Static analysis of concurrent avionics software with AstréeA

Workshop on Static Analysis of Concurrent Software

David Delmas
Airbus
11 September 2016
Agenda

1. Industrial context
   - Avionics software
   - Formal methods

2. Static analysis of avionics software

3. Concurrent avionics software

4. Perspectives
Avionics Software at Airbus

We develop software for cockpit avionics computers

**Software functions**

**Aircraft Control Domain**
- flight controls, warning, communication systems (C, asm)

**Airline Information Services Domain**
- administrative functions, maintenance support (Java)

**Software platforms**

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATSU</td>
<td>LynxOS®-based POSIX Host Platform</td>
<td>x86</td>
</tr>
<tr>
<td>IMA</td>
<td>ARINC 653 Integrated Modular Avionics</td>
<td>PowerPC</td>
</tr>
<tr>
<td>ASF</td>
<td>PikeOS®-based Avionics Server Function</td>
<td>PowerPC</td>
</tr>
</tbody>
</table>
Agenda

1. Industrial context
   - Avionics software
     - Formal methods

2. Static analysis of avionics software
   - Static analysis of executables in today’s industrial processes
   - Static analysis of source code in today’s industrial processes
   - Ongoing technology transfers

3. Concurrent avionics software
   - The AstréeA project
   - Case studies and experiments

4. Perspectives
Software inside civil aircraft

Avionics software
- critical components of embedded systems
- e.g. flight-by-wire control
- major impact on safety
- widely used inside modern aircraft
Software embedded on Airbus aircraft

First real use of embedded software: at the beginning of the eighties.
Industrial context

**Static analysis of avionics software**

**Concurrent avionics software**

**Perspectives**

---

# Software inside civil aircraft

## Avionics software

- critical components of embedded systems
- e.g. flight-by-wire control
- major impact on safety
- widely used inside modern aircraft

## Certification

- by third parties on behalf of Authorities
  
  Federal Aviation Administration, European Aviation Safety Agency

- stringent rules on development and verification processes

- DO-178/ED-12 international standard

  Software Considerations in Airborne Systems and Equipment Certification
DO-178C/ED-12C – Verification Process

- **A-3.2 Accuracy & Consistency**
- **A-3.3 HW Compatibility**
- **A-3.4 Verifiability**
- **A-3.5 Conformance**
- **A-3.7 Algorithm Accuracy**

- **A-4.8 Architecture Compatibility**
- **A-4.9 Consistency**
- **A-4.10 HW Compatibility**
- **A-4.11 Verifiability**
- **A-4.12 Conformance**
- **A-4.13 Partition Integrity**

- **A-5.2 Compliance**
- **A-5.3 Verifiability**
- **A-5.4 Conformance**
- **A-5.6 Accuracy & Consistency**

- **A-5.7 Complete & Correct**

- **A-6.3 Compliance**
- **A-6.4 Robustness**
- **A-6.1 Compliance**
- **A-6.2 Robustness**
DO-178C/ED-12C – Verification Process – Level A

System Requirements

High-Level Requirements

Software Architecture

Low-Level Requirements

Source Code

Executable Code

A-3.2 Accuracy & Consistency
A-3.3 HW Compatibility
A-3.4 Verifiability
A-3.5 Conformance
A-3.7 Algorithm Accuracy

A-4.8 Architecture Compatibility
A-4.9 Consistency
A-4.10 HW Compatibility
A-4.11 Verifiability
A-4.12 Conformance
A-4.13 Partition Integrity

A-5.2 Compliance
A-5.3 Verifiability
A-5.4 Conformance
A-5.6 Accuracy & Consistency
A-5.7 Complete & Correct

A-5.1 Compliance
A-5.5 Traceability
A-5.7 Complete & Correct

A-6.1 Compliance
A-6.2 Robustness
A-6.3 Compliance
A-6.4 Robustness
A-6.5 Compatible With Target

A-6.3 Compliance
A-6.4 Robustness

A-7 Verification of verification
(Functional & Structural coverage)

A-3.1 Compliance
A-3.6 Traceability

A-4.1 Compliance
A-4.6 Traceability

A-4.2 Accuracy & Consistency
A-4.3 HW Compatibility
A-4.4 Verifiability
A-4.5 Conformance
A-4.7 Algorithm Accuracy

Compliance: with requirements
Conformance: with standards
With Independence
DO-178C/ED-12C – Verification Process – Level B

A-3.2 Accuracy & Consistency
A-3.3 HW Compatibility
A-3.4 Verifiability
A-3.5 Conformance
A-3.7 Algorithm Accuracy

A-4.1 Compliance
A-4.6 Traceability

A-4.2 Accuracy & Consistency
A-4.3 HW Compatibility
A-4.4 Verifiability
A-4.5 Conformance
A-4.7 Algorithm Accuracy

A-5.2 Compliance
A-5.3 Verifiability
A-5.4 Conformance
A-5.6 Accuracy & Consistency

A-5.1 Compliance
A-5.5 Traceability

A-6.3 Compliance
A-6.4 Robustness

A-6.1 Compliance
A-6.2 Robustness

A-6.5 Compatible With Target

A7 Verification of verification
(Functional & Structural coverage)
DO-178C/ED-12C – Verification Process – Level C

A-3.2 Accuracy & Consistency
A-3.3 HW-Compatibility
A-3.4 Verifiability
A-3.5 Conformance
A-3.7 Algorithm Accuracy

A-4.8 Architecture Compatibility
A-4.9 Consistency
A-4.10 HW-Compatibility
A-4.11 Verifiability
A-4.12 Conformance
A-4.13 Partition Integrity

A-5.2 Compliance
A-5.3 Verifiability
A-5.4 Conformance
A-5.6 Accuracy & Consistency

A-5.7 Complete & Correct

A-6.1 Compliance
A-6.2 Robustness
A-6.3 Compliance
A-6.4 Robustness
A-6.5 Compatible With Target

A-7 Verification of verification
(Functional & Structural coverage)

A-3.1 Compliance
A-3.6 Traceability

A-4.1 Compliance
A-4.6 Traceability

A-5.1 Compliance
A-5.5 Traceability

A-5.2 Compliance
A-5.5 Traceability

A-5.7 Complete & Correct
A-6.5 Compatible With Target

Compliance: with requirements
Conformance: with standards
No more Independence for Verifications
No more required
DO-178C/ED-12C – Verification Process – Level D

A-3.2 Accuracy & Consistency
A-3.3 HW-Compatibility
A-3.4 Verifiability
A-3.5 Conformance
A-3.7 Algorithm Accuracy

A-4.8 Architecture Compatibility
A-4.9 Consistency
A-4.10 HW-Compatibility
A-4.11 Verifiability
A-4.12 Conformance
A-4.13 Partition Integrity

A-5.2 Compliance
A-5.3 Verifiability
A-5.4 Conformance
A-5.6 Accuracy & Consistency
A-5.7 Complete & Correct

A-6.1 Compliance
A-6.2 Robustness
A-6.3 Compliance
A-6.4 Robustness
A-6.5 Compatible With Target

A-6.5 Complete & Correct
A-6.7 Complete & Correct
A-7 Verification of verification
(Only Functional coverage)

A-4.1 Compliance
A-4.2 Accuracy & Consistency
A-4.3 HW-Compatibility
A-4.4 Verifiability
A-4.5 Conformance
A-4.7 Algorithm Accuracy

A-4.1 Compliance
A-4.6 Traceability
A-4.1 Compliance
A-4.6 Traceability

A-3.1 Compliance
A-3.6 Traceability

A-3.1 Compliance
A-3.6 Traceability

A-2: 1, 2
(A-2: 1, 2)
(A-2: 3, 4, 5)
(A-2: 6)
(A-2: 7)

System
Requirements
High-Level
Requirements
Software
Architecture
Low-Level
Requirements
Source Code
Executable
Object Code

Compliance: with requirements
Conformance: with standards

No more Independence for Verifications
No more required

May 2015
DO-178C/ED-12C – Verification Process – Level E

No **safety** requirements
“when level E is accepted by the certification authority”

**But**, industrial constraints has to be satisfied

=> specific requirements are set up by aircraft manufacturer or suppliers (e.g. ABD 100 for Airbus)!
Industrial context

Avionics software
Formal methods

Static analysis of avionics software
Static analysis of executables in today’s industrial processes
Static analysis of source code in today’s industrial processes
Ongoing technology transfers

Concurrent avionics software
The AstréeA project
Case studies and experiments

Perspectives
The need for formal verification

- legacy methods: tests, reviews, and analyses
  ⇒ more than half of avionics software development costs
- no longer scale within reasonable costs
The need for formal verification

**Verification**
- legacy methods: tests, reviews, and analyses
  ⇒ more than half of avionics software development costs
- no longer scale within reasonable costs

**Some formal verification techniques**
- are useable on real-world industrial software
  mostly AI-based static analysis
  some WP-based program proof
- increasingly used
  in Airbus verification processes
to replace legacy methods
The requirement for soundness

DO-333 Formal Methods Supplement to DO-178C

- guidance on the use of formal techniques
- emphasizes soundness as the key criterion

“The soundness of each formal analysis method should be justified. A sound method never asserts that a property is true when it may not be true.”
### The requirement for soundness

**DO-333**

- guidance on the use of formal techniques
- emphasizes soundness as the key criterion

> "The soundness of each formal analysis method should be justified. A sound method never asserts that a property is true when it may not be true."

DO-333 acknowledges FM may be used to achieve...
The requirement for soundness

DO-333 acknowledges FM may be used to achieve

The objectives of **reviews and analyses of source code**

- by static analysis of source code
- provided a formal semantics is well-defined at source code level
### The requirement for soundness

**DO-333**

- Formal Methods Supplement to DO-178C
  - guidance on the use of formal techniques
  - emphasizes soundness as the key criterion

**DO-333** acknowledges FM may be used to achieve

**The objectives of reviews and analyses of source code**

- by static analysis of source code
- provided a formal semantics is well-defined at source code level

**Part of the objectives of testing**

1. by static analysis of binary code
2. by static analysis of source code

provided property preservation between source and executable code can be demonstrated
Summary of industrial and regulatory requirements for static analysis tools in the avionics domain

**Soundness** because human expertise cannot be replaced with unsound methods in safety-critical domains. Tools must undergo stringent qualification processes before they can be used for certification credit.

**Automation and precision** to replace legacy methods cost-efficiently.

**Scalability** because we deal with large industrial software.
Agenda

1. Industrial context
2. Static analysis of avionics software
   - Static analysis of executables in today’s industrial processes
   - Static analysis of source code in today's industrial processes
   - Ongoing technology transfers
3. Concurrent avionics software
4. Perspectives
<table>
<thead>
<tr>
<th>1</th>
<th>Industrial context</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avionics software</td>
</tr>
<tr>
<td></td>
<td>Formal methods</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>Static analysis of avionics software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static analysis of executables in today’s industrial processes</td>
</tr>
<tr>
<td></td>
<td>Static analysis of source code in today’s industrial processes</td>
</tr>
<tr>
<td></td>
<td>Ongoing technology transfers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>Concurrent avionics software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The AstréeA project</td>
</tr>
<tr>
<td></td>
<td>Case studies and experiments</td>
</tr>
</tbody>
</table>

| 4 | Perspectives |
AbsInt binary static analyzers used for certification

**WCET Analysis** on DAL A time-critical software products
- AI-based static analysis of executable code
- deployed on A330/A340/A380/A400M/A350
  - e.g. PowerPC MPC755 and 7448, TI TMS320C33
  - sequential/synchronous programs, up to 650 klo\(^C\)
- aiT WCET qualified wrt. DO-178B

**Stack Analysis** on all embedded software products
- AI-based static analysis of executable code
- deployed on all aircraft types
  - e.g. x86, PowerPC 755, 7448, 8610, TI TMS320C3x
  - also multithreaded asynchronous programs with complex data structures up to 2 Mlo\(^C\)
- StackAnalyzer qualified wrt. DO-178B
Industrial context
- Avionics software
- Formal methods

Static analysis of avionics software
- Static analysis of executables in today’s industrial processes
- **Static analysis of source code in today’s industrial processes**
- Ongoing technology transfers

Concurrent avionics software
- The AstréeA project
- Case studies and experiments

Perspectives
Run-time error analysis of synchronous C programs

Run-time errors

- overflows in float, integer, enum arithmetic and cast
- division, modulo by 0 on integers and floats
- invalid pointer arithmetic or dereferencing
- violation of user-specified assertions
## Run-time error analysis of synchronous C programs

### Run-time errors
- **overflows** in float, integer, `enum` arithmetic and cast
- **division, modulo by 0** on integers and floats
- **invalid pointer** arithmetic or dereferencing
- violation of user-specified **assertions**

### The ASTRÉE static analyzer
- developed by CNRS/ENS (from 2002) and AbsInt GmbH
- commercialised by AbsInt since 2010
Run-time error analysis of synchronous C programs

Run-time errors
- overflows in float, integer, enum arithmetic and cast
- division, modulo by 0 on integers and floats
- invalid pointer arithmetic or dereferencing
- violation of user-specified assertions

The ASTRÉE static analyzer
- developed by CNRS/ENS (from 2002) and AbsInt GmbH
- commercialised by AbsInt since 2010

Deployment at Airbus
- DAL A A380/A400M fly-by-wire (A350 soon)
- successful proofs of absence of run-time error
- $\approx$ 8 hours for 650,000 lines of C
“Local” static analyses
on small subsets of the call graph

Data & control flow analyses

- AI-based static analysis of C code
- first deployments on A350
- Fan-C (Airbus), a Frama-C (CEA-INRIA) plugin qualified wrt. DO-178B

Unit Proof on DAL A software subsets

- WP-based program proof at C function level
- deployed on A380/A400M/A350
- Caveat (CEA) + Alt-Ergo SMT-solver (INRIA) qualified wrt. DO-178B
- on-going effort to
  1. migrate to Frama-C/WP
  2. enlarge the scope of the deployment
Agenda

1. **Industrial context**
   - Avionics software
   - Formal methods

2. **Static analysis of avionics software**
   - Static analysis of executables in today’s industrial processes
   - Static analysis of source code in today’s industrial processes
   - Ongoing technology transfers

3. **Concurrent avionics software**
   - The AstréeA project
   - Case studies and experiments

4. **Perspectives**
Industrial context

Static analysis of avionics software

Concurrent avionics software

Perspectives

Numerical accuracy assessment of basic floating-point operators

- AI-based static analysis of $C$ source code
- experimental deployment on A350
- automates (manual) accuracy analyses
- FLUCTUAT (CEA)

Certified compilation from source to assembly

- CompCert (INRIA) certified compiler
  - formally verified semantic equivalence
- complementary to formal verification of $C$ code

Security/reliability analysis of Java software

- AI-based static analysis of Java bytecode
- Julia (Julia srl)
- DAL E airline information service software
Agenda

1. Industrial context
2. Static analysis of avionics software
3. Concurrent avionics software
   - The AstréeA project
   - Case studies and experiments
4. Perspectives
Industrial context

Static analysis of avionics software

Concurrent avionics software

Perspectives

Trend towards concurrency

The most safety-critical software

DAL A-B

e.g. bare-metal synchronous fly-by-wire

- sequential/synchronous safety-critical software

- increasingly addressed by sound static analysis
Trend towards concurrency

The most safety-critical software (DAL A-B)
- bare-metal synchronous fly-by-wire
  - sequential/synchronous safety-critical software
  - increasingly addressed by sound static analysis

Less critical software (DAL C-D-E)
- flight management, warning, communication, maintenance
  - increasingly integrated into generic avionics computers
  - Integrated Modular Avionics (IMA)
  - replaces buses with shared memory communications
  - real-time operating system (RTOS) provides:
    - robust time and space partitioning
    - (preemptive) single-core real-time scheduling
    - static resource allocation
  - implemented in asynchronous software

IMA Integrated Modular Avionics
A380/A400M/A350

RTOS provides
- ARINC 653 standard
- ≃ POSIX threads real-time
- threads, locks, memory
Issues

Verification

- **test**: ineffective due the huge number of thread interleavings
- human **reviews** and analyses: costly and error-prone
- source-level **formal methods**: lacking
Issues

Verification

- **test**: ineffective due the huge number of thread interleavings
- human **reviews** and analyses: costly and error-prone
- source-level **formal methods**: lacking

Available sound techniques

- **WCET Analysis**
- **Stack Analysis**
- Data & control flow analyses
- **Unit Proof**
- **Numerical accuracy assessment**
- Certified compilation
- **Security analysis for Java**
- **Run-time error analysis with ASTRÉEA**

memory abstracted as $\top$

assuming shared variables given
Agenda

1. Industrial context
   - Avionics software
   - Formal methods

2. Static analysis of avionics software
   - Static analysis of executables in today’s industrial processes
   - Static analysis of source code in today’s industrial processes
   - Ongoing technology transfers

3. Concurrent avionics software
   - The AstréeA project
   - Case studies and experiments

4. Perspectives
Static analysis of parallel C programs

Concrete semantics and specification

Model: real-time operating system

- **fixed** set of concurrent **threads** on a **single** processor
- **shared** memory
- **synchronisation** **primitives**
- **real-time scheduling** **with** **fixed** **priorities**
- e.g. A380/A400M/A350 IMA, A350 ASF, A320/A330/A340 ATSU
- **mono-threaded startup ≠ multi-threaded run**

- **Run-time errors**
  - classic C run-time errors (overflows, invalid pointers, etc.)
  - unprotected data-races
  - deadlocks
  - incorrect system calls

- **Concrete semantics and specification**
  - Model: real-time operating system
  - Fixed set of concurrent threads on a single processor
  - Shared memory (implicit communications)
  - Synchronisation primitives (fixed set of mutexes)
  - Real-time scheduling with fixed priorities (priority-based)
  - E.g. A380/A400M/A350 IMA, A350 ASF, A320/A330/A340 ATSU
  - Mono-threaded startup ≠ multi-threaded run (restriction)
Static analysis of parallel C programs

Concrete semantics and specification

Model: real-time operating system

- fixed set of concurrent threads on a single processor
- shared memory (implicit communications)
- synchronisation primitives (fixed set of mutexes)
- real-time scheduling with fixed priorities (priority-based)
  
  e.g. A380/A400M/A350 IMA, A350 ASF, A320/A330/A340 ATSU

- mono-threaded startup ≠ multi-threaded run (restriction)

Run-time errors

- classic C run-time errors (overflows, invalid pointers, etc.)
- unprotected data-races (report & factor in the analysis)
- deadlocks
- incorrect system calls
## ASTRÉEA

**Analyseur Statique de logiciels Temps-Réel Embarqués Asynchrones**

<table>
<thead>
<tr>
<th>Industrial context</th>
<th>Static analysis of avionics software</th>
<th>Concurrent avionics software</th>
<th>Perspectives</th>
</tr>
</thead>
</table>

The **ASTRÉEA** static analyser

- prototype extension of ASTRÉE developed by CNRS/ENS/INRIA since 2009
- industrialized by AbsInt since October 2015  
  
  see next talk
ASTRÉEA
Analyseur Statique de logiciels Temps-Réel Embarqués Asynchrones

The ASTRÉEA static analyser

- prototype extension of ASTRÉE developed by CNRS/ENS/INRIA since 2009
- industrialized by AbsInt since October 2015

The ASTRÉEA project

- cooperation between CNRS/ENS/INRIA and Airbus
- supported by French ANR

Goal

based on industrial case studies, specialise the analyser to make it precise for a family of large, complex multithreaded avionics software products
Agenda

1. Industrial context
   - Avionics software
   - Formal methods

2. Static analysis of avionics software
   - Static analysis of executables in today’s industrial processes
   - Static analysis of source code in today’s industrial processes
   - Ongoing technology transfers

3. Concurrent avionics software
   - The AstréeA project
   - Case studies and experiments

4. Perspectives
Main case study
Performed in an academic settings

while specialising the analyzer
A. Miné

a large ARINC 653 application:
- monitors aircraft functions and relays them to the pilots
- embedded avionics software (DAL C)
- 2.1 Mloc (2 Mloc generated)
- 15 threads, shared memory, locks
- preemptive real-time scheduling on a single processor
- reactive code + network code + lists, strings, pointers
- many variables, large arrays many (nested) loops
- no dynamic memory allocation, no recursivity
Main case study
Performed in an academic settings

while specialising the analyzer
A. Miné
Analysis context

Concrete execution context:

- **Target application, in C**
- **ARINC 653 operating system, in C+asm**
- **Other applications**
- **Hardware**

The target application:

- runs concurrently with other applications (memory separation)
- interacts dynamically with an ARINC 653 operating system (thread control, mutex lock and unlock, communication services)
- interacts with other applications through the OS
- creates system objects only during an initialization phase (the set of objects is inferred by the analysis)
Analysis context

Abstract analysis context:

- Target application, in C
- ARINC 653 model, in C + built-ins

The target application is enriched with hand-written model of the OS
- 2.6 Kloc of C + low-level AstréeA built-ins
- stub and simulate all OS system calls
- manage (fat) OS objects, mapped to (thin) AstréeA objects
  (e.g., AstréeA’s locks are simple integers, ARINC 653’s locks have a string name)

⇒ analyze stand-alone C programs, with no undefined symbol
Results

**Precision:** achieved through specialization

- 2010: **12,257** alarms
- 2015: **1,195** alarms (60% on hand-written code)
  - **99.94%** selectivity (% of lines without alarm)
  - **2302** directives added (to improve precision)

**Efficiency:**

- on an intel i7 2.90 GHz workstation (1 core used)
- computation time: **24h**
- analysis iterations: **6** (no widening needed on interferences)
- **27 GB** RAM
Second case study after analyzer specialisation
Performed in an industrial settings

a large ARINC 653 application from the same family
- 11 threads, similar structure
- 2 (non-consecutive) revisions analysed
  - S2.1 1.9 Mloc (1.8 Mloc generated)
  - S3.2 2.1 Mloc (2 Mloc generated)

Analysis adaptation
- ARINC 653 model reused (8% adaptation + 7 new primitives)
  - S2.1 2178 directives (adapted from main case study)
  - S3.2 5% adaptation wrt S2.1

Results
- S2.1 8573 alarms (99.56% selectivity)
- S3.2 10735 alarms (99.52% selectivity)
Third case study after analyzer specialisation
Performed in an industrial setting

A subset of a POSIX threads real-time application
- monitoring/correlation of maintenance related-information
- 7 threads, 32 Kloc (DAL E, 100% hand-written)
- pervasive string processing
- large arrays of structures, (nested) loops, pointer arithmetics

Analysis preparation
- 790 loc of stubs to abstract away the unanalysed (parts of) threads of the application
- development of a library of stubs for a subset of POSIX threads real-time
  45 API functions, 1.2 Kloc
- 197 ASTRÉE A directives to improve precision

Results
865 alarms (97.28% selectivity)
Fourth case study after analyzer specialisation
Performed in an industrial settings Airbus

Industrial context
Static analysis of avionics software
Concurrent avionics software
Perspectives

Fourth case study
Performed in an industrial settings

A subset of a POSIX threads real-time middleware
- data-link communication cockpit displays
- 1 process (4 threads), 33 Kloc (100% hand-written)
- filesystem, named pipes, shared memories
- very old version of POSIX

Analysis preparation
- important adaptations of stub library 70 API functions, 1 Kloc
- 228 ASTRÉE A directives to improve precision

Results
932 alarms (94.5% selectivity)
Summary

Main case study (academic settings) ⇒ analyser specialisation

Case studies in industrial settings

- enrich ARINC 653 stubs with newly used functions
- design a full set of POSIX threads stubs
- analysis precision tuning through end-user directives
- no modification of the analyzer
- constrained adaptation effort (1 to 4 man-month)
- reproduced with commercial ASTRÉE

<table>
<thead>
<tr>
<th>Case</th>
<th>Size</th>
<th>RTOS</th>
<th>Stubs</th>
<th>Selectivity</th>
<th>Time</th>
<th>Memory</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.1 M</td>
<td>A 653</td>
<td>5.2 K</td>
<td>99.94%</td>
<td>24 h</td>
<td>27 GB</td>
<td>acad</td>
</tr>
<tr>
<td>2.a</td>
<td>1.9 M</td>
<td>A 653</td>
<td>2.4 K</td>
<td>99.56%</td>
<td>154 h</td>
<td>18 GB</td>
<td>indus</td>
</tr>
<tr>
<td>2.b</td>
<td>2.2 M</td>
<td>A 653</td>
<td>2.3 K</td>
<td>99.52%</td>
<td>160 h</td>
<td>23 GB</td>
<td>indus</td>
</tr>
<tr>
<td>3</td>
<td>31.8 K</td>
<td>POSIX</td>
<td>2.2 K</td>
<td>97.28%</td>
<td>50 mn</td>
<td>0.6 GB</td>
<td>indus</td>
</tr>
<tr>
<td>4</td>
<td>33.1 K</td>
<td>POSIX</td>
<td>1.2 K</td>
<td>97.18%</td>
<td>35 h</td>
<td>2.5 GB</td>
<td>indus</td>
</tr>
</tbody>
</table>

selectivity only slightly worse than for the main case study
⇒ towards a cost-effective industrial use of AstréeA
## Agenda

1. Industrial context
2. Static analysis of avionics software
3. Concurrent avionics software
4. Perspectives
On-going work

Towards zero alarm

Precision target for cost-efficient deployment (avionics domain)

- 99.80% selectivity on hand-written code
  (currently: 95.97% to 99.2%)
- 99.99% selectivity on automatically generated code
  (currently: 99.78% to 99.98%)

⇒ not quite there yet, but reasonable goal

Multicore

Sound and precise analysis for weak consistency memory models
Industrial deployment

First scope: automate reviews and analyses of source code

1. “Accuracy and consistency”
   - absence of run-time error
   - probably on a subset-basis for huge applications

2. “Compliance with the software architecture”
   - check the set of threads, locks, and shared objects
   - access modes by each thread

This will require qualification wrt DO-178C/DO-333.

Longer term: alleviate robustness testing

this will require

1. a more demanding qualification level
2. a formal proof that the semantics is preserved on the compiled code
   - extension of CompCert’s proof?
Thank you for your attention.

Questions?