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

Robert Moss
JuliaCon | 6/29/2015

* This work is sponsored by the Federal Aviation Administration under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the author and are not necessarily endorsed by the United States Government.
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

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

* Traffic Alert and Collision Avoidance System
TCAS Traffic Advisory
TCAS Resolution Advisory

Climb... Climb
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
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**

- Dynamic Uncertainty
- State Uncertainty

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

**Optimized Logic Tables**
(no pseudocode)
ACAS X Modular Architecture

ACAS X Processor

Surveillance and Tracking Module (STM)

State Estimation

Threat Resolution Module (TRM)

Optimized Logic Tables

Development Focus

Needs to be transferred to industry

Sensor Measurements

Pilot Display

Climb… Climb

Needs to be transferred to industry
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
1. Writing the spec.
   – Improvements…
     • Minimize time taken away from development
     • Easily test and validation document

2. Vendors implementing the spec.
   – Improvements…
     • Minimize implementation time
     • Remove confusion of text and algorithms
     • Provide all necessary information
**TCAS Algorithm Specification: Pseudocode**

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

**Variable Based Pseudocode**

```plaintext
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;
  ELSEIF (ITF.R GE P.PROXR)
    THEN CLEAR PRXHITA;
  ELSEIF (ITF.MODC EQ SFALSE)
    THEN IF (G.ZOWN GE P.ABOVNMC)
      THEN CLEAR PRXHITA;
    ELSE IF ((ITF.TACODE NE $TANMC AND ITF.TACODE NE $PA) AND
      (ITF.BEAROK EQ SFALSE OR ITF.RFLG EQ SFALSE))
      THEN CLEAR PRXHITA;
    ELSE SET PRXHITA;
  OTHERWISE IF (ITF.A LT P.PROXA)
    THEN SET PRXHITA;
    ELSE CLEAR PRXHITA;
END Proximity_test;
```

**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;
  ELSEIF (ITF.R GE P.PROXR)
    THEN CLEAR PRXHITA;
  ELSEIF (ITF.MODC EQ SFALSE)
    THEN IF (G.ZOWN GE P.ABOVNMC)
      THEN CLEAR PRXHITA;
    ELSE IF ((ITF.TACODE NE $TANMC AND ITF.TACODE NE $PA) AND
      (ITF.BEAROK EQ SFALSE OR ITF.RFLG EQ SFALSE))
      THEN CLEAR PRXHITA;
    ELSE SET PRXHITA;
  OTHERWISE IF (ITF.A LT P.PROXA)
    THEN SET PRXHITA;
    ELSE CLEAR PRXHITA;
END 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
## 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

<table>
<thead>
<tr>
<th>Improvement Objectives</th>
<th>Justification for Julia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vendor Implementation Needs</strong></td>
<td><strong>Spec. Advantages</strong></td>
</tr>
<tr>
<td>- Readable</td>
<td>- Natural, convenient syntax</td>
</tr>
<tr>
<td>- Unambiguous</td>
<td>- Technical language</td>
</tr>
<tr>
<td>- Adaptive to requirements</td>
<td>- Highly documented</td>
</tr>
<tr>
<td>- Free and open source</td>
<td>- Validation Advantages</td>
</tr>
<tr>
<td><strong>Test Case Generation</strong></td>
<td>- Executable</td>
</tr>
<tr>
<td>- Generate test case inputs based on desired criteria</td>
<td>- Platform independent results</td>
</tr>
<tr>
<td>- No imprecision or ambiguity</td>
<td>- High performance</td>
</tr>
<tr>
<td>- Interact w/ code via REPL</td>
<td>- Interpreted</td>
</tr>
<tr>
<td>- Display simulated results</td>
<td>- Graphical plotting libraries</td>
</tr>
</tbody>
</table>
# 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

<table>
<thead>
<tr>
<th>Improvement Objectives</th>
<th>Justification for Julia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vendor Implementation Needs</strong></td>
<td><strong>Spec. Advantages</strong></td>
</tr>
<tr>
<td>• Readable</td>
<td>• Natural, convenient syntax</td>
</tr>
<tr>
<td>• Unambiguous</td>
<td>• Technical language</td>
</tr>
<tr>
<td>• Adaptive to requirements</td>
<td>• Highly documented</td>
</tr>
<tr>
<td>• Free and open source</td>
<td></td>
</tr>
<tr>
<td><strong>Test Case Generation</strong></td>
<td><strong>Validation Advantages</strong></td>
</tr>
<tr>
<td>• Generate test case inputs based on desired criteria</td>
<td>• Executable</td>
</tr>
<tr>
<td>• No imprecision or ambiguity</td>
<td>• Platform independent results</td>
</tr>
<tr>
<td><strong>Efficient Development of Software</strong></td>
<td>• High performance</td>
</tr>
<tr>
<td>• Validate system using simulations</td>
<td>• Interact w/ code via REPL</td>
</tr>
<tr>
<td>• Enable interactive I/O examination of individual parts of the code</td>
<td>• Interpreted</td>
</tr>
<tr>
<td>• Display simulated results</td>
<td>• Graphical plotting libraries</td>
</tr>
</tbody>
</table>
# 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

<table>
<thead>
<tr>
<th>Improvement Objectives</th>
<th>Justification for Julia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vendor Implementation Needs</strong></td>
<td><strong>Spec. Advantages</strong></td>
</tr>
<tr>
<td>- Readable</td>
<td>- Natural, convenient syntax</td>
</tr>
<tr>
<td>- Unambiguous</td>
<td>- Technical language</td>
</tr>
<tr>
<td>- Adaptive to requirements</td>
<td>- Highly documented</td>
</tr>
<tr>
<td></td>
<td>- Free and open source</td>
</tr>
<tr>
<td><strong>Test Case Generation</strong></td>
<td><strong>Validation Advantages</strong></td>
</tr>
<tr>
<td>- Generate test case inputs based on desired criteria</td>
<td>- Executable</td>
</tr>
<tr>
<td>- No imprecision or ambiguity</td>
<td>- Platform independent results</td>
</tr>
<tr>
<td></td>
<td>- High performance</td>
</tr>
<tr>
<td></td>
<td>- Interact w/ code via REPL</td>
</tr>
<tr>
<td></td>
<td>- Interpreted</td>
</tr>
<tr>
<td></td>
<td>- Graphical plotting libraries</td>
</tr>
<tr>
<td><strong>Efficient Development of Software</strong></td>
<td></td>
</tr>
<tr>
<td>- Validate system using simulations</td>
<td></td>
</tr>
<tr>
<td>- Enable interactive I/O examination of individual parts of the code</td>
<td></td>
</tr>
<tr>
<td>- Display simulated results</td>
<td></td>
</tr>
</tbody>
</table>
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
**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
3.3 Proximity Estimation

The **Proximity Estimation** (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). **Proximity Estimation** 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 `TRMUUPDATEPREP` line 13. Eventually, the STM should filter such tracks out before input to the TRM.

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

### Algorithm 11 ProximityEstimation

```julia
function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
    belief_vert_int::Vector{IntruderVerticalBelief}(p. 102), quant_int::Z )
    D_proximity_range_threshold::R = params().display.proximity_range_threshold
    H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
    is_proximate::Bool = false
    z_rel::R = abs( z_own_ave - z_int_ave )
    if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
        if !IsNARS(p. 95)( belief_vert_int, quant_int ) || valid_bearing_int
            is_proximate = true
        end
    end
    return is_proximate::Bool
end
```

*Referenced In: StateEstimation(p. 16)*
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 TRMUpdateRep 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

```julia
function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
    belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::Z )
    D_proximity_range_threshold::R = params().display.proximity_range_threshold
    H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
    is_proximate::Bool = false
    z_rel::R = abs( z_own_ave - z_int_ave )
    if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
        if !IsNARS(p. 95)( belief_vert_int, quant_int ) || valid_bearing_int
            is_proximate = true
        end
    end
    return is_proximate::Bool
end
```

Referenced In: StateEstimation(p. 16)
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

```
function ProximityEstimation( z-own-ave::R, z-int-ave::R, r-ground-int::R, valid-bearing-int::Bool,
                           belief-vert-int::Vector{IntruderVerticalBelief(p. 102)}, quant-int::Z )

  D-proximity-range-threshold::R = params().display.proximity-range-threshold
  H-proximity-altitude-threshold = params().display.proximity-altitude-threshold
  is-proximate::Bool = false
  z-rel::R = abs(z-own-ave - z-int-ave)
  if (r-ground-int <= D-proximity-range-threshold) && (z-rel < H-proximity-altitude-threshold)
    if !IsNARS(p. 95)( belief-vert-int, quant-int ) || valid-bearing-int
      is-proximate = true
    end
  end

return is-proximate::Bool
```

Referenced In: StateEstimation(p. 16)
3.3 Proximity Estimation

The **Proximity Estimation** (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). **Proximity Estimation** 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 **Proximity Estimation** functionality will be moved into the STM in a future release of the ACAS X ADD.

**Algorithm 11 ProximityEstimation**

```plaintext
function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
    belief_vert_int::Vector{IntruderVerticalBelief(p. 102)}, quant_int::Z )
    D_proximity_range_threshold::R = params().display.proximity_range_threshold
    H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
    is_proximate::Bool = false
    z_rel::R = abs( z_own_ave - z_int_ave )
    if (r_ground_int <= D_proximity_range_threshold)
        if !IsNARS(p. 95)( belief_vert_int )
            is_proximate = true
        end
    end
    return is_proximate
end
```

Referenced In: **StateEstimation**(p. 16)
In-line Algorithm Descriptions as Comments

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

```plaintext
function ProximityEstimation(z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
believe_vert_int::Vector{IntruderVerticalBelief}, quant_int::Z)

# These will be added to the paramsfile for Run13
D_proximity_range_threshold::R = params().display.proximity_range_threshold
H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
is_proximate::Bool = false
z_rel::R = abs(z_own_ave - z_int_ave)
if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
  # The {\sc \nameref{alg:ProximityEstimation}} (\cref{alg:ProximityEstimation}) algorithm determines whether the intruder is
  # within the range and altitude thresholds for issuing a proximity advisory. Eventually, the STM should filter such tracks out before
  # input to the TRM. *=#
  if !IsNARS(believe_vert_int, quant_int) || valid_bearing_int # \label{PE_surv_quality_check}
    # The surveillance must be of sufficient quality to determine proximity (dictated by line-\ref{PE_surv_quality_check}).
    # \{\sc \nameref{alg:ProximityEstimation}\} checks for Non-Altitude Reporting Surveillance (NARS) using \{\sc \nameref{alg: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 \{\sc
    # \nameref{alg:TRMUpdatePrep}\} line-\ref{TRMUP_z_own_thresh}.
    #=*#
    is_proximate = true
  end
end
return is_proximate::Bool
end
```

The \{\sc \nameref{alg:ProximityEstimation}\} functionality will be moved into the STM in a future release of the ACAS X ADD.

`*=#`
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.
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.

```plaintext
function ProximityEstimation( z_own_ave::R, z_int_ave::R, r_ground_int::R, valid_bearing_int::Bool,
   belief_vert_int::Vector{IntruderVerticalBelief}, quant_int::Z )
    # These will be added to the paramsfile for Run13
    D_proximity_range_threshold::R = params().display.proximity_range_threshold
    H_proximity_altitude_threshold = params().display.proximity_altitude_threshold
    is_proximate::Bool = false
    z_rel::R = abs( z_own_ave - z_int_ave )
    if (r_ground_int <= D_proximity_range_threshold) && (z_rel < H_proximity_altitude_threshold)
        # The \{\nameref{alg:ProximityEstimation}\} algorithm determines whether the intruder is within the range and altitude thresholds for issuing a proximity advisory. Eventually, the STM should filter such tracks out before input to the TRM. *=
        if !IsNARS( belief_vert_int, quant_int ) || valid_bearing_int
            # \label{PE_surv_quality_check}
            # The \{\nameref{alg:ProximityEstimation}\} checks for Non-Altitude Reporting Surveillance (NARS) using \{\nameref{alg:ProximityEstimation}\}.
            # 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 \{\nameref{alg:TRMUpdatePrep}\} line~\ref{TRMUP_z_own_thresh}.
            #=
            is_proximate = true
        end
    end
    return is_proximate::Bool
end
```

* 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 and Type Descriptions

Algorithm 3 ConvertHorizontal

```plaintext
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 = 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(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})
end
```

Referenced In: StateEstimation(p. 16)

Type 6 | IntruderHorizontalBelief

```plaintext
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)
9 end
```

Referenced In: ConvertHorizontal(p. 17)
Algorithm and Type Descriptions

Algorithm 3 ConvertHorizontal

```c
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

```c
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)

Where it’s used…
(hyperlink and page ref.)
Algorithm 2 StateEstimation

```plaintext
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, w-int-horiz)
7     (tau, w-tau) = TauEstimation(p. 15)(mode_int, r-ground, s-ground, phi-rel, w-int-horiz, z_own, dz_own, w own_vert, z_int, dz_int, w_int_vert)
8     (z-rel-samp, dzOwn-samp, dz-int-samp, s_RA-samp, w-vert-samp) = PilotResponseEstimation(p. 16)(mode_int, z_own, dz_own, w_own_vert, z_int, dz_int, w_int_vert, b_int_prev, a_own_prev)
9     (tau-samp, z-rel-samp, dzOwn-samp, dz-int-samp, s_RA-samp, w-samp) = CombineSamples(p. 19)(tau, w-tau, z_rel-samp, dzOwn-samp, dz-int-samp, s_RA-samp, w-vert-samp)
10    return (z-rel-samp::Vec(q), dzOwn-samp::Vec(q), dz-int-samp::Vec(q), s_RA-samp::Arr(q), tau-samp::Vec(q), w-samp::Vec(q), tau::Vec(1), w-tau::Vec(1))
11 end
```

Referenced In: TRMUpdate(p. 9)

- Makes use of Julia’s metaprogramming by graphing input data to output data through operations
- Walks through Julia code looking for “interesting” assignments

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 libacasx interface**

- Development Logic
- ADD Logic
- Parameter File
- Lookup Tables

**PyPlot.jl Package**

**Diagram**

1. **Ground Track**
2. **Altitude**
3. **Horizontal Range**
4. **Vertical Rate**

- (Ownship) ACAS X
- (Intruder) ACAS X

**ACAS X Julia - 39**

**RM 6/26/15**

**LINCOLN LABORATORY**

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**
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*

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>23 min.</td>
</tr>
<tr>
<td>Julia Binary</td>
<td>34 min.</td>
</tr>
<tr>
<td>Julia Source</td>
<td>36 min.</td>
</tr>
</tbody>
</table>

Julia’s performance enables us to validate our entire system

* Simulations were ran using 512 cores of the LLGrid parallel computing cluster
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

Example:
ACAS X Simulation Interactive Module (ASIM)

Point of contact
Rachel Szczesiul (rachel.szczesiul@jhuapl.edu)
**Problem:** What are the most likely failures of the system?

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

---

**Black box system**

- **Dynamic Probabilistic Model**
  - Sensor Model
  - ACAS X
  - Pilot Response

- **Monte Carlo Search Tree**
  - Seed: 423422
    - Metric: 1153
    - Log Likelihood: -1.132
  - Seed: 382943
    - Metric: 3144
    - Log Likelihood: -1.648
  - Seed: 837263
    - Metric: 1137
    - Log Likelihood: -4.879

---

**Framework written entirely in Julia**

**Example output encounter**

---

**Point of contact**

Mykel Kochenderfer (mykel@stanford.edu) or Ritchie Lee (ritchie.lee@nasa.gov)
Example: Theorem Proving for Safety Analysis

- Hybrid system model for ACAS
- Safety conditions based on continuous dynamics and encounter geometry
- Identifying where ACAS cost table gives safe/unsafe advice

- Formal safety proof using KeYmaera theorem prover
- Extension of geometrical safety proof to analysis of ACAS system safety

- **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

**Point of contact**
Yanni Kouskoulas (yanni.kouskoulas@jhuapl.edu)
Example: Probabilistic Model Checking

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

Finite State Machine Query

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

Point of contact
Anshu Saksena (anshu.saksena@jhuapl.edu)
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

---

Example:

```python
Function_of_Interest(args)
...
branch1 = false
branch2 = false
if condition  # branch of interest
    branch1 = true
    statements
else
    branch2 = true
    statements
end
...
assert(branch1 == true)
return val
end
```

Generated WP Predicates

- $P_1$
- $P_2$

Point of contact
Mark Thober (mark.thober@jhuapl.edu)
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**
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 Brash
- Ryan Gardner
- Aurora Schmidt
- Yanni Kouskoulas
- Anshu Saksena
- Mark Thober
- Rachel Szczesiul
- Richie Lee
- Neal Suchy

Julia Computing contributions

- Jeff Bezanson
- Jameson Nash

...and the entire Julia community!
Surveillance and Tracking Module (STM)

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`

**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`
Threat Resolution Module (TRM)

TRMUpdate

per intruder

StateEstimation
→
IndividualCostEstimation

Action Selection

per intruder

CoordinationSelection
→
TrackThreatAssessment

Display Logic
Determination
→
Generate TRM Output

type TRMInput

own::TRMOwnInput

intruder::Vector{TRMIntruderInput}

end

type TRMOwnInput

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

type TRMState

st_own::TRMOwnState

st_intruder::Vector{TRMIntruderState}

params::paramsfile_type
end

type TRMOwnState

a_prev::Advisory
dz_ave_prev::R
action_prev::Z
end

type TRMIntruderState

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

end

type TRMReport

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

Threat Resolution Module (TRM)