Formal Verification in Aeronautics: Current Practice and Upcoming Standard

Yannick Moy, AdaCore
ACSL Workshop, Fraunhofer FIRST
What is DO-178B/ED-12B?

RTCA

EUROCAE

Software Considerations in Airborne Systems and Equipment Certification
What is DO-178B/ED-12B?

RTCA

EUROCAE

Software Considerations in Airborne Systems and Equipment Certification

System Aspects
What is DO-178B/ED-12B?

RTCA

EUROCAE

Software Considerations in Airborne Systems and Equipment Certification

Development Processes

Verification Processes

System Aspects
What is DO-178B/ED-12B?

RTCA  EUROCAE

Software Considerations in Airborne Systems and Equipment Certification
What is DO-178B/ED-12B?

RTCA

EUROCAE

Software Considerations in Airborne Systems and Equipment Certification
Verification is not simply testing. Testing, in general, cannot show the absence of errors.
Automatic Verification by Testing

Reviews and Analyses

EVERYTHING from:
- Development process
- Verification process

Mostly manual

Review = inspection
Analysis = evidence
Automatic Verification by Testing

Reviews and Analyses
EVERYTHING from:
• Development process
• Verification process
Mostly manual
Review = inspection
Analysis = evidence

Testing Process
Requirements-based
• Normal range
• Robustness
Test coverage
• Statement
• Branch
• MC/DC
Automatic Verification by Testing

Reviews and Analyses
EVERYTHING from:
• Development process
• Verification process
Mostly manual
Review = inspection
Analysis = evidence

Testing Process
Requirements-based
• Normal range
• Robustness
Test coverage
• Statement
• Branch
• MC/DC
Cost is Driving Interest for Correct-by-Construction

Kirstie Bellman, META project, Boeing
Formal Methods

One page in DO-178B

*Formal methods are complementary to testing.*

Reserved to problems which cannot be tested: *concurrency, distributed processing, redundancy management and synchronization*

*The use of formal specifications alone forces requirements to be unambiguous.*
Most of the errors in software development are now generally accepted as being attributable to errors in requirements (LLR or HLR).

FM Discussion Paper
Airbus Pioneering Work

HLR → LLR

code

tests
Airbus Pioneering Work

Pre:
X > 0
Post:
Y = X + 1

HLR  LLR  =  contracts

code

tests
Airbus Pioneering Work

HLR → LLR = contracts

Pre:
X > 0
Post:
Y = X + 1

code

proofs
Airbus / CEA LIST Joint Work


- Caveat: qualification at Airbus for use in A380 program
- Caveat: formal verification of C programs
- Frama-C + Why + Alt-Ergo: formal verification of C programs
- Formal Verification of Avionics Software Products (FM 2009):
  10 years of formal methods application summarized
- GPL release of Frama-C
- Airbus develops TASTER plug-in in Frama-C (coding standards)
Acknowledging Modern Development Techniques

DO-178C
Acknowledging Modern Development Techniques

DO-178C

Tools qualification document + Model based development supplement + Object-oriented & related tech. supplement + Formal methods supplement
Acknowledging Modern Development Techniques

- Tools qualification document
- Model based development supplement
- Object-oriented & related tech. supplement

DO-178C

Formal methods supplement
Formal methods [...] might be the primary source of evidence for the satisfaction of many of the objectives concerned with development and verification.

**Formal model** → **formal analysis** → **proofs**

- deductive methods
- model checking
- abstract interpretation
Formal methods [...] might be the primary source of evidence for the satisfaction of many of the objectives concerned with development and verification.

**UNAMBIGUOUS**

formal model

**SOUND**

formal analysis

deductive methods
model checking
abstract interpretation

**JUSTIFIED ASSUMPTIONS**

proofs
Benefits of Formal Modeling

FM Supplement & FM Discussion Paper

Improve requirements

Unambiguous description of requirements
Precise communication between engineers

Reduce error introduction

Objective verification evidence:
- one formal model (consistency and accuracy)
- between formal models (compliance)
Benefits of Formal Analysis

FM Supplement & FM Discussion Paper

Improve error detection

Detect exceptions and deadlocks
Detect unintended functions (wrongdoing)

Reduce effort

Non-interference (MILS)
WCET / bounded stack size
Correct (a)synchronous behavior
Relation to Coverage

Structural coverage analysis, however, is driven out of the impracticality of achieving exhaustive testing.

When only formal methods are used [...] alternative activities are required for coverage analysis.

FM Supplement
Industrial Examples
Airbus – Unit Proof

- Coding Phase
  - C Source
  - Definition of proof environment
    - Flows Generation
  - Flows
  - Caveat

- Design Phase
  - Data & control flows
  - Verification of Flows against Design
  - Caveat

- Functional Properties
  - Caveat

- Proof performing
  - Caveat

- Analysis of Proof Results
  - Process Management Tool
  - If OK

Source code compliant With Design

If not OK
Airbus – Example of Property

\[
\begin{align*}
\text{LET } \text{COND\_FCT} &= ( \\
& \forall k \in \text{int. } k>0 \text{ and } k \leq A1F2\_ZONE\_SIZE \Rightarrow \\
& \quad (A1F2\_Memory\_Zone[k] = 0xFF) \\
\end{align*}
\]

The initial value is correct for all the indexes

\[
\begin{align*}
\text{LET } \text{COND\_ERR} &= ( \\
& \exists k \in \text{int. } k>0 \text{ and } k \leq A1F2\_ZONE\_SIZE \text{ and} \\
& \quad (A1F2\_Memory\_Zone[k] \neq 0xFF) \\
\end{align*}
\]

There exists an index for which the initial value is wrong
AG(LEFT_DU_AVAILABLE -> LEFT_DU_APPLICATION != BLANK)
AG(RIGHT_DU_AVAILABLE -> RIGHT_DU_APPLICATION != BLANK)

In all reachable states, if the left DU is available, then its application shall not be blank.

AG( LEFT_DU_APPLICATION != MAP -> AX( LEFT_DU_APPLICATION = MAP -> CURSOR_LOCATION = LEFT_DU ) )

In any state which sets LEFT_DU_APPLICATION to MAP, the CURSOR_LOCATION must be LEFT_DU.
SHOLIS – Separation, Contracts and Run-time Errors

Z specification ➔ SPARK code + specification

Rigorous argument
- 150 proofs
- 500 pages

SPARK Examiner ➔ SPARK Simplifier
- 9000 VCs

SPARK Checker
procedure AddElementToLogFile
  (ElementID : in ElementType; Description : in DescriptionT);

  -- global in       Clock.Now;
  -- in out NumberLogEntries;
  -- derives AuditSystemFault,
  -- LogFiles from *
  -- Description &
  -- NumberLogEntries from *;

  -- pre NumberLogEntries < MaxLogEntries;
  -- post NumberLogEntries = NumberLogEntries~ + 1 and
  -- (LogFileEntries~(CurrentLogFile~) = Max ->
  -- LogFileEntries(CurrentLogFile) = 1;
Perceived Limitations of Formal Methods
today for...

Software

state-space explosion
alien to engineers
theory versus reality

FM Discussion Paper
Perceived Limitations of Formal Methods

today for... Software

state-space explosion
alien to engineers
theory versus reality

FM Discussion Paper

not long ago for... Hardware
capacity limited
difficult to use
lacking methodologies

SCDsource
Special Technology Report
Limitations of Unit Proof vs Unit Testing

- **Expertise**: required for writing contracts and carrying proof
- **Duplication**: “contract” not shared between testing and proof
- **Isolation**: unit test and unit proof cannot be combined
- **Confusion**: not the same semantics for testing and proof
- **Debugging**: contracts and proof cannot be executed
Solution: Executable Contracts

- User Input
- Inferred by Static Analysis
- Generated with Code from Model

Executable Annotation Language

- Testing
- Static Analysis
- Formal Verification
function Sqrt (X : Integer) return Integer with
  Pre  => X >= 0,
  Post => Sqrt’Result >= 0 and then
          Sqrt’Result ** 2 <= X and then
          (Sqrt’Result + 1) ** 2 > X,
Test_Case => (Name => "test case 1",
              Requires => X = 100,
              Ensures => Sqrt’Result = 10),
Test_Case => (Name => "robustness test case",
              Requires => X = -1,
              Ensures => Sqrt’Result = 0);
function Sqrt (X : Integer) return Integer with
Pre  => X >= 0,
Post => Sqrt’Result >= 0 and then
        Sqrt’Result ** 2 <= X and then
        (Sqrt’Result + 1) ** 2 > X,
Test_Case => (Name   => "test case 1",
               Requires => X = 100,
               Ensures  => Sqrt’Result = 10),
Test_Case => (Name   => "robustness test case",
               Requires => X = -1,
               Ensures  => Sqrt’Result = 0);
function Sqrt (X : Integer) return Integer with

[Pre] => X >= 0,
[Post] => Sqrt’Result >= 0 and then
Sqrt’Result ** 2 <= X and then
(Sqrt’Result + 1) ** 2 > X.

Test_Case => (Name => "test case 1",
Requires => X = 100,
Ensures => Sqrt’Result = 10),
Test_Case => (Name => "robustness test case",
Requires => X = -1,
Ensures => Sqrt’Result = 0);
function Sqrt (X : Integer) return Integer with
   Pre => X >= 0,
   Post => Sqrt’Result >= 0 and then
           Sqrt’Result ** 2 <= X and then
           (Sqrt’Result + 1) ** 2 > X,

Test_Case => (Name => "test case 1",
              Requires => X = 100,
              Ensures => Sqrt’Result = 10),

Test_Case => (Name => "robustness test case",
              Requires => X = -1,
              Ensures => Sqrt’Result = 0);
function Sqrt (X : Integer) return Integer with
  Pre  => X >= 0,
  Post  => Sqrt’Result >= 0 and then
           Sqrt’Result ** 2 <= X and then
           (Sqrt’Result + 1) ** 2 > X,
  Test_Case => (Name => "test case 1",
                Requires => X = 100,
                Ensures  => Sqrt’Result = 10),
  Test_Case => (Name => "robustness test case",
                Requires => X = -1,
                Ensures  => Sqrt’Result = 0);
Objectives of Project Hi-Lite

- Facilitate proofs of safety / security / properties
- One language of assertions
- Testing + proof + static analysis
- Subset of C or Ada code
- Mixed Ada-C code
Sketch of Communications Between Tools
## Open Project

<table>
<thead>
<tr>
<th>Project</th>
<th>FLOSS</th>
<th>Latest Open Release</th>
<th>Open BTS</th>
<th>Open mailing-list</th>
<th>Open VCS</th>
<th>Open dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARK</td>
<td>GPL</td>
<td>2010</td>
<td>GAP</td>
<td>one-way</td>
<td></td>
<td>2 teams</td>
</tr>
<tr>
<td>GCC (GNAT)</td>
<td>GPL</td>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPS</td>
<td>GPL</td>
<td>2010</td>
<td>GAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodePeer</td>
<td>GPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Why / Alt-Ergo</td>
<td>GPL / CeCILL-C</td>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frama-C</td>
<td>LGPLv2</td>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td>plug-ins</td>
</tr>
<tr>
<td>Hi-Lite</td>
<td>GPL</td>
<td>2011?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

While DO-178B was centered on testing, DO-178C allows formal verification instead of testing.

Past industrial applications have shown formal verification can be cost-effective.

To bridge the gap between unit proof (FM) and unit testing (engineers), project Hi-Lite defines executable contracts for C and Ada.