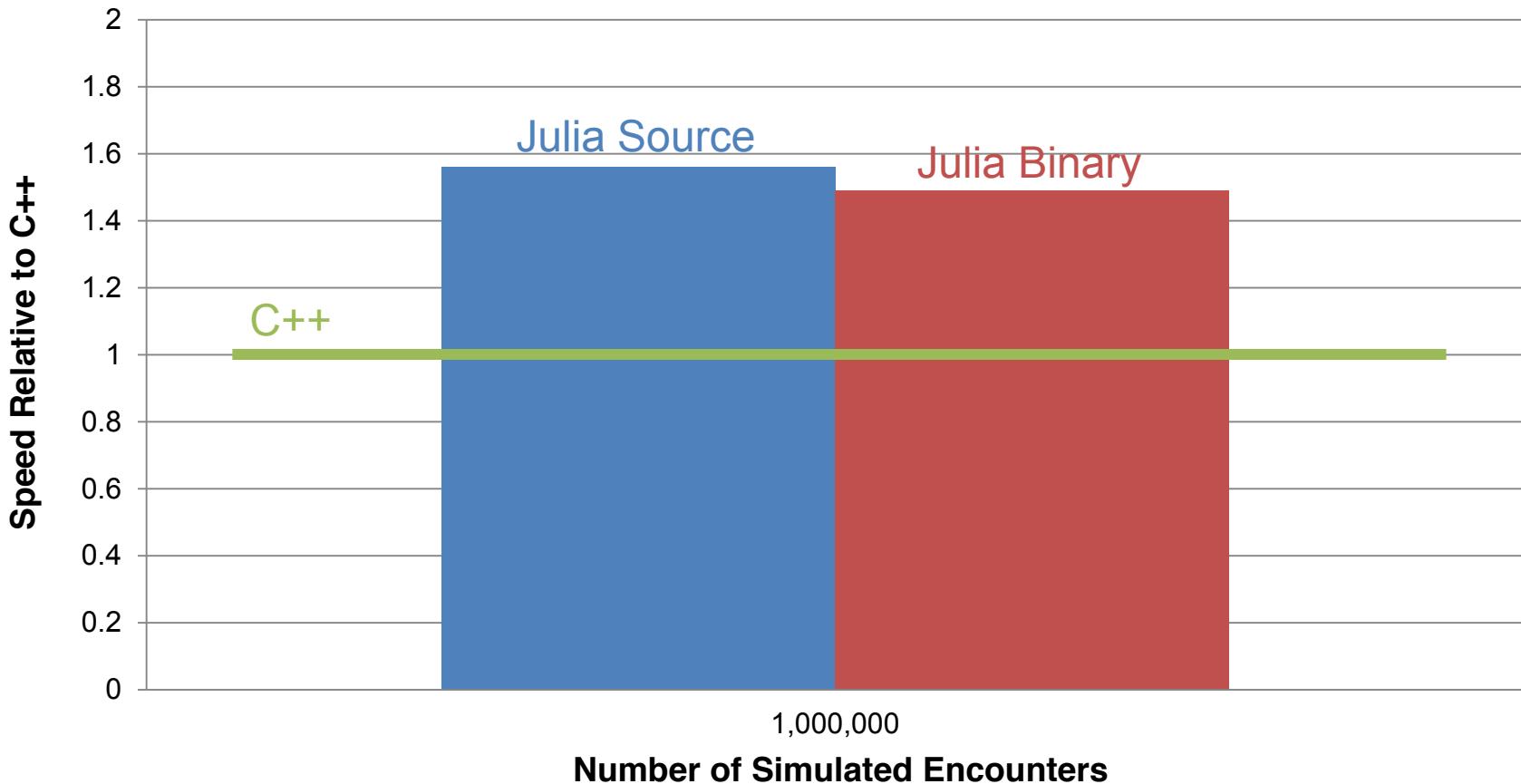
- Julia's system library + ACAS X algorithms
- Algorithms are preloaded when Julia starts
- Speed benefit compared to raw source



- When distributing the ADD, we provide the precompiled Julia binaries to ensure the source is preserved
 - The performance increase is also an advantage



Benchmarks*



Julia's performance enables us to validate our entire system



Overview

- Background on CAS
- CAS Specification
- Simulation Frameworks
- **Verification and Validation Tools in Julia**
- Summary



Variety of Julia Applications

MATLAB Graphical User Interface
(Select input files, call C++ entry point, plot encounter geometry, facilitate playback, display logic understanding tools)

C++ Language Bridge
(Initialize Julia, pass file parameters to Julia execute function)

Julia ASIM Wrapper
(Parse encounter input file, call algorithms, generate STM and TRM output files)

Julia ACAS X Algorithms



Point of contact
Rachel Szczesiul (rachel.szczesiul@jhuapl.edu)

Example:

ACAS X Simulation Interactive Module (ASIM)



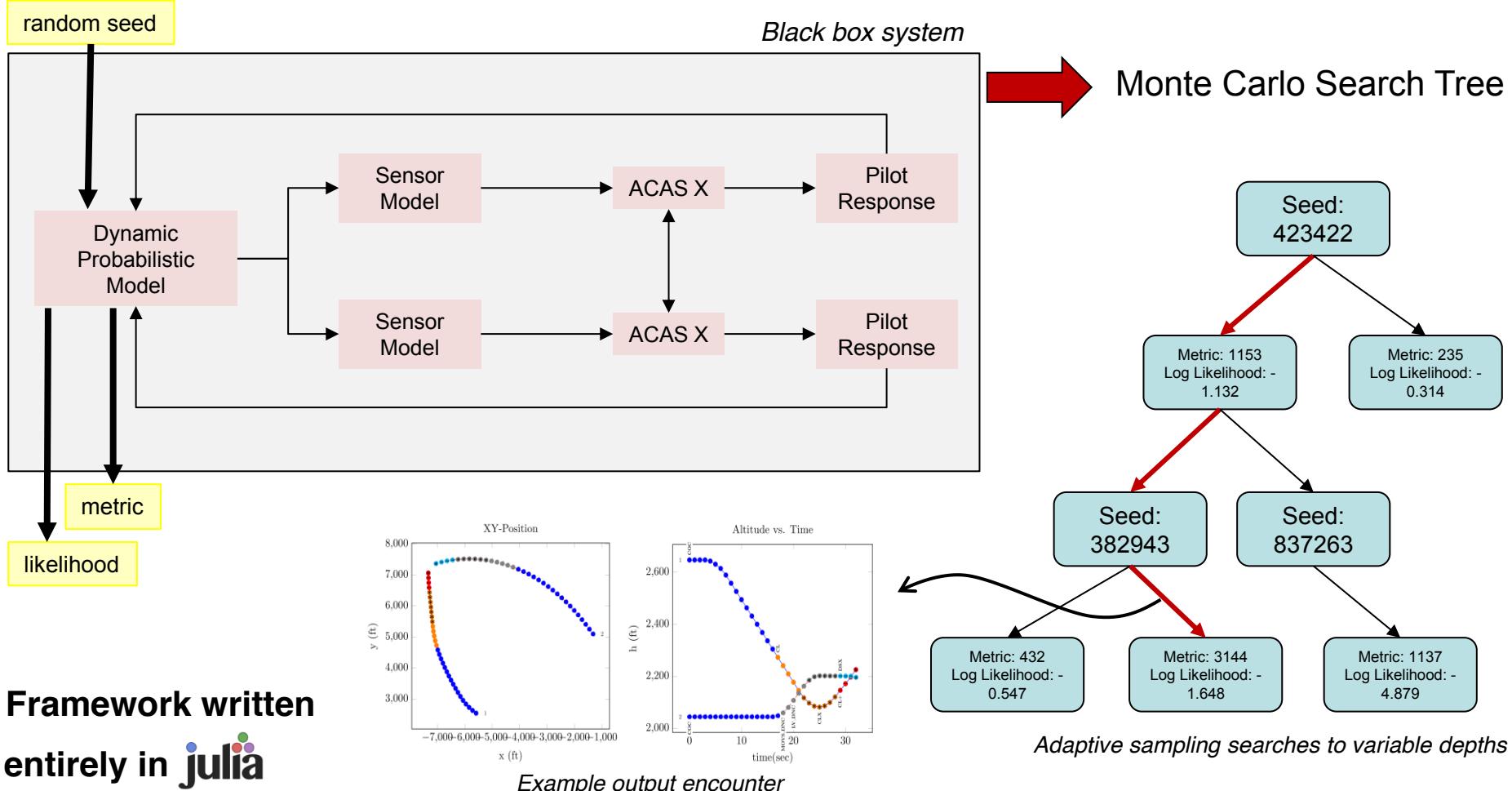


Example: Adaptive Stress Testing

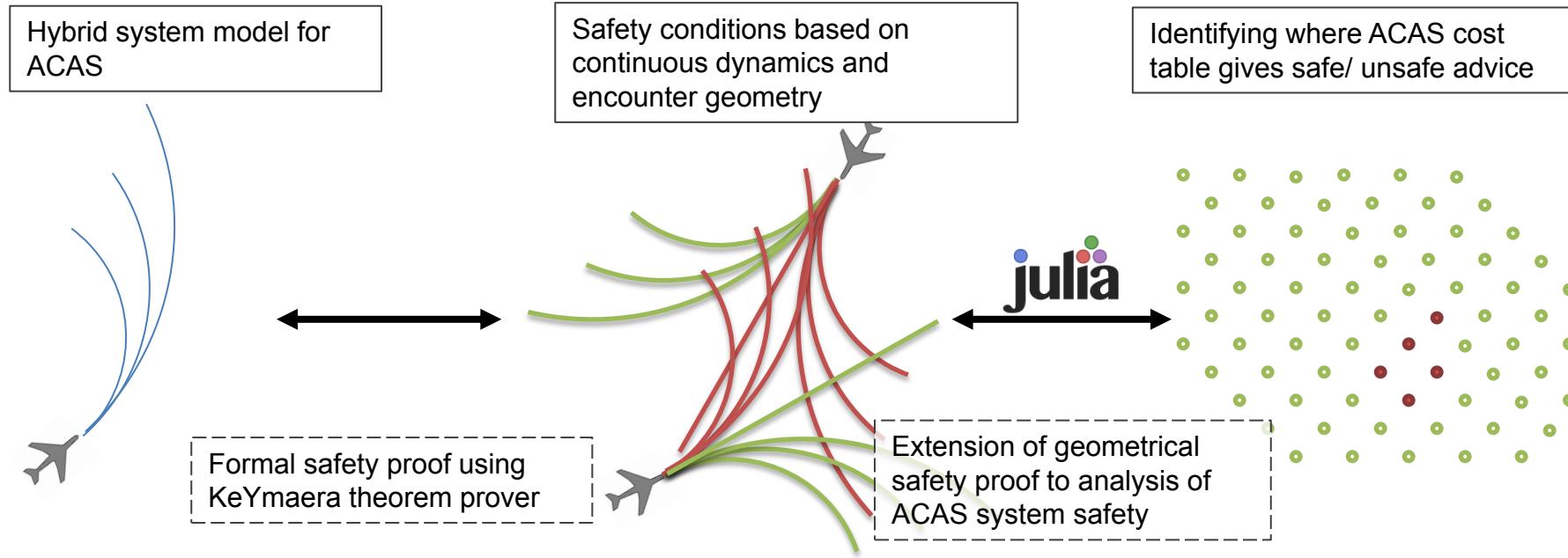
R. Lee (CMU), M. J. Kochenderfer (Stanford), O. J. Mengshoel (CMU) and G. P. Brat (NASA)

Problem: What are the most likely failures of the system?

Approach: Adaptively sample from a black box simulator to maximize likelihood of failure



Theorem Proving for Safety Analysis



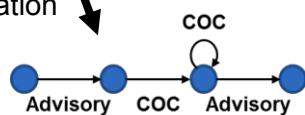
- **Julia computes exhaustive comparison over $6.5e11$ discrete decision points**
 - Would be computationally expensive for MATLAB
- **Utilized lexical closures and macros**
 - “Precompiled” descriptions of pilot behavior into customized safety functions at runtime for efficiency



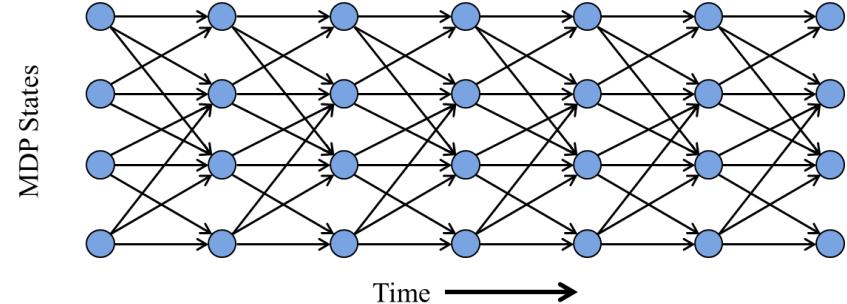
Example: Probabilistic Model Checking

Advisory (COC+) Advisory
Regular expression query of
MDP state-action sequence

Translation



Finite State Machine Query



LARGE Markov Decision Process System

julia

Probabilistic Model Checking

- Searches all 10^{70} paths through MDP that match query over time
- Maintains parse state during dynamic programming with FSM

Most helpful Julia features...

- SparseMatrixCSC for the 6 TB transition probability matrix
 - Allows iterating over columns directly and efficiently
- MAT package
- Debug package
- Base.Test module
- @parallel

Julia features to look into...

- Distributed arrays
- Macros to specialize code to query

Julia features desired...

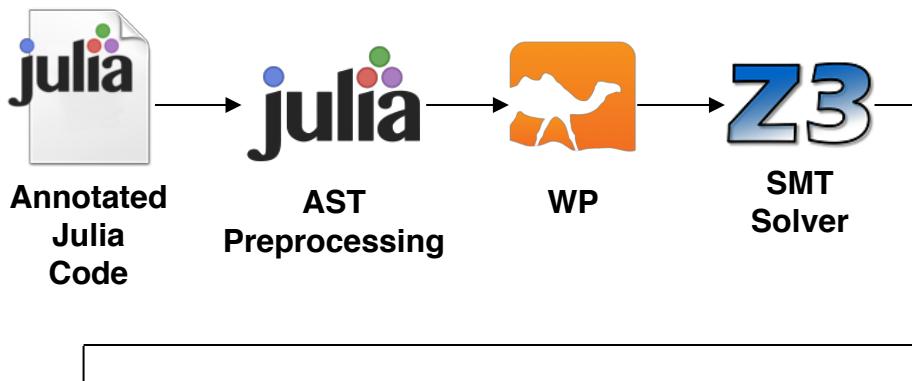
- Debugging macro-expanded code
- Debugging parallel code on workers
- Docs of functions by parameter type for us OO enthusiasts



Example: Weakest Precondition Tool

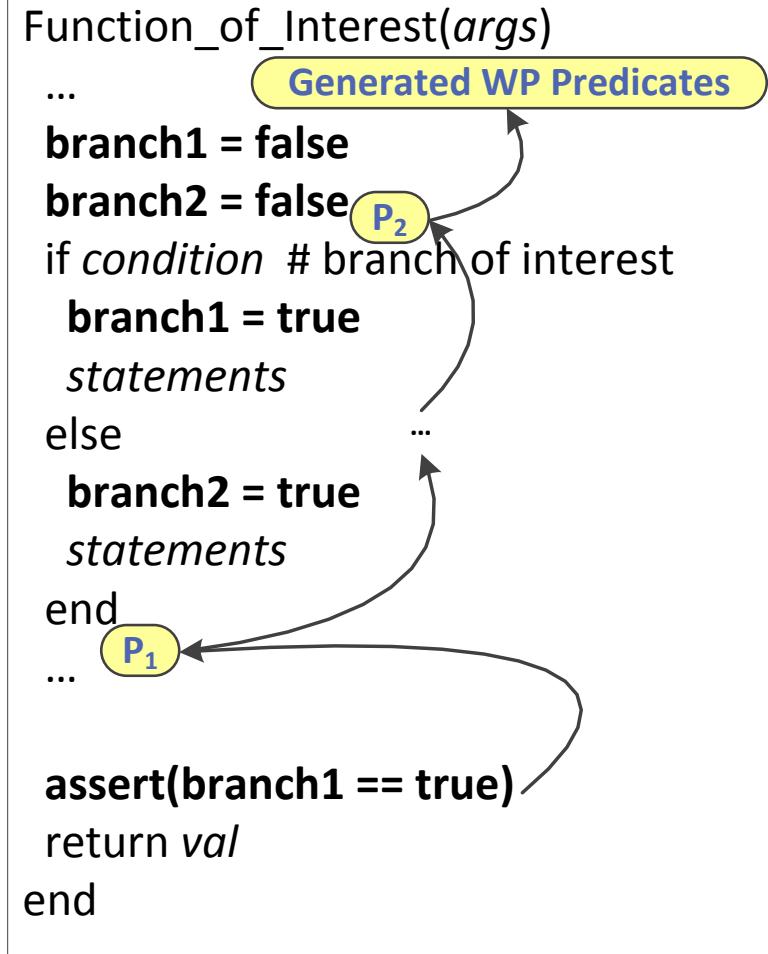
- Code analysis tool based on Dijkstra's Weakest Precondition (WP Tool)

Used for semi-automated generation of inputs to achieve branch coverage of Julia code



Output:
concrete inputs, or “unsatisfiable”

- Tool is generic – can be used for properties other than branch coverage





Overview

- **Background on CAS**
- **CAS Specification**
- **Simulation Frameworks**
- **Verification and Validation Tools in Julia**
- **Summary**



Summary

- **Julia resolves many outstanding issues with legacy specification methods**
 - Expected to reduce tech transfer cost and errors
- **Julia enables validation of entire system**
 - Allows the specification to be executed and tested directly
- **Precompiled binary file protects the source when distributed as a shared library**
- **Julia is useful for a variety of tasks throughout the program**
 - Core ACAS X system
 - Document generation framework
 - V&V tools



Future Work

- **Incorporate Julia implementation into tuning framework**
 - Final ACAS X logic will be tuned with the Julia ADD algorithms
- **Resolve shortcomings of a quickly evolving language**
 - Experimenting with a precompiled binary
 - MATLAB → C++ → Julia
- **Continue to push Julia as the standard for avionics specifications**
- **A flight test of ACAS X is scheduled for the fall of 2015**



Acknowledgements

- I would like to thank **Mykel Kochenderfer**, **Tomas Elder**, and **Josh Silbermann** for their contributions to this brief.
- Also, a big thanks to the entire ACAS X team:
 - Wes Olson
 - Michael Owen
 - Cindy McLain
 - Robert Klaus
 - Justin Mackay
 - Matt Feldman
 - Ted Londner
 - Adam Panken
 - Larry Capuder
 - Ian Jessen
 - Keertana Settaluri
 - Jim Jackson
 - Ann Drumm
 - Bill Harman
 - Loren Wood
 - Rod Cole
 - Jeff Brush
 - Ryan Gardner
 - Aurora Schmidt
 - Yanni Kouskoulas
 - Anshu Saksena
 - Mark Thober
 - Rachel Szczesiu
 - Richie Lee
 - Neal Suchy

...and many others!
- Julia Computing contributions
 - Jeff Bezanson
 - Jameson Nash

...and the entire Julia community!





Backup



Surveillance and Tracking Module (STM)

10 Asynchronous Entry Points

Algorithm

Algorithm 1, RECEIVEDFO
Algorithm 29, RECEIVESTATEVECTORPOSITIONREPORT
Algorithm 31, RECEIVESTATEVECTORVELOCITYREPORT
Algorithm 33, RECEIVEMODESTATUSREPORT
Algorithm 54, RECEIVEDISCRETES
Algorithm 62, RECEIVEBAROALTOSERVATION
Algorithm 65, RECEIVERADALTOSERVATION
Algorithm 66, RECEIVEHEADINGOSERVATION
Algorithm 72, RECEIVEWGS84OSERVATION
Algorithm 73, RECEIVEUF16UDS30
Algorithm 75, GENERATESTMREPORT

The STM utilizes 3 top level data structures

```
type Target
    modes_tracks::Vector{ModeSTrackFile}
    modec_tracks::Vector{ModeCTrackFile}
    adsb_tracks::Vector{ADSBTrackFile}
    coord_data::ReceivedCoordinationData
    av_history::ActiveValidationHistory
    av_state::Z
    designation::UInt32
end
```

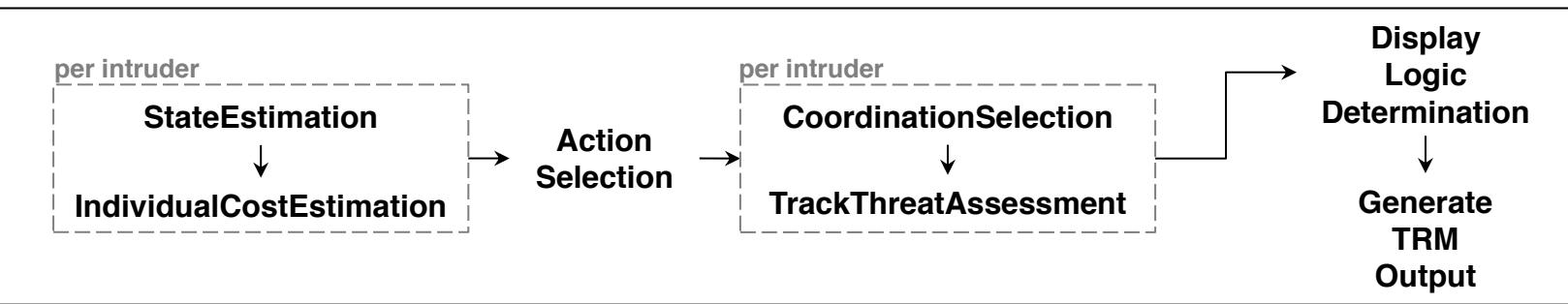
```
type OwnShipData
    modes::UInt32
    radalt::R
    mu_heading::Array{R}
    Sigma_heading::Matrix{R}
    toa_heading::R
    mu_h::Array{R}
    Sigma_h::Matrix{R}
    toa_h::R
    ...
    ...
```

```
type StmReport
    trm_input::TRMInput
    transponder::TransponderData
    display::Vector{StmDisplayStruct}
    stm_global::StmGlobalStruct
end
```



Threat Resolution Module (TRM)

TRMUpdate



```
type TRMInput  
own::TRMOwnInput
```

```
type TRMOwnInput  
h::R  
psi::R  
mode_s::Z  
equipage::Z  
belief_vert::Vector{OwnVerticalBelief}  
end
```

```
intruder::Vector{TRMIntruderInput}
```

```
type TRMIntruderInput  
id::Z  
mode_s::Z  
equipage::Z  
belief_vert::Vector{IntruderVerticalBelief}  
belief_horiz::Vector{IntruderHorizontalBelief}  
end
```

```
end
```

```
type TRMState  
st_own::TRMOwnState
```

```
type TRMOwnState  
a_prev::Advisory  
dz_ave_prev::R  
action_prev::Z  
end
```

```
st_intruder::Vector{TRMIntruderState}
```

```
type TRMIntruderState  
id::Z  
a_prev::Advisory  
b_prev::AdvisoryBeliefState  
st_cost_on::OnlineCostState  
is_coordinating::Bool  
end
```

```
params::paramsfile_type  
end
```

```
type TRMReport  
display::TRMDisplayData  
coordination::Vector{TRM  
CoordinationInterrogationData}  
debug::TRMDebugData  
end
```