# FORMAL METHODS FOR AEROSPACE APPLICATIONS

Guillaume Brat (NASA), Eric Feron (Georgia Tech, USA), Pierre-Loic Garoche (Onera), Pete Manolios (Northeastern University), and Marc Pantel (IRIT)

Cambridge, UK FMCAD October 2012

1

## The Friendly Skies

2


Take-off, Landing & During Movement
Water Evacuation
Oxygen Supply
Land & Water Evacuation Exits
EXIT OPERATION
ASSESS OUTSIDE CONDITIONS. IF OPENING EXIT INCREASES THE HAZARD TO PASSENGERS, DO NOT OPEN.
Bracing Positions
Slide Evacuation

3

# Outline


787 DREAMLINER

4

# Outline

5

# Regulatory Environment

- Certification required by FAA/ EASA/ Transport Canada
- Safety/certification guidelines: ARP 4761, 4754
- RTCA DO-178C/ EUROCAE ED-12C: Software Considerations in Airborne Systems and Equipment Certification
  - Approved in 12/2011; FAA approval pending
  - Replaces DO-178B, last revised 12/1992
  - Software design process assurance
  - Planning, development, verification
  - Design Assurance Level A (catastrophic) 71 objectives
- Systems get certified, not components
- Certification is expensive

6

# Observations

- Current standards are mostly process-oriented

  *"Because we cannot demonstrate how well we've done, we'll show how hard we've tried."* John Rushby
- Incidents, but no crashes due to software.
  - May 1997, AA Flight 903: monitoring error led to blank displays while pilots were recovering from a succession of stalls
- Testing: *"Today's certification regimes ... rely on testing, which cannot provide sufficient evidence"* *
- Culture: *"Much of the benefit of ... DO-178B ... may be due to the safety culture that their strictures induce"* *

*Software for Dependable Systems. National Research Council of the National Academies. Edited by Jackson et al (2007)

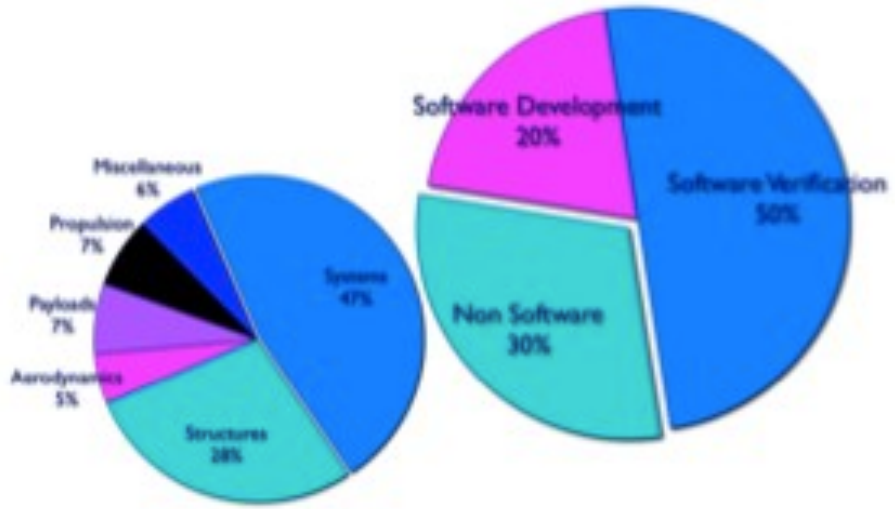
7

# Formal Methods Supplement

- Supplement to DO-178C (not DO-178B)
- Motivation
  - Recognition of limits of testing
  - Recognition of progress in formal methods
- Formal: *"All notations...should be verified to have precise, unambiguous, mathematically defined syntax and semantics"*
- Formal analyses: *"(1) deductive methods, such as theorem proving, (2) model checking, and (3) abstract interpretation"*
- Sound analyses: *"A sound method never asserts that a property is true when it may not be true"*
- Conservative: *"...demonstrate that the formal statement is a conservative representation of the informal requirement"*

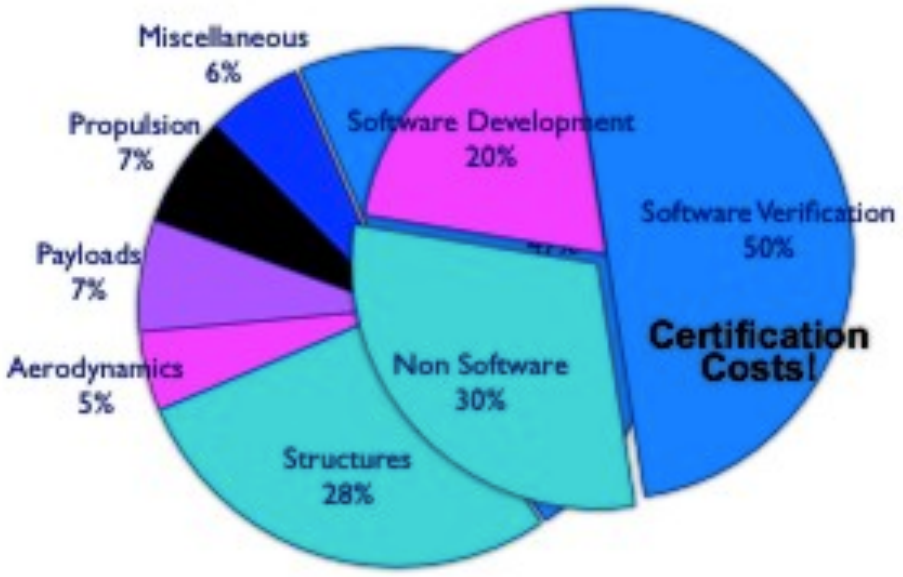
8

# Outline




9

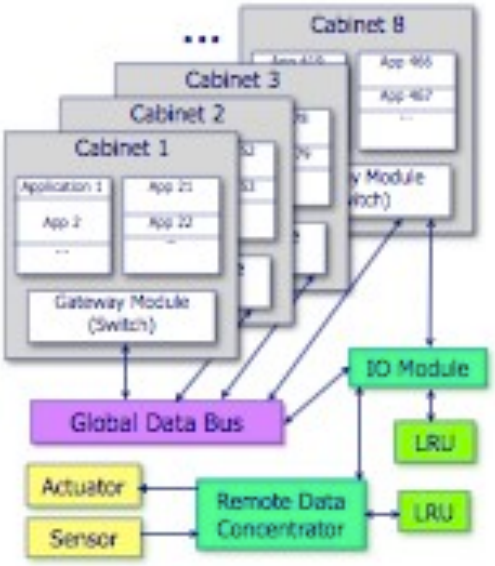
# Development Costs for 777




10

# Integrated Modular Avionics

- Past: federated systems
- IMA: shared resources
- COTS components
- Multiplexed communication
- Smaller, lighter, cost-effective components
- Powerful computer processing modules handle multiple apps
- Cabinets provide the various resources and interfaces
- Cabinets are connected to global data bus, IO modules, LRUs, sensors, actuators, etc






# Managing Complexity

- Fly-by-wire: >10 MLOC
- Raise the level of discourse
- Utilize abstraction
- Architectural level
- Interacting components
- Errors here are costly
- Enable design exploration

# System Assembly


- From a pool of components:
  - Select
  - Connect
  - Integrate
  - Assemble
- Subject to global requirements

Boeing core competency (out of 3):

*"Large-scale systems integration: We will continuously develop, advance, and protect the technical excellence that allows us to integrate effectively the systems we design and produce."*

13

# Outline


14

# Case Study


787 production design

Provided by Boeing

Thanks to John Chilenski

15

# CoBaSA

- General purpose tool for a broad class of problems
- OO design language w/ domain specific extensions
- Declarative specification language
- Goal: automatically synthesize architectural models
- *What*, not *how!*
- Enables design exploration [MSV07]

16

# Components

- Hardware
  - Cabinets provide resources (CPU, memory, ...)
  - Sensors, actuators, network switches, ...
- Software
  - Applications, *e.g.*, navigation guidance & control, collision detection, ...
  - Global memory spaces for sharing data

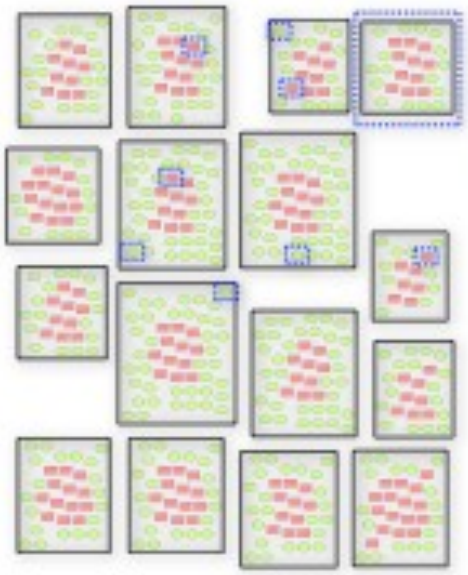


# Components in CoBaSA

```
entity cab {
  id: string;
  cpu_avail: int;
  ram_avail: int;
  ...
};

entity app {
  id: string;
  cpu_req: int;
  ram_req: int;
  ...
};

entity gmem {
  id: string;
  ram_req: int;
  ...
};
```

```
var cab_1 = cab{"cab_1"; 1000000; 536870912;...};
var cab_2 = cab{"cab_2"; 1200000; 536870912;...};
...
var cabs = [cab_1; cab_2; ...];

var app_1 = app{"app_1"; 16800; 9095861;...};
var app_2 = app{"app_2"; 12800; 8265143;...};
...
var apps = [app_1; app_2; ...];

var gmem_1= gmem{"gmem_1"; 351893; ...};
var gmem_2= gmem{"gmem_2"; 3462821; ...};
...
var gmems = [gmem_1; gmem_2; ...];
```

# Resource and Structural Constraints



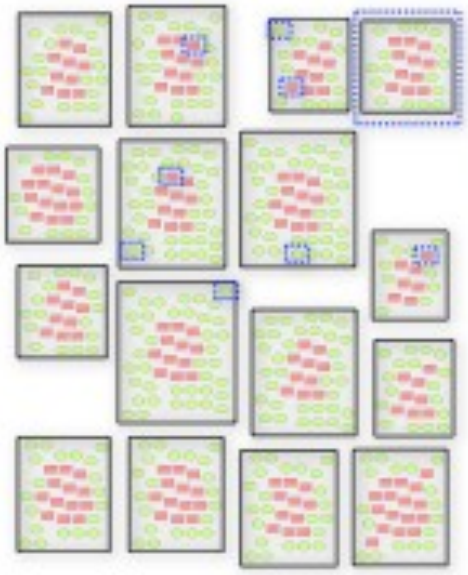
Map apps and memory spaces to cabinets, subject to

- resource requirements

```
map m apps cabs
  [cpu_req, ram_req, ...]
  [cpu_avail, ram_avail, ...];
```

19

# Resource and Structural Constraints



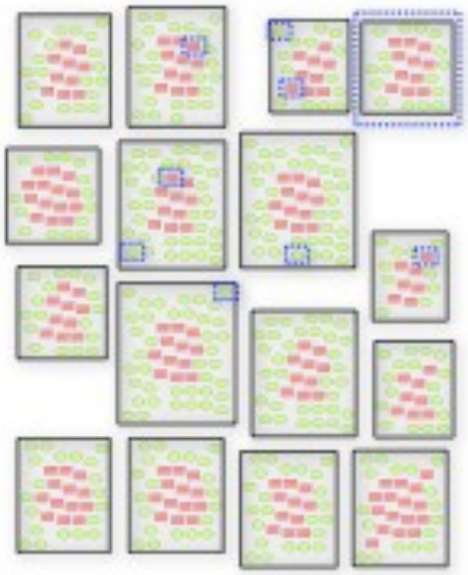
Map apps and memory spaces to cabinets, subject to

- resource requirements
- fixed apps/ memory spaces
- co-location constraints
- separation constraints

```
forall i in cabs:
  m(app_61, i) implies
  not m(app_183, i);
```

20

# Resource and Structural Constraints



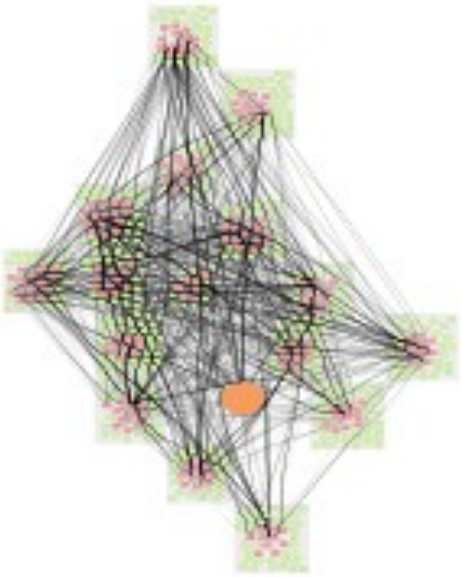
Map apps and memory spaces to cabinets, subject to

- resource requirements
- fixed apps/ memory spaces
- co-location constraints
- separation constraints
- fault-tolerance constraints
- replication of memory spaces

```
gmap >= 1 gm gmems cabs
  [ram_req, ...]
  [ram_avail, ...];
```

21

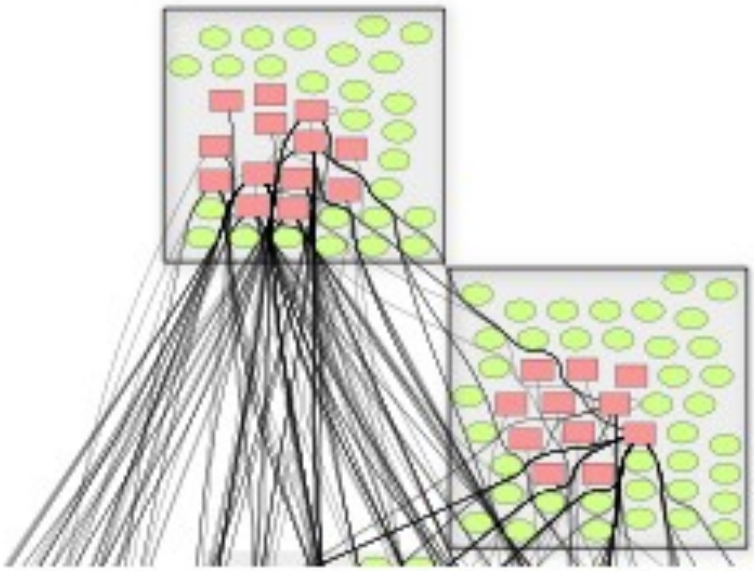
# Publish-Subscribe Network



- 1200 *virtual links*: aggregate messages and multicast
- Interaction between apps and the external network
- Virtual links consume bandwidth

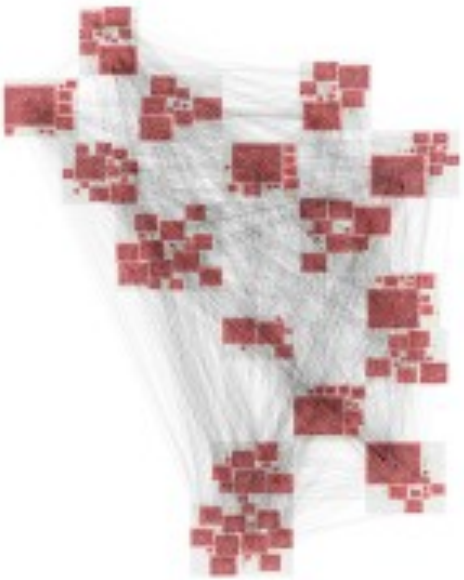
22

# Publish-Subscribe Network



23

# Messages



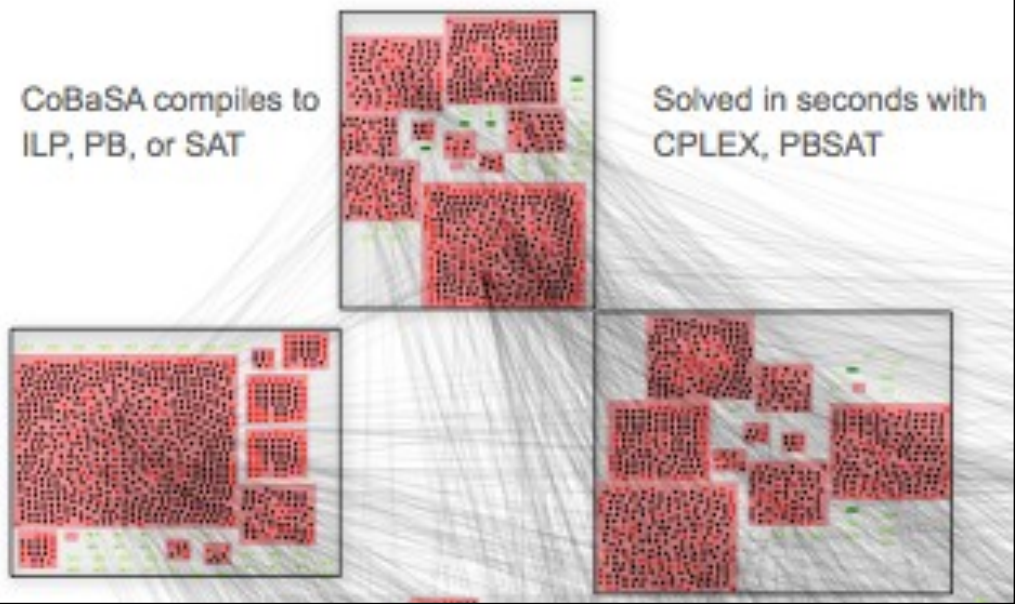
- Virtual links consist of messages
- 6000 messages and associated constraints
- Separate buffers for message transmission & reception
- Sampling and queueing buffering modes

24

# Messages

CoBaSA compiles to ILP, PB, or SAT

Solved in seconds with CPLEX, PBSAT



25

# Hard Real Time!

- Display units cannot lag
- Collect messages from sensors; send to actuators in real time
- Collision avoidance system should really get its CPU cycles
- Hard real time constraints for all applications (see [HMP11])

26

# Static Cyclic Scheduling Constraints


27

# Static Cyclic Scheduling

- Time is divided into same-sized slots
- A job is represented as a pair ⟨rate,cost⟩
  - rate: # of executions (per second)
  - cost: # of slots per execution
- Jobs are non-preemptive
- A schedule of job is the first slot it occupies
  - determines infinitely many slots occupied by job
- A multiset of jobs is schedulable iff:
  - there exists a schedule for each job
  - with no collisions

28

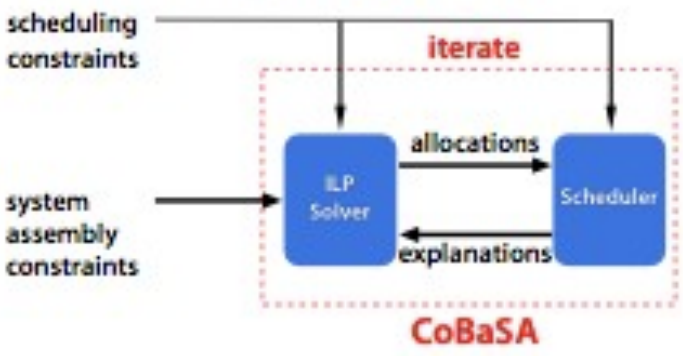
# Uniprocessor Case

- All jobs have rate 1 and arbitrary costs, except one job that has rate 2 and cost 1
- NP-complete (set-partition problem)
- Strongly NP-complete (3-partition problem)
- Hard, *but* enables downstream analyses


29

# ILP Modulo Theories (IMT)

- ILP where symbols have meaning in background theories
- Native support for optimization
- Better than SAT/SMT for certain problems
- BC*(T)*: Branch and Cut abstract transition system
- Complete for stably-infinite theories



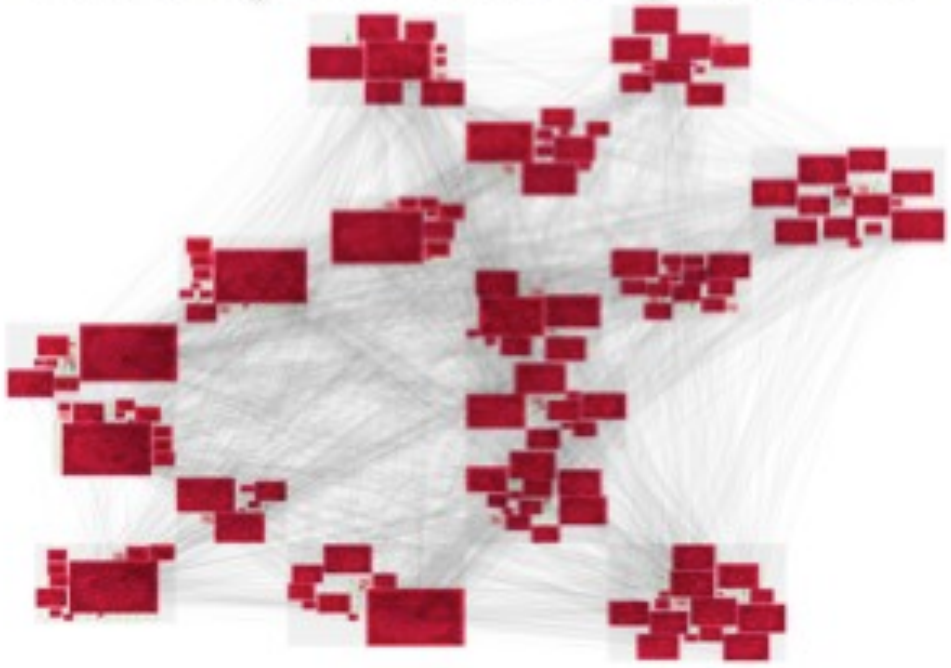

30

## Resource Limits

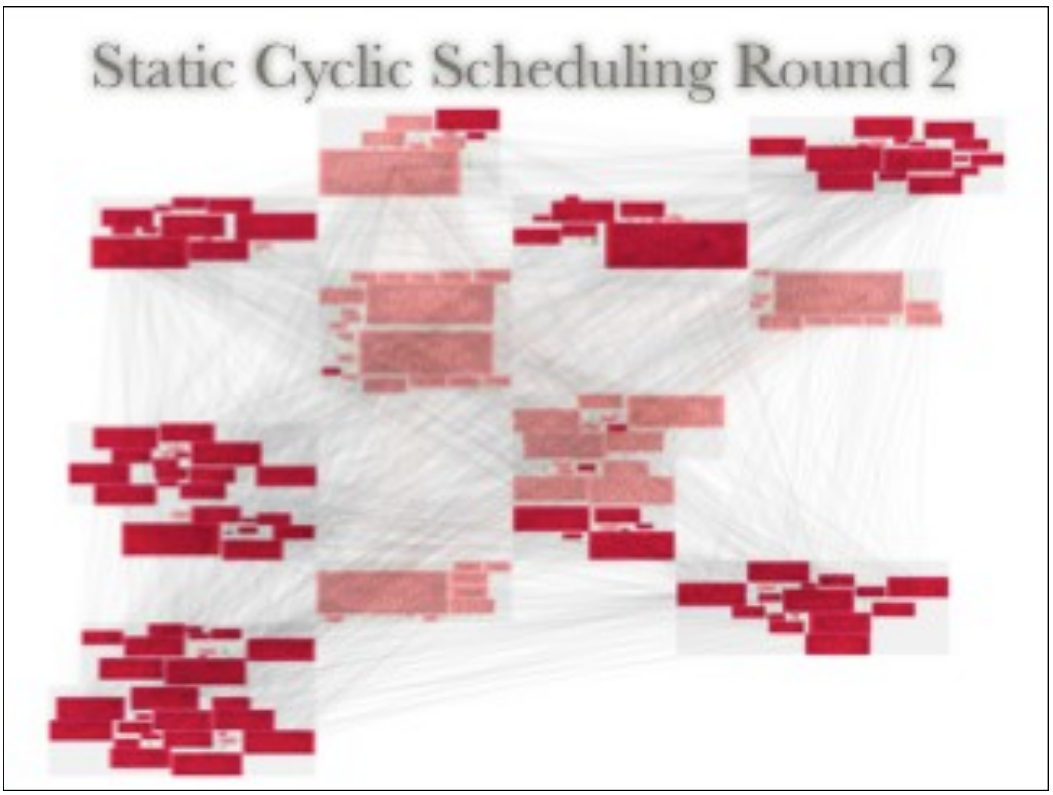
- Idea: synthesize, but avoid hard scheduling instances
- Call the theory solver with resource limits (time)
- Special kind of lemmas when resources exceeded
- Reversion mechanism if we get unsat
- Now we need limits for the core solver too!
- Automatic work balancing of core and theory solver
- Without resource limits, we cannot solve problem

31

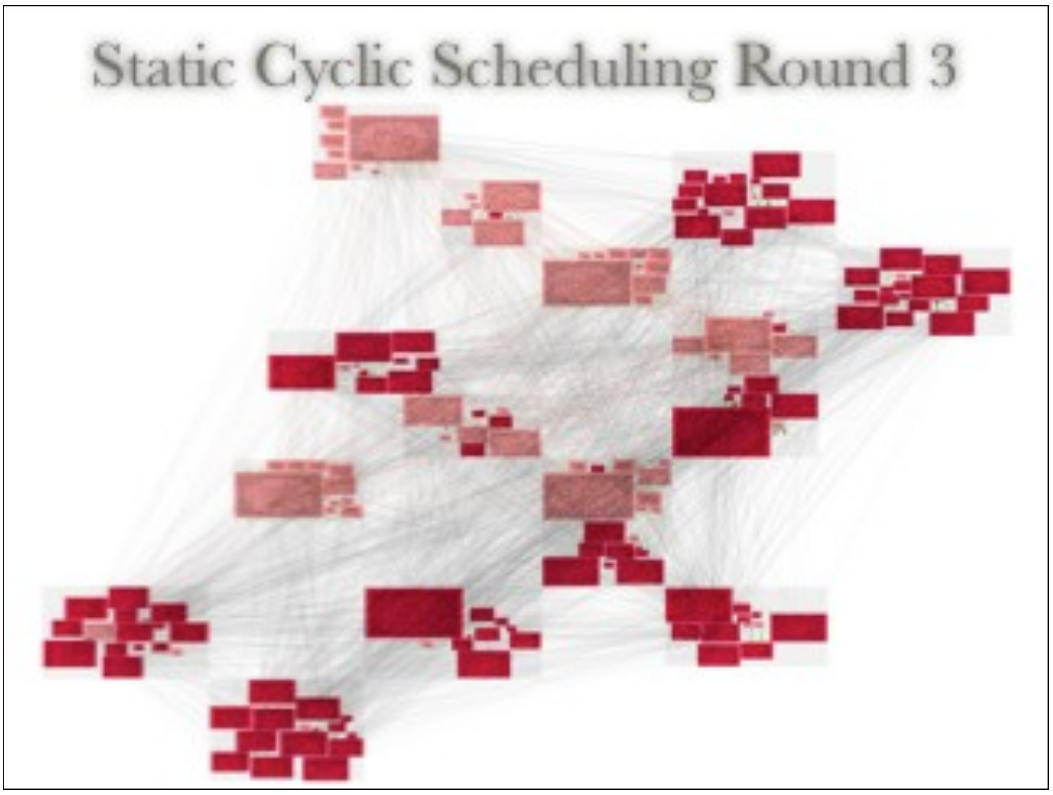
## Static Cyclic Scheduling Round 1



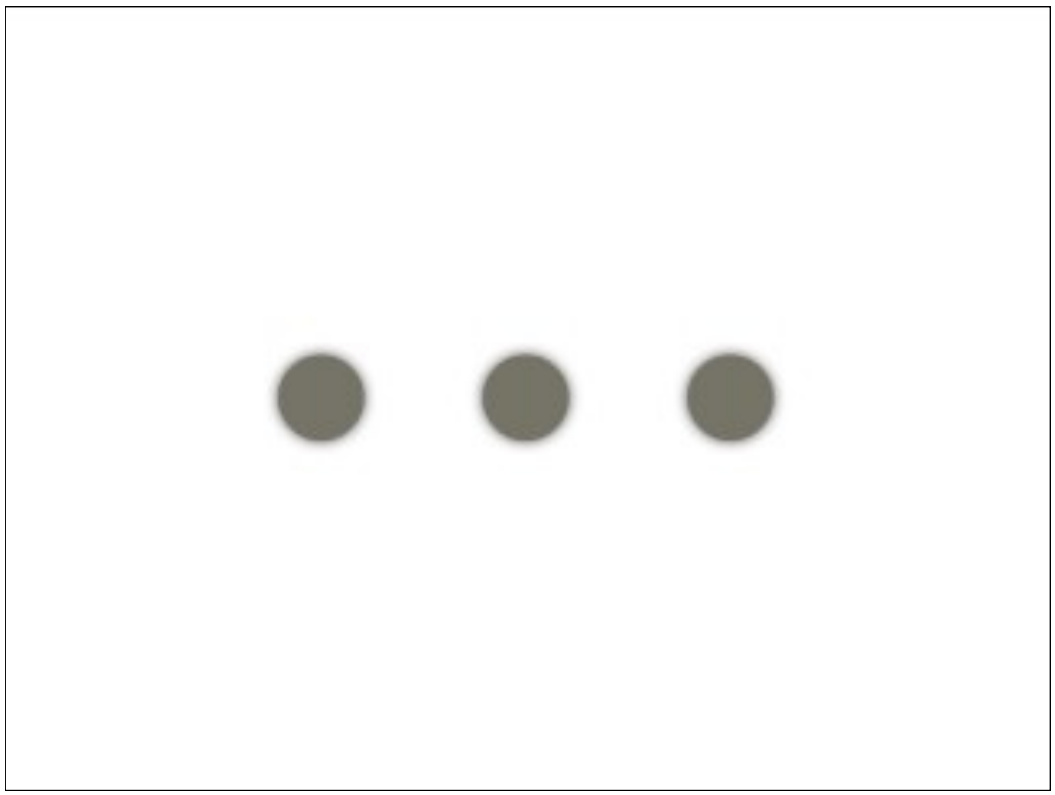
32


Static Cyclic Scheduling Round 2

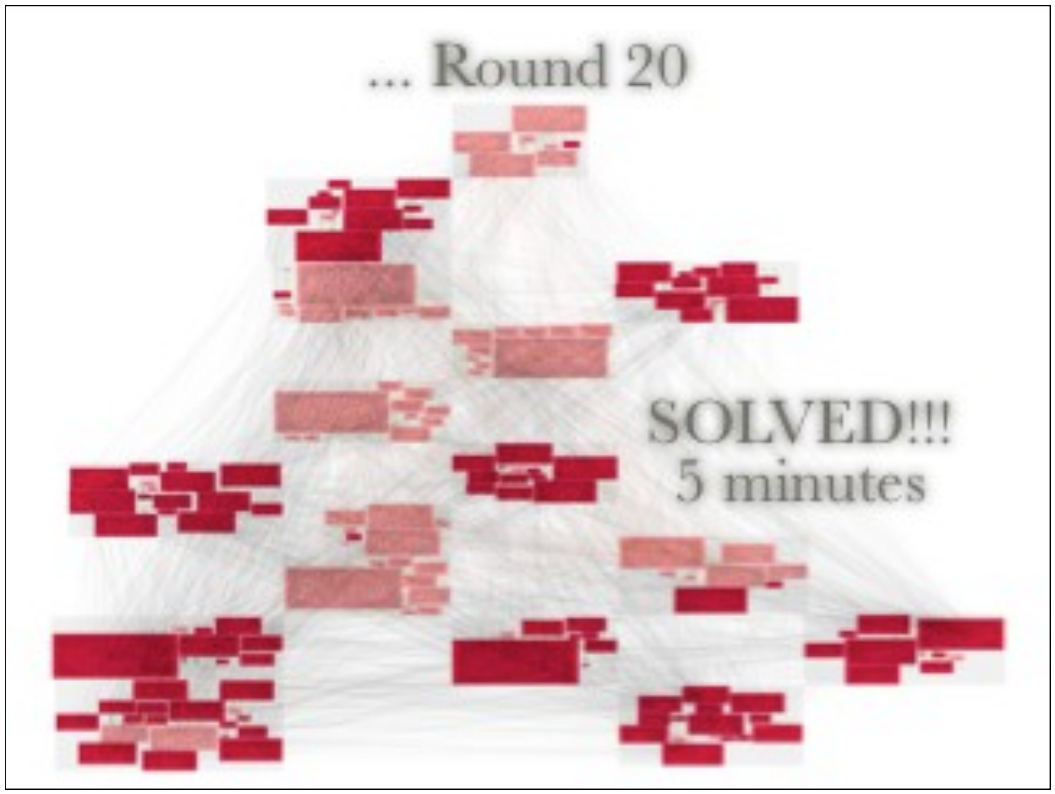
33


Static Cyclic Scheduling Round 3

34



35




36

# Conclusions

- Software accounts for 1/3 of airplane design costs
- Algorithmically synthesized *optimal* architectural models for 787
  - Using CoBaSA, IMT
  - The system consists of millions of LOC
  - Previously, *"required multiple groups working closely together over long periods of time"*
- Formal methods are increasingly important
  - Science of software
  - Model-based design
  - DO-178C supplement on formal methods
  - Safety critical applications: Avionics, medical devices, ...

ACADIE

CNRS - INPT - UPS - UT1 - UTM

# FMCAD'2012, Cambridge
# Formal methods in Avionics tutorial

22$^{th}$ october 2012


Marc Pantel – ACADIE team

Assistance à la Certification
d'Applications DIstribuées et Embarquées

# Safe MDE concerns



- Main purpose : Safety critical systems
- Main approach : formal specification and verification
- Problems : expressiveness, decidability, completeness, consistency

- Proposals : Raise abstraction
  - Higher level programming languages and frameworks
  - Domain specific (modeling) languages
    - easy to access for end users
    - with a simple formal embedding
    - with automatic verification tools
    - with usefull validation and verification results
    - that are accepted by certification authorities
- Needs :
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

# Safe MDE concerns

DO178/ED12 safety standards

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- Main purpose : Safety critical systems
- Main approach : formal specification and verification
- Problems : expressiveness, decidability, completeness, consistency

- Proposals : Raise abstraction
  - Higher level programming languages and frameworks
  - Domain specific (modeling) languages
    - easy to access for end users
    - with a simple formal embedding
    - with automatic verification tools
    - with usefull validation and verification results
    - that are accepted by certification authorities

- Needs :
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

# Safe MDE concerns

DO178/ED12 safety standards

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- Main purpose : Safety critical systems
- Main approach : formal specification and verification
- Problems : expressiveness, decidability, completeness, consistency
- Proposals : Raise abstraction
  - Higher level programming languages and frameworks
  - Domain specific (modeling) languages
    - easy to access for end users
    - with a simple formal embedding
    - with automatic verification tools
    - with usefull validation and verification results
    - that are accepted by certification authorities
- Needs :
  - methods and tools to ease their development
  - algebraic and logic theoretical fondations
  - proof of transformation and verification correctness
  - links with certification/qualification

# Related past projects

DO178/ED12 safety standards

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- RNTL COTRE : Transformation to verification languages
- ACI FIACRE : Intermediate verification language
- ITEA GeneAuto : Qualified Simulink/Stateflow to C code generator
- ITEA ES_PASS : Static analysis for Product insurance
- ITEA SPICES : AADL behavioral annex
- ANR OpenEmbedd : AADL to FIACRE verification chain (Kermeta based)
- CNES (French Space Agency) AutoJava : profiled UML to RTSJ code generator
- FUI TOPCASED : Metamodels semantics, Model animators, Verification chains based on model transformations
- ANR SPaCIFY : GeneAuto + AADL = Synoptic $\leftrightarrow$ Polychrony (Kermeta based)
- ANR iTemis : SOA/SCA verification
- JTI ARTEMIS CESAR : V & V view for safety critical components.

# Related current projects

DO178/ED12 safety standards

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- FRAE quarteFt : model transformation based on Java/TOM for AADL to FIACRE
- ITEA2 OPEES : Formal methods and Certification authorities
- FUI Projet P : Qualified code Generation for Functional and Architecture models
- EuroStars HiMoCo : High Integrity Model Compilers
- ITEA2 openETCS : Formal specification and verification for Train Control Systems
- ANR INS GeMoC : Execution of heterogeneous models
- ANR ASTRID VORACE :

# DO-178/ED-12 safety standards : Certification



- Onboard software in aeronautics : Design Assurance Level
  Failure impact : DAL A – Catastrophic failure . . .DAL E – No impact
- Early releases in the 80s, major revision in 1992 (B – 3 years of work), and 2012 (C – 7 years of work) : adaptation to technological changes
- Most constraining standard up to now
  accepted by other standards (automotive, space, . . .)
- Main concern : Safety of passengers
  System requirement : $10^{-9}$ per flight hour for DAL A – ARP 4754
- Main purpose :
  Provide confidence in the system and its development

# DO-178/ED-12 safety standards : Certification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Onboard software in aeronautics : Design Assurance Level
  Failure impact : DAL A – Catastrophic failure . . .DAL E – No impact
- Early releases in the 80s, major revision in 1992 (B – 3 years of work), and 2012 (C – 7 years of work) : adaptation to technological changes
- Most constraining standard up to now
  accepted by other standards (automotive, space, . . .)
- Main concern : Safety of passengers
  System requirement : $10^{-9}$ per flight hour for DAL A – ARP 4754
- Main purpose :
  Provide confidence in the system and its development

# DO-178/ED-12 safety standards : Certification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Key issue :
  Choose the strategy and technologies that will minimize risks
- Assessment : Stochastic for system, Zero-default for Software
- Process and test-centered approach
  - Definition of a precise process (development/verification)
  - MC-DC test coverage for DAL A
    truth-table lines of sub-expressions in conditions (some can be merged)
  - Asymmetry with independence argument : several activities (and products) by different teams, with different tools, . . .

# Process centered approach



- Requirement : What is expected from a system
  - High level (HLR) : focus on end users needs (user provided)
  - Low level (LLR) : focus on technical solutions (developer provided)
- Traceability : Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification : System fulfills its requirements explicit specification (make the right product)
- Validation : System fulfills its requirements implicit human needs (make the product right)
- Certification : System (and its development) follows standards (safety in our case : DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification : Tools for system development follows standards
- Certification and qualification : Historically, system context related (no component, COTS, reuse, . . .up to DO-178C/ED-12C)

# Process centered approach



- Requirement : What is expected from a system
  - High level (HLR) : focus on end users needs (user provided)
  - Low level (LLR) : focus on technical solutions (developer provided)
- Traceability : Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification : System fulfills its requirements explicit specification (make the right product)
- Validation : System fulfills its requirements implicit human needs (make the product right)
- Certification : System (and its development) follows standards (safety in our case : DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification : Tools for system development follows standards
- Certification and qualification : Historically, system context related (no component, COTS, reuse, . . .up to DO-178C/ED-12C)

# Process centered approach

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Requirement : What is expected from a system
  - High level (HLR) : focus on end users needs (user provided)
  - Low level (LLR) : focus on technical solutions (developer provided)
- Traceability : Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification : System fulfills its requirements explicit specification (make the right product)
- Validation : System fulfills its requirements implicit human needs (make the product right)
- Certification : System (and its development) follows standards (safety in our case : DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification : Tools for system development follows standards
- Certification and qualification : Historically, system context related (no component, COTS, reuse, . . .up to DO-178C/ED-12C)

# Process centered approach



- Requirement : What is expected from a system
  - High level (HLR) : focus on end users needs (user provided)
  - Low level (LLR) : focus on technical solutions (developer provided)
- Traceability : Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification : System fulfills its requirements explicit specification (make the right product)
- Validation : System fulfills its requirements implicit human needs (make the product right)
- Certification : System (and its development) follows standards (safety in our case : DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification : Tools for system development follows standards
- Certification and qualification : Historically, system context related (no component, COTS, reuse, . . .up to DO-178C/ED-12C)

# Process centered approach



- Requirement : What is expected from a system
  - High level (HLR) : focus on end users needs (user provided)
  - Low level (LLR) : focus on technical solutions (developer provided)
- Traceability : Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification : System fulfills its requirements explicit specification (make the right product)
- Validation : System fulfills its requirements implicit human needs (make the product right)
- Certification : System (and its development) follows standards (safety in our case : DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification : Tools for system development follows standards
- Certification and qualification : Historically, system context related (no component, COTS, reuse, . . .up to DO-178C/ED-12C)

# DO-178/ED-12 Global process

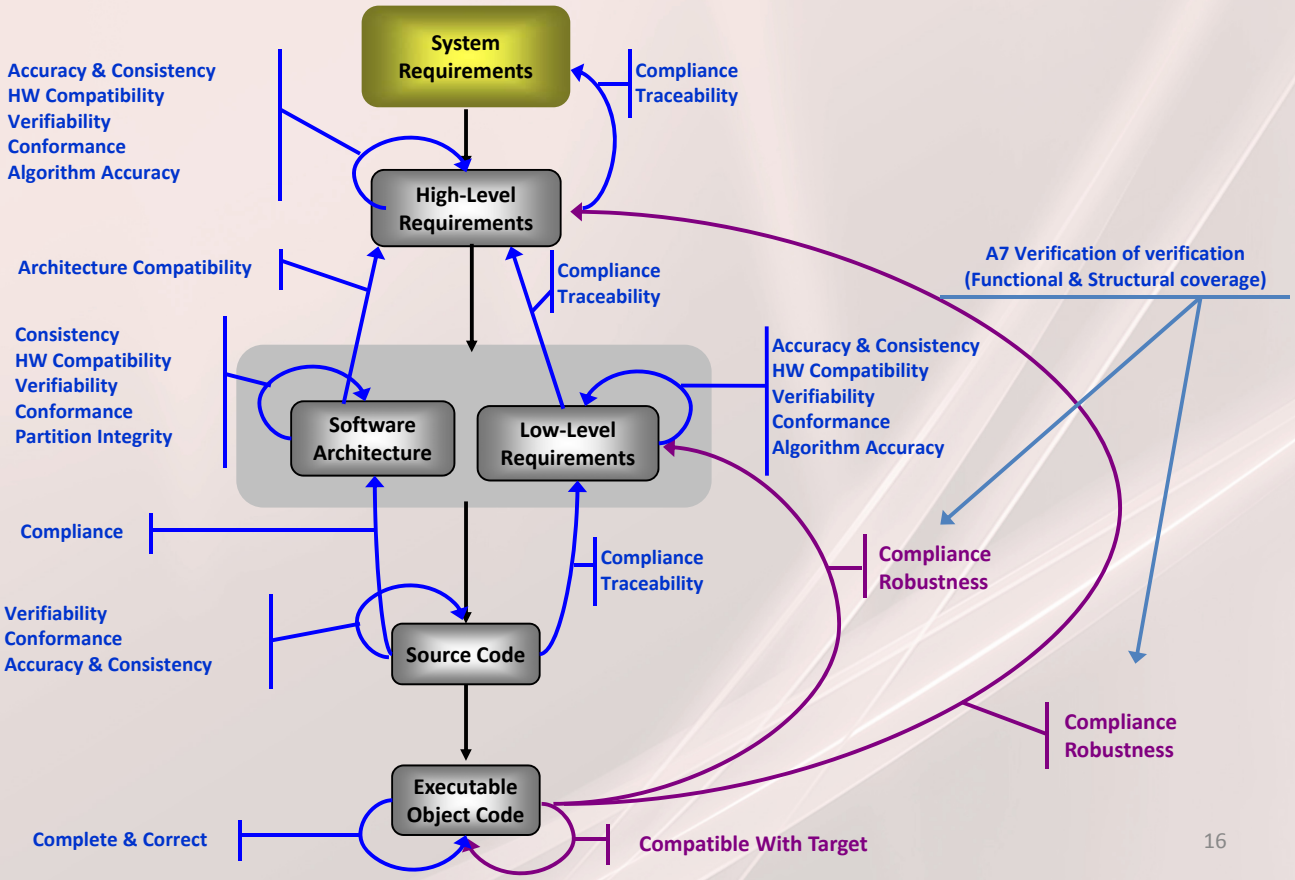
ACADIE

**DO178/ED12 safety standards**

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?



# Phase 1 : Process definition and early certification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Plan for Software Aspects of Certification (PSAC)
- Software Development Plan (SDP)
- Software Verification Plan (SVP)
- Software Configuration Management Plan (SCMP)
- Software Quality Assurance Plan (SQAP)
  applied only to the other plans
- Tool Qualification Plan (TQP)
  it tools are used to automatize activities

# DO-178/ED-12 Software verification process

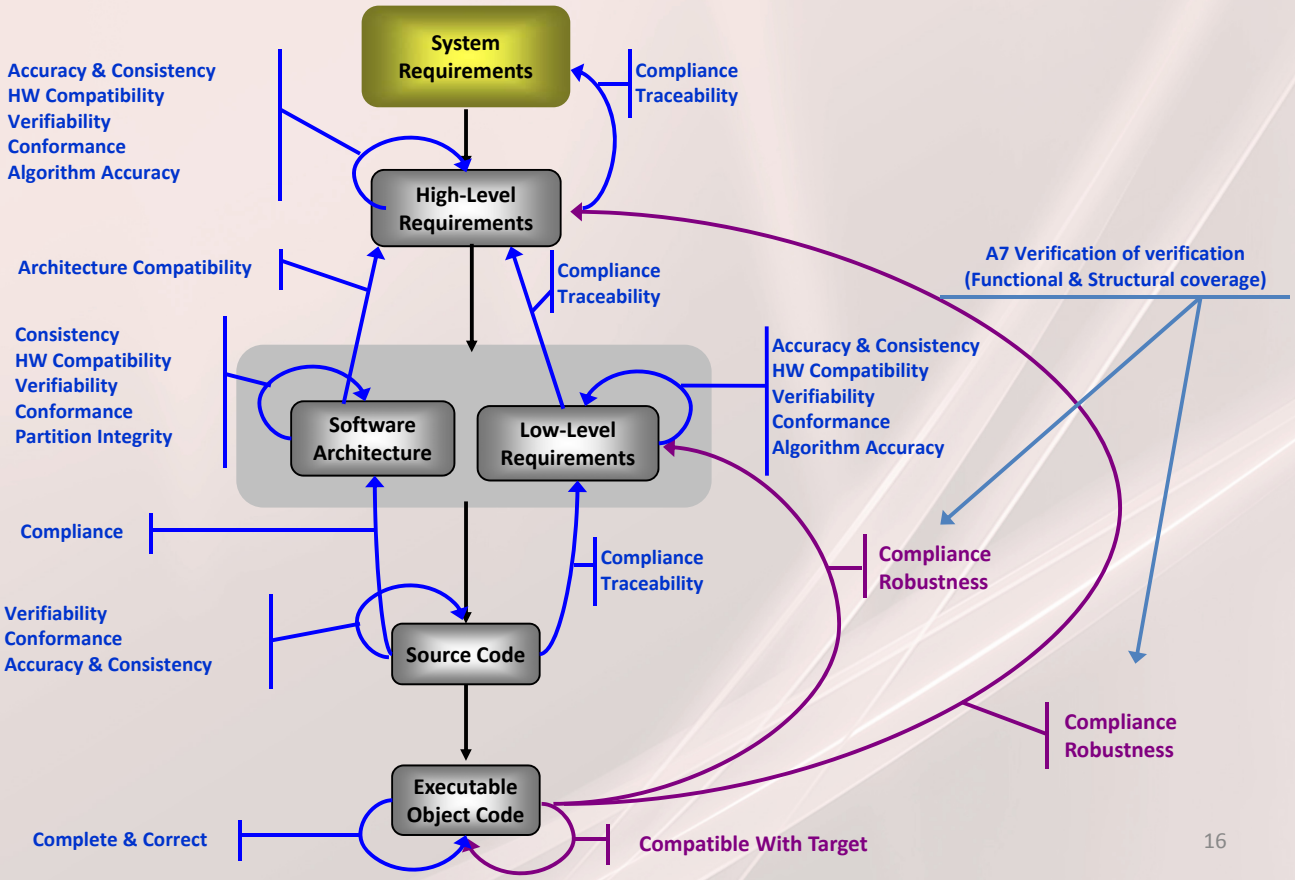
ACADIE

**DO178/ED12 safety standards**

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?



# Phase 2 : Process application verification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- User requirements (HLR)
- Software architecture (elementary parts and their assembly)
- Software requirements (Detailled design of elementary parts) : Can be refined user requirements or derived requirements (linked to technology choices, should be avoided or strongly justified)
- Executable Object Code (EOC) integration on Hardware
- Verification results
- Traceability links between requirements and software

# DO-178C/ED-12C : Main changes

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Convergence with DO-278 (ground software)
- Merge elements from DO-248 and many CASTs
- Supplements :
  - DO-331 : Model based development and verification
  - DO-332 : Object oriented technologies and related technics
  - DO-333 : Formal methods
- New document : DO-330 Tool Qualification

# Model based development and verification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Use of models as requirements : HLR from System phases and LLR from Design
- Applies to any models related to Software elements (including System phases)
- Can be used for communication or automatization (analysis, code generation)
- Models can be more abstract the Software and partial
- Requires Higher Lever Requirements (HiLR) to assess the models
- Modeling language must be precise and appropriate
  - Specification models : HLR (can be Design models HiLR)
  - Design models : LLR (requires test based on HiLR)

# Formal methods



- A formal method must be correctly defined, justified and appropriate
  - Correctly defined :
    precise, unambiguous, mathematically defined syntax and semantics
  - Justified :
    Sound (never assert a false property)
  - Appropriate :
    All assumptions required for the formal analysis should be described and justified
- Requirement formalization correctness
- Formal analysis can replace :
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis :
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintented data flow
  - Detection of extraneous code (dead or deactivated)
- But : Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

# Formal methods



- A formal method must be correctly defined, justified and appropriate
  - Correctly defined :
    precise, unambiguous, mathematically defined syntax and semantics
  - Justified :
    Sound (never assert a false property)
  - Appropriate :
    All assumptions required for the formal analysis should be described and justified
- Requirement formalization correctness
- Formal analysis can replace :
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis :
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintented data flow
  - Detection of extraneous code (dead or deactivated)
- But : Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

# Formal methods



- A formal method must be correctly defined, justified and appropriate
  - Correctly defined :
    precise, unambiguous, mathematically defined syntax and semantics
  - Justified :
    Sound (never assert a false property)
  - Appropriate :
    All assumptions required for the formal analysis should be described and justified
- Requirement formalization correctness
- Formal analysis can replace :
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis :
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintented data flow
  - Detection of extraneous code (dead or deactivated)
- But : Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

# Formal methods



- A formal method must be correctly defined, justified and appropriate
  - Correctly defined :
    precise, unambiguous, mathematically defined syntax and semantics
  - Justified :
    Sound (never assert a false property)
  - Appropriate :
    All assumptions required for the formal analysis should be described and justified
- Requirement formalization correctness
- Formal analysis can replace :
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis :
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintented data flow
  - Detection of extraneous code (dead or deactivated)
- But : Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

# Formal methods



- A formal method must be correctly defined, justified and appropriate
  - Correctly defined :
    precise, unambiguous, mathematically defined syntax and semantics
  - Justified :
    Sound (never assert a false property)
  - Appropriate :
    All assumptions required for the formal analysis should be described and justified
- Requirement formalization correctness
- Formal analysis can replace :
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis :
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintented data flow
  - Detection of extraneous code (dead or deactivated)
- But : Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

# DO-178B/ED-12B standards : Qualification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Apply to the tools the same rules as the developped system at the same level
- Not really adequate : Additional documents (CAST) were provided
- Tool Operational Requirements (TOR) :
  Tool user point of view (similar to System requirements – HLR)
- Tool Requirements (TR) :
  Tool implementor point of view (LLR)
- Tool kind :
  - Development tools :
    Tools whose output is part of airborne software and thus can introduce errors (same constraints as the developed system).
  - Verification tools :
    Tools that cannot introduce errors, but may fail to detect them (much softer constraints : black box V & V).
  - No proof of error absence category

# DO-178B/ED-12B standards : Qualification

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Apply to the tools the same rules as the developped system at the same level
- Not really adequate : Additional documents (CAST) were provided
- Tool Operational Requirements (TOR) :
  Tool user point of view (similar to System requirements – HLR)
- Tool Requirements (TR) :
  Tool implementor point of view (LLR)
- Tool kind :
  - Development tools :
    Tools whose output is part of airborne software and thus can introduce errors (same constraints as the developed system).
  - Verification tools :
    Tools that cannot introduce errors, but may fail to detect them (much softer constraints : black box V & V).
  - No proof of error absence category

# DO-178C/ED-12C standards : Qualification



- DO-330 additional document :
  Adaptation of core DO-178C/ED-12C to tools
- Tool Qualification Level (1 downto 5) related to DAL
- Tool kind :
  - Criteria 1 : A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2 : A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of :
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A).
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A).
  - Criteria 3 : A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

# DO-178C/ED-12C standards : Qualification

**DO178/ED12 safety standards**

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- DO-330 additional document :
  Adaptation of core DO-178C/ED-12C to tools
- Tool Qualification Level (1 downto 5) related to DAL
- Tool kind :
  - Criteria 1 : A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2 : A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of :
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3 : A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

# DO-178C/ED-12C standards : Qualification

**DO178/ED12 safety standards**

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- DO-330 additional document :
  Adaptation of core DO-178C/ED-12C to tools
- Tool Qualification Level (1 downto 5) related to DAL
- Tool kind :
  - Criteria 1 : A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2 : A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of :
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3 : A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

# DO-178C/ED-12C standards : Qualification

**DO178/ED12 safety standards**

Application to Code generation tools

Application to Static analysis tools

Synthesis : Open questions ?

- DO-330 additional document :
  Adaptation of core DO-178C/ED-12C to tools
- Tool Qualification Level (1 downto 5) related to DAL
- Tool kind :
  - Criteria 1 : A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2 : A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of :
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3 : A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

# DO-178C/ED-12C standards : Qualification



- DO-330 additional document :
  Adaptation of core DO-178C/ED-12C to tools
- Tool Qualification Level (1 downto 5) related to DAL
- Tool kind :
  - Criteria 1 : A tool whose output is part of the resulting software and thus could insert an error (TQL-1 for DAL A).
  - Criteria 2 : A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of :
    - verification process(es) other than that automated by the tool (TQL-4 for DAL A),
    - or development process(es) which could have an impact on the resulting software (TQL-4 for DAL A)
  - Criteria 3 : A tool that, within the scope of its intended use, could fail to detect an error (TQL-5 for DAL A).
  - Still no proof of error absence category (might be TQL-2 for DAL A).

# Some comments

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Standards were designed for systems not tools : Adaptation required
- MCDC not mandatory for tools, but similar arguments might be required
- Traceability of all artefacts in the development, relate requirement (HLR)s, design (LLR) and implementation choices
- Purpose is to provide confidence
- Both cooperative and coercive approach
- Any verification technology can be used, from proofreading to automatic proof if confidence is given
- Choose the strategy and technologies that will best reduce risks

# Some comments



- Standards were designed for systems not tools : Adaptation required
- MCDC not mandatory for tools, but similar arguments might be required
- Traceability of all artefacts in the development, relate requirement (HLR)s, design (LLR) and implementation choices
- Purpose is to provide confidence
- Both cooperative and coercive approach
- Any verification technology can be used, from proofreading to automatic proof if confidence is given
- Choose the strategy and technologies that will best reduce risks

# Some comments II

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Must be applied as soon as possible (cost reduction)
- Small is beautiful (simplicity is the key)
- Certification authorities need to understand the technologies
- Cross-experiments are mandatory (classical w.r.t. alternative methods)

# Some comments II

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Must be applied as soon as possible (cost reduction)
- Small is beautiful (simplicity is the key)
- Certification authorities need to understand the technologies
- Cross-experiments are mandatory (classical w.r.t. alternative methods)

# Transformation verification technologies



- Verification subject :
  - Transformation :
    done once, no verification at use, white box, very high cost
  - Transformation application :
    done at each use, black box, easier, complex error management
- Classical technologies :
  - Document independant proofreading (requirements, specification, implementation)
  - Test
    - Unit, Integration, Functional, Deployment level
    - Requirement based test coverage
    - Source code test coverage
    - Structural coverage, Decision coverage,
      Multiple Condition Decision Coverage (MCDC)

# Transformation verification technologies II

DO178/ED12 safety standards
**Application to Code generation tools**
Application to Static analysis tools
Synthesis : Open questions ?

- Formal technologies (require formal specification) :
  - Automated test generation
  - Model checking (abstraction of the system)
  - Static analysis (abstraction of the language)
  - Automated proof
  - Assisted (human in the loop) proof
- Transformation case
  - Transformation specification : Structural/Behavioral
  - Proof of transformation correctness
  - Links with certification/qualification

# Classical development and verification process

DO178/ED12 safety standards

**Application to Code generation tools**

Application to Static analysis tools

Synthesis : Open questions ?

- Tool development, verification and qualification plans
- Tool Operational Requirements
- Tool Requirements (human proofreading)
- Test plan (requirements based coverage, code coverage verification)
- Implementation and test application

# GeneAuto experiment : Proof assistant based



- Derived from the classical process, early review by french certification bodies
- Formal specification using Coq of tool requirements, implementation and correctness
- Proofreading verification of requirements specification
- Automated verification of specification correctness
- Extraction of OCaML source implementation
- Proofreading verification of extracted OCaML source
- Integration of OCaML implementation with Java/XML implementation (communication through simple text files with regular grammars)
- Proofreading verification of OCaML/Java wrappers (simple regular grammar parsing)
- Test-based verification of user requirements conformance

# Open questions ?

DO178/ED12 safety standards
**Application to Code generation tools**
Application to Static analysis tools
Synthesis : Open questions ?

What are :

- User requirement (TOR) for a transformation/verification ?
- Developer requirement (TR) for a transformation/verification ?
- Formal specification for a transformation/verification ?
- Test coverage for a transformation/verification ?
- Test oracle for a transformation/verification ?
- Qualification constraint for transformation/verification languages ?
- Best strategy for tool verification (once vs at each use) ?

# GeneAuto feedbacks

DO178/ED12 safety standards

**Application to Code generation tools**

Application to Static analysis tools

Synthesis : Open questions ?

- From the certification perspective : Very good but...
  - Still some work on qualification of the proof assistant tools
    - Proof verifier
    - Program extractor
  - Complex management of input/output
- From the developer perspective :
  - High dependence to the technologies
  - Very high cost to use the technology
  - Not easy to subcontract
  - Scalability not ensured
  - Bad separation between semantics-based verification and requirements-based specification
  - Hard to assess development time
- On the use of Java : How to provide confidence in the libraries ?

# Going further : CompCert use experiment

DO178/ED12 safety standards

**Application to Code generation tools**

Application to Static analysis tools

Synthesis : Open questions ?

- CompCert : C to PowerPC optimising code generator developed at INRIA by Xavier Leroy
- PhD thesis at Airbus : Improve certified code efficiency
  - Metrics : WCET, Code and memory size, Cache and memory accesses
  - Improvements of the various phases from models to embedded binary code
  - New verification process using formal methods
  - First CompCert experiments : -12% WCET, -25% code size, -72% cache read, -65% cadre write
  - Design of a CompCert dedicated verification process
  - Feed static analysis results (Astrée, frama-C) from C to binary through CompCert (improve WCET precision)
  - Improve SCADE block scheduling to reduce memory accesses (signal liveness)
  - Design of a whole development cycle verification process with tools qualification

# Going further : CompCert use experiment

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- CompCert : C to PowerPC optimising code generator developed at INRIA by Xavier Leroy
- PhD thesis at Airbus : Improve certified code efficiency
  - Metrics : WCET, Code and memory size, Cache and memory accesses
  - Improvements of the various phases from models to embedded binary code
  - New verification process using formal methods
  - First CompCert experiments : -12% WCET, -25% code size, -72% cache read, -65% cadre write
  - Design of a CompCert dedicated verification process
  - Feed static analysis results (Astrée, frama-C) from C to binary through CompCert (improve WCET precision)
  - Improve SCADE block scheduling to reduce memory accesses (signal liveness)
  - Design of a whole development cycle verification process with tools qualification

# Proposal : Mixed approach

DO178/ED12 safety standards
**Application to Code generation tools**
Application to Static analysis tools
Synthesis : Open questions ?

- Separate specification verification from implementation verification
- Define explicitly semantics traceability link metamodel
- Specify transformation as properties of links
- Implementation verification (mostly syntactic)
  - Implementation must generate both target and links
  - Implementation verification checks properties on generated links links
  - By the way, these links are already mandatory but hand made
- Specification verification : Prove the semantics equivalence between source and target in a trace link
- Ongoing PhDs in project OPEES, P and HiMoCo (industrial cooperation)

# Early feedbacks

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Separation of concerns :
  - Industrial partners : Specification, Implementation, Implementation verification (mainly syntactic)
  - Academic partners : Specification verification (semantics)
- Very good subcontracting capabilities
- Almost no technology constraints on the industrial partner (classical technologies)
- Good scalability
- Rely on already mandatory traceability links (formalized and verified)
- Easy to analyse syntactic error reports
- Enables to modify generated code and links
- Parallel work between syntactic and semantics concerns

# Work in progress



- Positive first experiments on simple use cases from GeneAuto
- But requires some grayboxing (expose parts of the internals)
  - Flattening of statecharts
  - Either very complex specification (doing the flattening)
  - Or express the fixpoint nature of implementation (in the specification)
- Require full scale experiments
- Require exchange with certification authorities
- Require qualified syntactic verification tool (OCL-like, but simpler)
- Require explicit relations between syntactic and semantics work
- Require explicit description of semantics in metamodels

# Static analysis tools

DO178/ED12 safety standards

Application to Code generation tools

**Application to Static analysis tools**

Synthesis : Open questions ?

- Several kind of tools
  - Qualitative and quantitative properties
  - Fixed or user defined properties
  - Semantic abstraction or Proof technologies
- Common aspects : Common pre-qualification
  - Product (source of binary code) reader : fully common ?
  - Configuration (properties, . . .) reader : partly common
  - Result writer and browser : partly common ?
- Split the verification tool in a sequence of elementary activities
  - Common ones (pre-qualification could be shared)
  - Technology specific ones
  - Easier to specify, to validate and to verify
  - Can be physical or virtual (produce intermediate results even in a single tool)

# Required activities

DO178/ED12 safety standards
Application to Code generation tools
**Application to Static analysis tools**
Synthesis : Open questions ?

- Specify user requirements
- Specify tool architecture (elementary tools and their assembly)
- Specify tool level requirements (elementary tools and their assembly)
- Specify functional test cases and results
- Choose verification strategy :
  - Tool verification or Result verification
  - Integration and unit tests (eventually with test generators and oracles)
  - Proof reading of tool source or test results
  - Formal verification of the verification tool itself (i.e. Coq in Coq, Compcert in Coq, . . .)

# Abstraction kind



- Translate to non standard semantics
- Compute recursive equations
- Compute fixpoint of equations
  - Fixpoint algorithm
  - Abstract domains and operators
  - Widening, narrowing
- Check that properties are satisfied on the abstract values
- Produce user friendly feedback (related to product and its standard semantics)

# Deductive kind



- Produce proof obligations (weakest precondition, verification condition, . . .)
- Check the satisfaction of proof obligations
  - Proof term rewriting to simpler language
  - Split to different sub-languages (pure logic, arithmetic, . . .)
  - Apply heuristics to produce a proof term
  - Check the correctness of the proof term
  - Produce failure feedback or proof certificate (related to product and its standard semantics)
- Produce user friendly feedback

# Potential strategy : Common parts

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer :
  - Cross-reading
  - Introduce dual reader/writer : check composition is identity
  - Asymmetric implementation : Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified :
  - Formal tool requirements : foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification : properties of the trace links given as tool requirements
  - Semantic verification : validation of the technology
- User-friendly feedback : Code generation based on trace links

# Potential strategy : Common parts

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer :
  - Cross-reading
  - Introduce dual reader/writer : check composition is identity
  - Asymmetric implementation : Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified :
  - Formal tool requirements : foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification : properties of the trace links given as tool requirements
  - Semantic verification : validation of the technology
- User-friendly feedback : Code generation based on trace links

# Potential strategy : Common parts



- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer :
  - Cross-reading
  - Introduce dual reader/writer : check composition is identity
  - Asymmetric implementation : Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified :
  - Formal tool requirements : foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification : properties of the trace links given as tool requirements
  - Semantic verification : validation of the technology
- User-friendly feedback : Code generation based on trace links

# Potential strategy : Common parts

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Build “semantics”-related trace links during transformations
  Helps in verification of results w.r.t. parameters
- Reader and writer :
  - Cross-reading
  - Introduce dual reader/writer : check composition is identity
  - Asymmetric implementation : Several independent implementations and results comparison
- Code generation and transformation can be formally specified and verified :
  - Formal tool requirements : foreach source construct, what are the generated targets and the links with the source
  - Syntactic verification : properties of the trace links given as tool requirements
  - Semantic verification : validation of the technology
- User-friendly feedback : Code generation based on trace links

# Potential strategy : Abstraction kind



- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification : monotony at the equations-level
  - Semantic verification : soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification : soundness of the abstraction

# Potential strategy : Abstraction kind



- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification : monotony at the equations-level
  - Semantic verification : soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification : soundness of the abstraction

# Potential strategy : Abstraction kind



- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification : monotony at the equations-level
  - Semantic verification : soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification : soundness of the abstraction

# Potential strategy : Abstraction kind

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Non-standard semantics and recursive equation production are similar to code generation
  - Semantic verification : monotony at the equations-level
  - Semantic verification : soundness of the abstraction
- No verification on the fixpoint computation
  - Verification of the result (if least solution is not required)
  - A qualified (much simpler) verification tool is then required
- Verification of the properties of the abstract domains (join, meet, operators, $\alpha \circ \gamma$, widening, narrowing, monotony, ...)
  - Proof reading
  - Automated test generation with oracles
  - Formal specification and proof
- Property checks (based on abstract property generation)
  - Related to code generation
  - Semantic verification : soundness of the abstraction

# Potential strategy : Deductive kind

DO178/ED12 safety standards
Application to Code generation tools
**Application to Static analysis tools**
Synthesis : Open questions ?

- Proof obligation computation is a kind of code generation
  - Semantic verification : correctness of the axiomatic semantics
- Satisfaction of the proof obligations :
  - No verification on proof certificate generation
  - Verification of the certificate itself (much simpler than some heuristic-based automatic prover)
  - Term rewriting can be considered as code generation (endogenous)
  - Curry-Howard type checking can be verified in a similar way
  - Rely on Coq In Coq, Isabelle in Isabelle, . . .

# What about validation of the technologies ?

DO178/ED12 safety standards

Application to Code generation tools

**Application to Static analysis tools**

Synthesis : Open questions ?

- Mainly scientific work and a lot of publications
- Brings confidence but paperwork is not enough
- Mechanized is better but still not enough
- Functional user level tests still mandatory currently
- Mixed system verification experiments (both tests and static analysis)
- Reverse analysis of existing systems

# Synthesis

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

- Technical exchange with certification authorities mandatory
- Cross experiments and reverse engineering experiments mandatory
- Verification strategy must be designed early to choose the right architecture and trace information
- Semi-formal (even formal) requirements must be written as soon as possible

# Synthesis : Open questions ?

DO178/ED12 safety standards
Application to Code generation tools
Application to Static analysis tools
Synthesis : Open questions ?

For model management, verification, transformation, what are :

- User requirement format (TOR) ?
- Developer requirement format (TR) ?
- Formal specification of these requirements ?
- Test coverage based on requirements and structure ?
- Test oracles ?
- Qualification constraint for implementation languages and tools ?
- Best strategy for tool verification (once vs at each use) ?

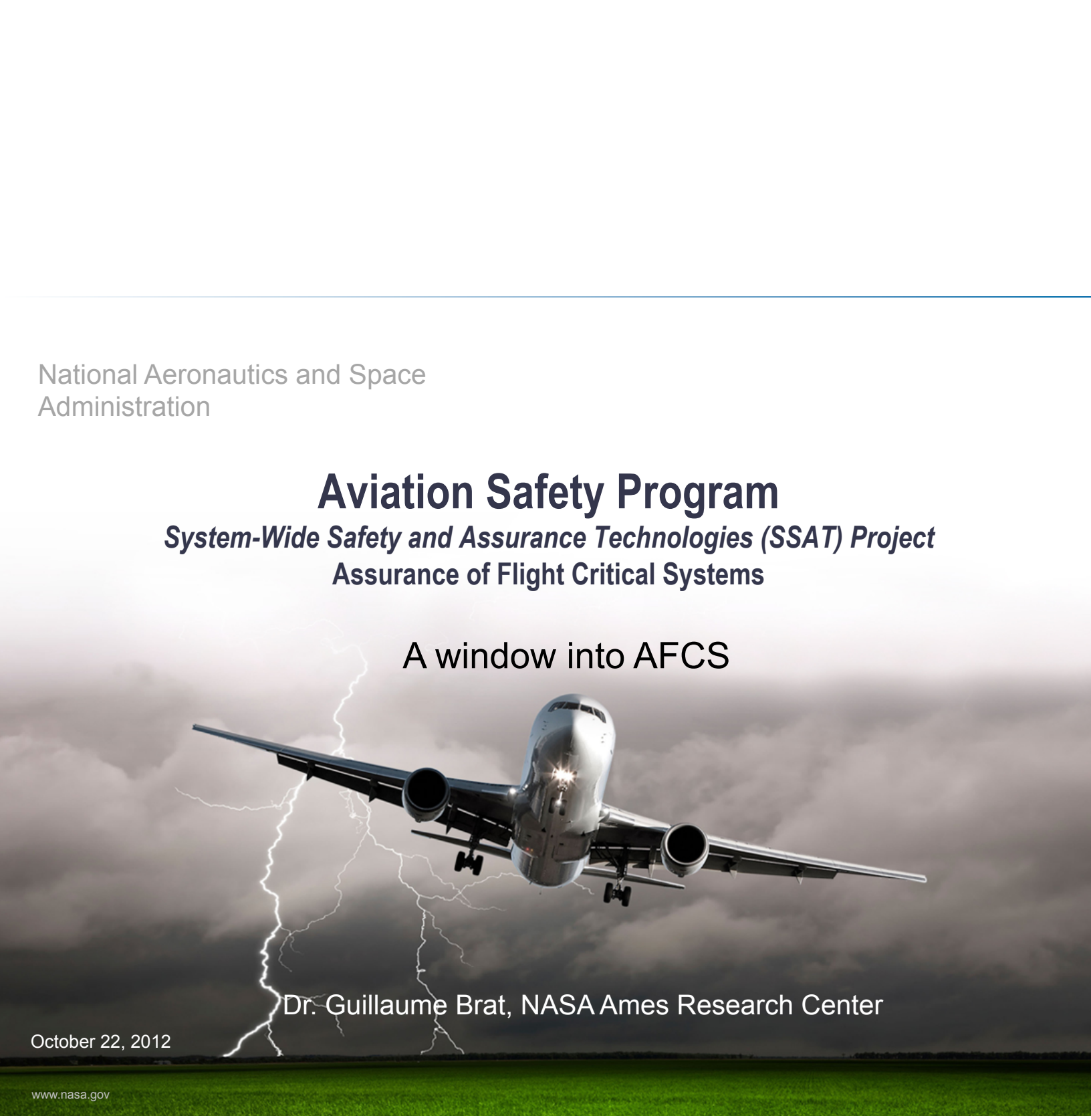
National Aeronautics and Space Administration

# Aviation Safety Program

## *System-Wide Safety and Assurance Technologies (SSAT) Project*

## Assurance of Flight Critical Systems

A window into AFCS



Dr. Guillaume Brat, NASA Ames Research Center

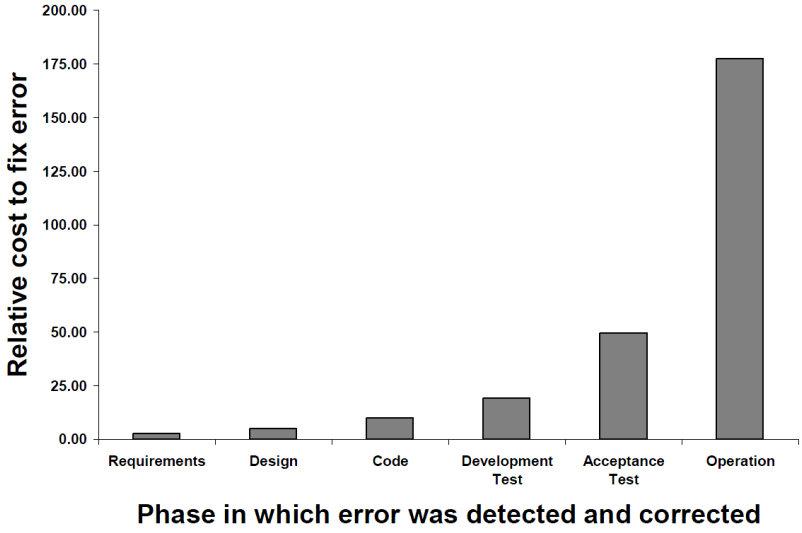
October 22, 2012

# Impact: Cost, and Constraints on Innovation

Size Comparisons of Embedded Software

| System | Lines of Code |
|---|---|
| Mars Reconnaissance Orbiter | 545K |
| Orion Primary Flight Sys. | 1.2M |
| F-22 Raptor | 1.7M |
| Seawolf Submarine Combat System AN/ BSY-2 | 3.6M |
| Boeing 777 | 4M |
| Boeing 787 | 6.5M |
| F-35 Joint Strike Fighter | 5.7M |
| Typical GM car in 2010 | 100M |

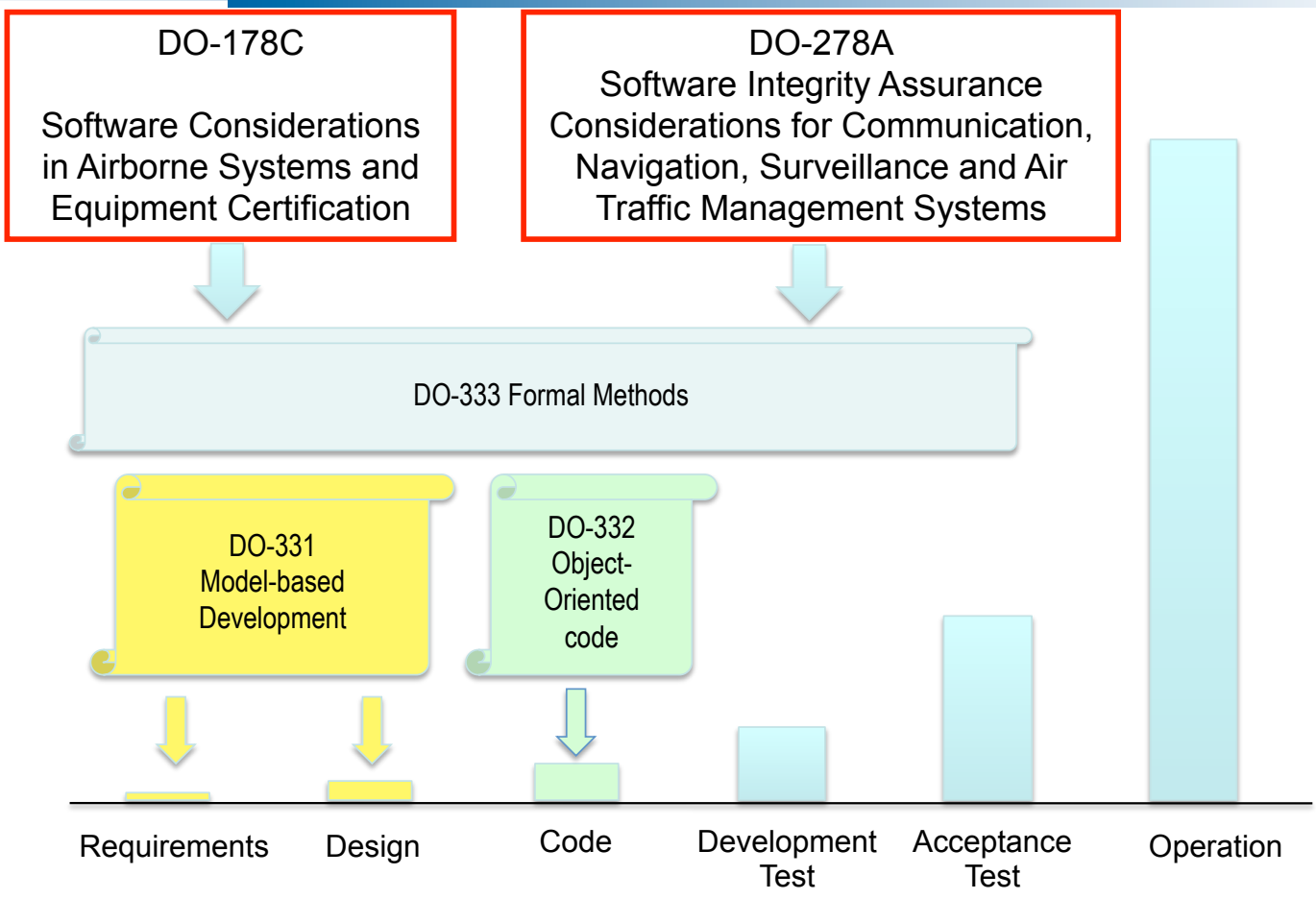
*NASA Study*
Flight Software Complexity, 4/23/2009






Boehm, B. 1981 *Software Engineering Economics*, as cited in DAA, 2008

# Certification Aspect



# IKOS
## Inference Kernel of Open Static analyzers


**Accomplishment**: we have designed and implemented IKOS, a static analysis framework, which allows the custom design of mathematically sound analyzers, e.g., no false negatives, producing less than 10% false positives.

✓ We implemented a generalization of the array-out-of-bound accesses analysis
  ✓New Gauge abstraction domain (CAV 2012)
✓ The framework is being processed to be released under a NOSA license
✓ Initial experiments show that we have achieved less than 10% false positive on embedded system code.

No 3 on the CWE/ SANS Top 25 Most Dangerous Software Errors (MITRE)

| code | Size | Analysis time | Precision |
|---|---|---|---|
| Paparazzi | 35 KLOC | 22s | 99% |
| Gen2 | 22 KLOC | 1mn03s | 98% |
| FLTz | 144 KLOC | 10mn30s | 91% |
| Arduplane | 278 KLOC | 6mn30s | 94% |

# Recent Advances


- The analysis of the OpenSSH code required a sophisticated abstraction based on higher-dimensional convex polyhedra:
  - Combinatorial explosion
  - Brittle abstraction
  - A nightmare to analyze 700 lines of code
- Developed a new abstraction in IKOS: the Gauge Domain
  - Published in CAV 2012
  - Accuracy comparable to convex polyhedra
  - Analyzes 150 KLOC in minutes
  - Scales linearly with the size of the code
  - Commercial analyzer (PolySpace) takes hours on the same code

# Gauges


- In our experience with analyzing large NASA codes, we have observed that most of the time, the value of a scalar variable inside a loop nest was entirely determined by the control structure in terms of symbolic bounds of the form $a_0 + a_1\lambda_1 + \cdots + a_k\lambda_k$, where $\lambda_1, \ldots, \lambda_k$ denote loop counters and $a_1, \ldots, a_k$ are integer coefficients.

```
p = &msg;
for (i = 0; i < n; i++) {
  if(*p == ...) {
    ...
    p += 16;
  } else {
    ...
    p += 32;
  }
}
```

Convex polyhedra:

$$\begin{cases} 0 \leq \mathtt{i} \leq \mathtt{n} - 1 \\ 16\mathtt{i} \leq \mathtt{p} \leq 32\mathtt{i} \end{cases}$$

Gauges:

$$\begin{cases} \lambda \leq \mathtt{i} \leq \lambda \\ 16\lambda \leq \mathtt{p} \leq 32\lambda \end{cases}$$

Additional properties:

$$\begin{cases} \lambda \leq \mathtt{n} - 1 \\ \lambda \in [0, +\infty] \end{cases}$$

**Fig. 1.** Loop invariant expressed with convex polyhedra and gauges

# Analysis of Floating-Point Computations

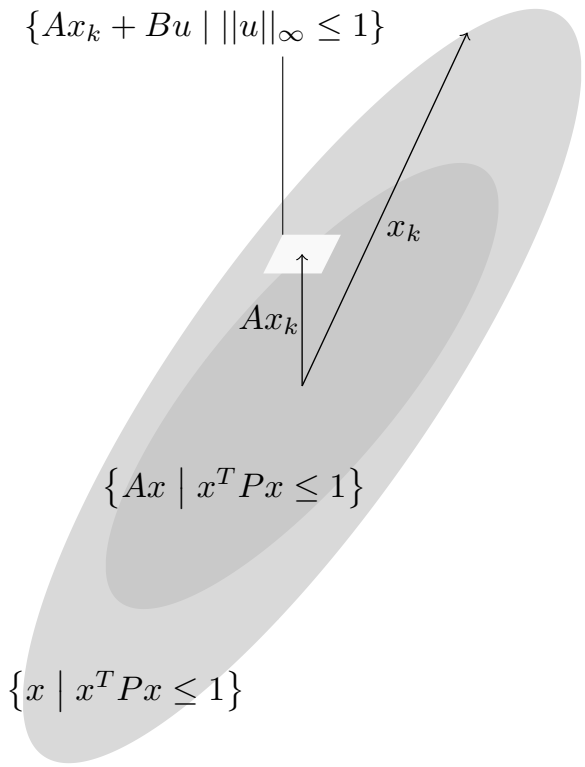

- Challenging problem
- Solutions exist for a very small class of codes
  - ASTREE analyzer developed in France for Airbus
  - In practice only works for Airbus code
- Ongoing development of abstractions for floating-point computations in IKOS
  - Broader class of codes (UAVs, ground control)
  - Performance is the key issue

# Analysis of Autocoded Systems


- Model-based development:
  - Specify a controller in a mathematical modeling environment (MATLAB/Simulink)
  - Do all the verification at the model level
  - Automatically generate code from the model
- Question:
  - Does the generated code verify the same properties established at the model level?
  - Use static analysis to answer this question automatically

# Example: Stability of Control Systems

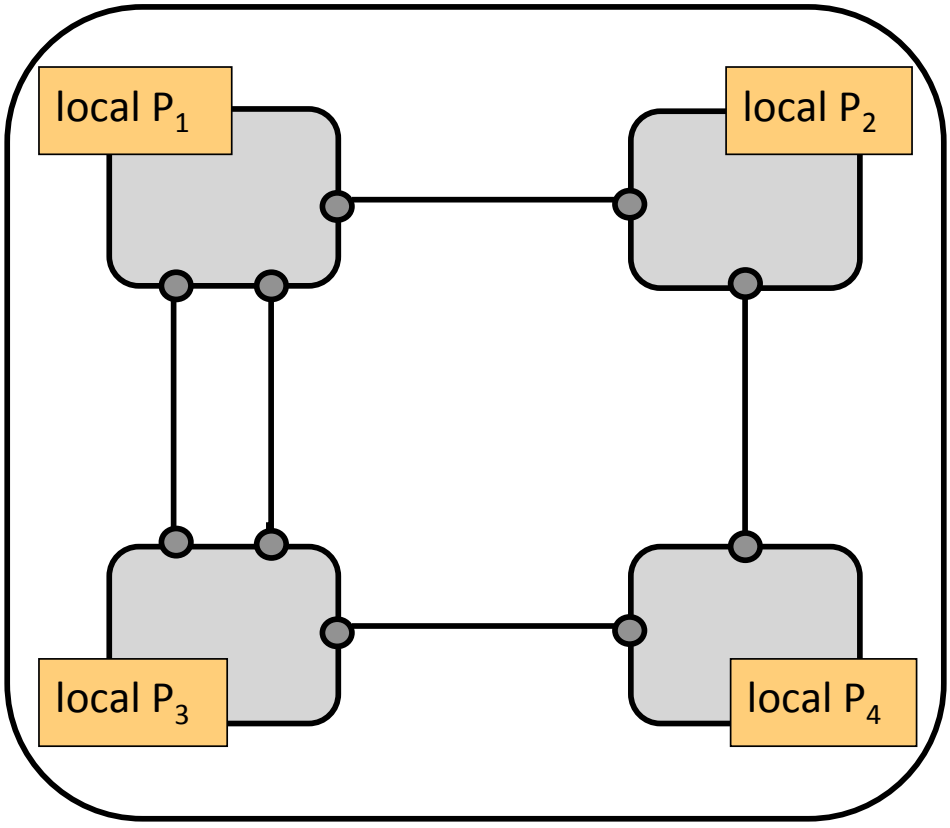
- Lyapunov theory
- Well understood at the model level: find an ellipsoidal invariant
- Does this hold for the generated code?
  - Discretization
  - Floating-point arithmetic
- Development of a suitable domain of ellipsoids for static analysis



# IKOS: Applications

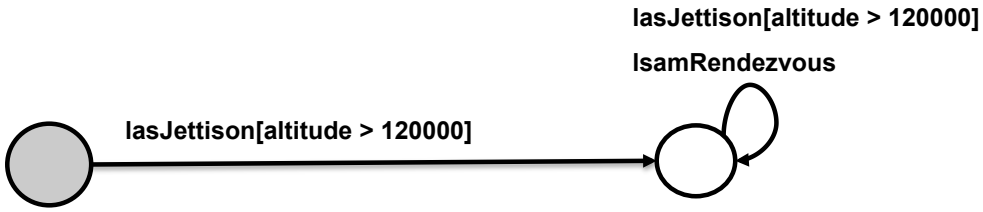

- Applied to UAV autopilots (40 to 250 KLOC)
  - Few inconclusive reports (< 2%)
- Running on LADEE (Lunar Atmosphere and Dust Environment Explorer) flight software
- Ongoing:
  - Scientific computation code (Corey Ippolito)
  - Image processing code for GOES-R (Geostationary Operational Environmental Satellite R-Series)

# compositional verification




- performed at design time
- local properties guarantee P
- decomposition can be performed manually (e.g., for system architectures)
- we provide automated techniques based on learning component interfaces
- individual components can be checked against local properties using model checking or testing

# what local properties express (CEV)



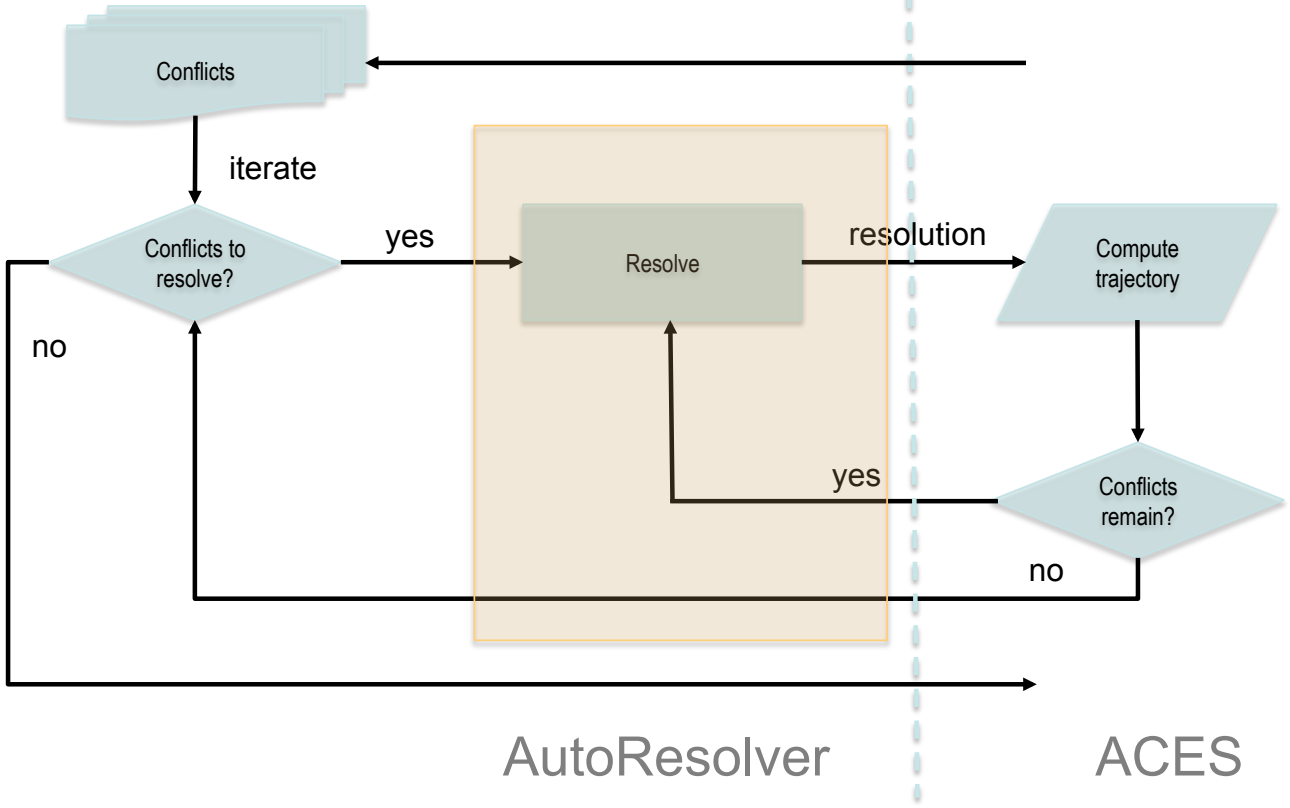

- constraints on inputs for correct operation
- constraints on sequences of method / service invocations

# verification of separation assurance systems


• AutoResolver is a NextGen air traffic management system developed by Prof. Heinz Erzberger and his team in Code A

• it is high in complexity (non-linear math and heuristics) with over 50k lines of Java; it has a complex interaction mechanism via callbacks into the Airspace Concept Evaluation System or ACES, a simulator that captures the key feedback response mechanism of the National Airspace System (NAS)

# AutoResolver



## our efforts


- abstract ACES –trajectory computation
- generate conflicts
- work with Code A to identify properties:
  - *"If the resolution produced for a conflict results in a secondary conflict, then the time to loss of separation for the secondary conflict should be greater than the time to loss of separation for the original conflict"*
- developing advanced automated testing techniques that ensure coverage of all the paths of the AutoResolver
- working on developing and connecting all the parts of the compositional verification picture

# Conclusions


- Stable research program under ARMD for developing safety assurance techniques for flight-critical systems
- The main focus is on formal methods
  - Abstract interpretation
  - Compositional verification
  - Advanced test generation techniques
- Collaboration with
  - NASA Langley
  - Industry
  - International partners (ONERA, IRIT, may be VERIMAG)

# Formal Methods for Aerospace Applications: A control engineer's perspective

Eric Feron
Dutton/Ducoffe Professor of Aerospace Engineering
Georgia Institute of Technology

feron@gatech.edu

# Take-Home Message

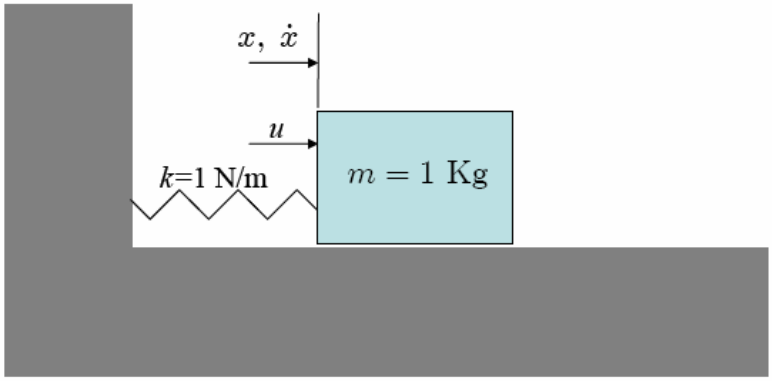
Safety-critical embedded software design best tackled through proper specification, followed by automatic coding of specs AND their semantics

# Analyze and Design Early

Most errors arise during specification of software, not coding.

- Allow the engineer to specify, analyze, then auto-code.
- SCADE/Esterel Technologies, Picture2code/Pratt & Whitney, Realtime Workshop/Mathworks, Gene-auto/ENSEEIHT, Gryphon/Rockwell-Collins.

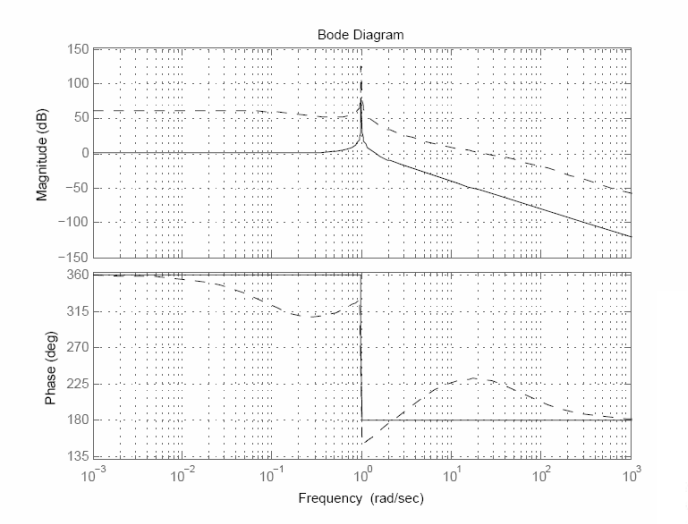
# A simple control example




$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0$$

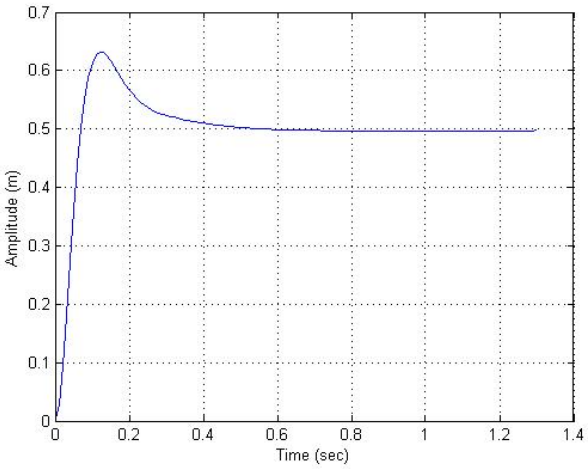
$$y = [1 \ \ 0]\begin{bmatrix} x \\ \dot{x} \end{bmatrix}.$$

# A simple control example




$$\tilde{y}(t) = \mathbf{SAT}(y(t)),$$
$$u(s) = 128\frac{s+1}{s+0.1}\frac{s/5+1}{s/50+1}\tilde{y}(s),$$

Step response

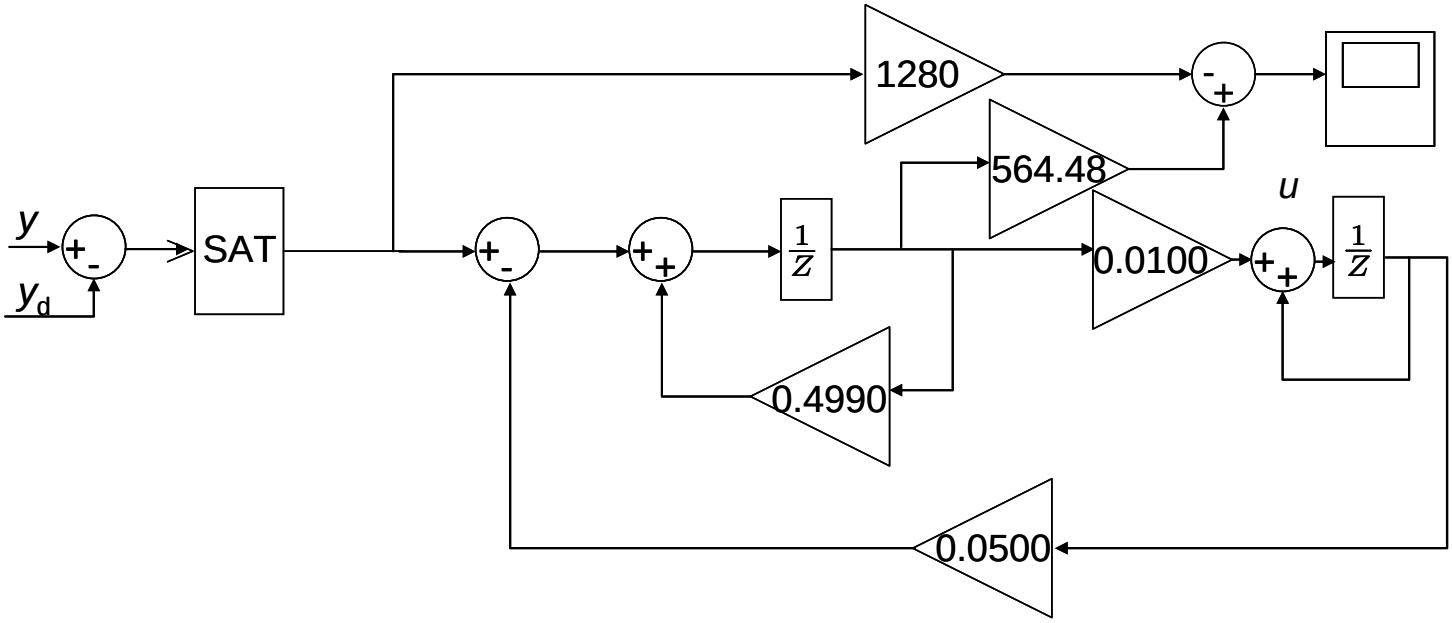


# Controller implementation

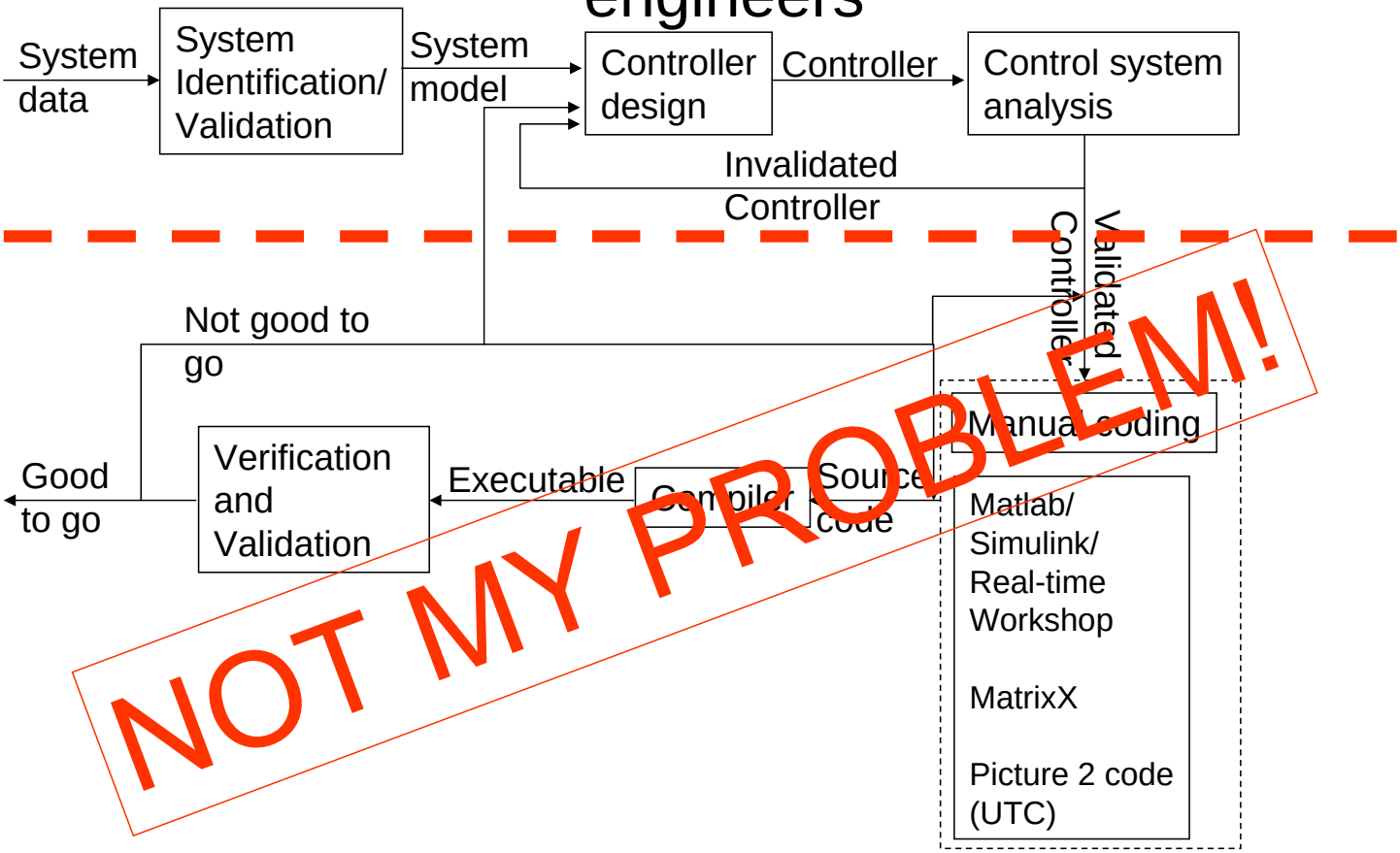
$$x_{c,k+1} = \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k)$$

$$u_k = -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_{c,k} + 1280\, \mathbf{SAT}(y_k)$$

Discrete time
Implementation
100Hz



# Control system design as seen by control engineers



# Code-level analyses of control software

- Most significant contribution is from Patrick Cousot's research group at Ecole Normale Superieure, Paris.
- Abstract interpretation aims at capturing semantics of programs
- Most important application is ASTREE analyzer for Airbus A380 control code.
- From Feret, "Static Analysis of Digital Filters", 2004 (also with ASTREE).



A simplified second order filter relates an input stream $E_n$ to an output stream defined by:

$$S_{n+2} = aS_{n+1} + bS_n + E_{n+2}.$$

Thus we experimentally observe, in Fig. 4, that starting with $S_0 = S_1 = 0$ and provided that the input stream is bounded, the pair $(S_{n+2}, S_{n+1})$ lies in an ellipsoid. Moreover, this ellipsoid is attractive, which means that an orbit starting out of this ellipsoid, will get closer of it. This behavior is explained by Thm. 5.

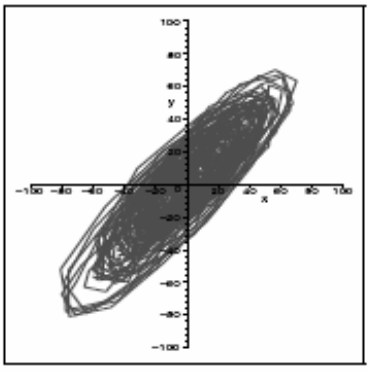
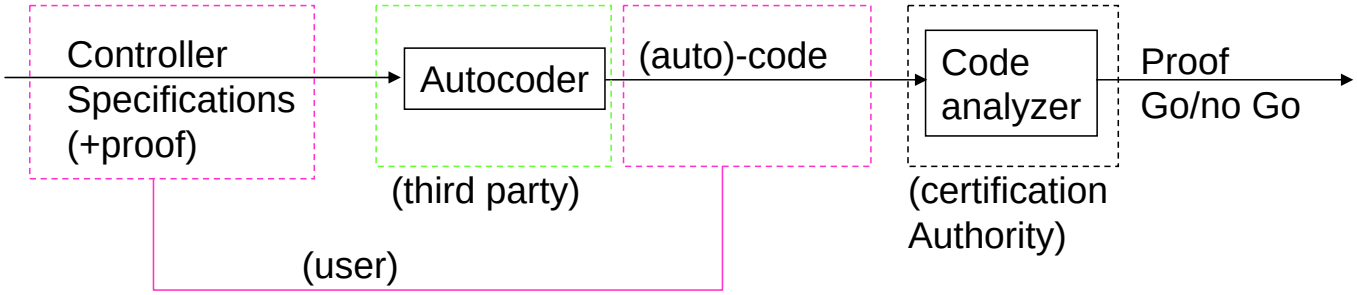


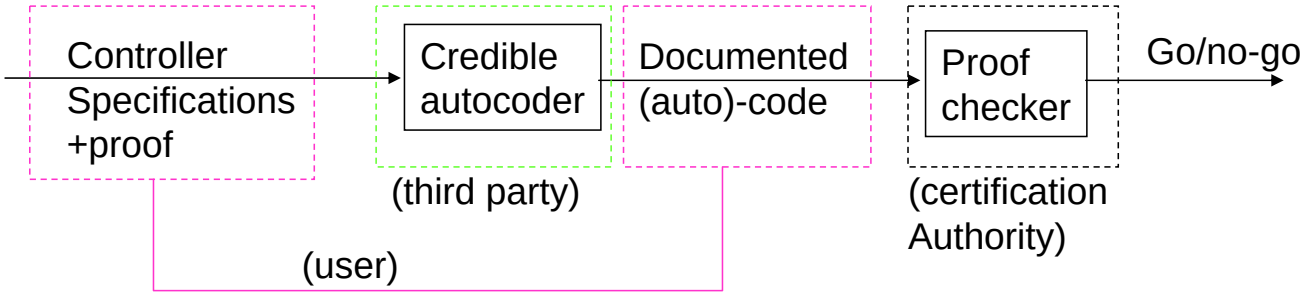
**Fig. 4.** Orbit.

# A Paradigm Shift Enabled by Good Specification Analyses

(auto) Code analyzer



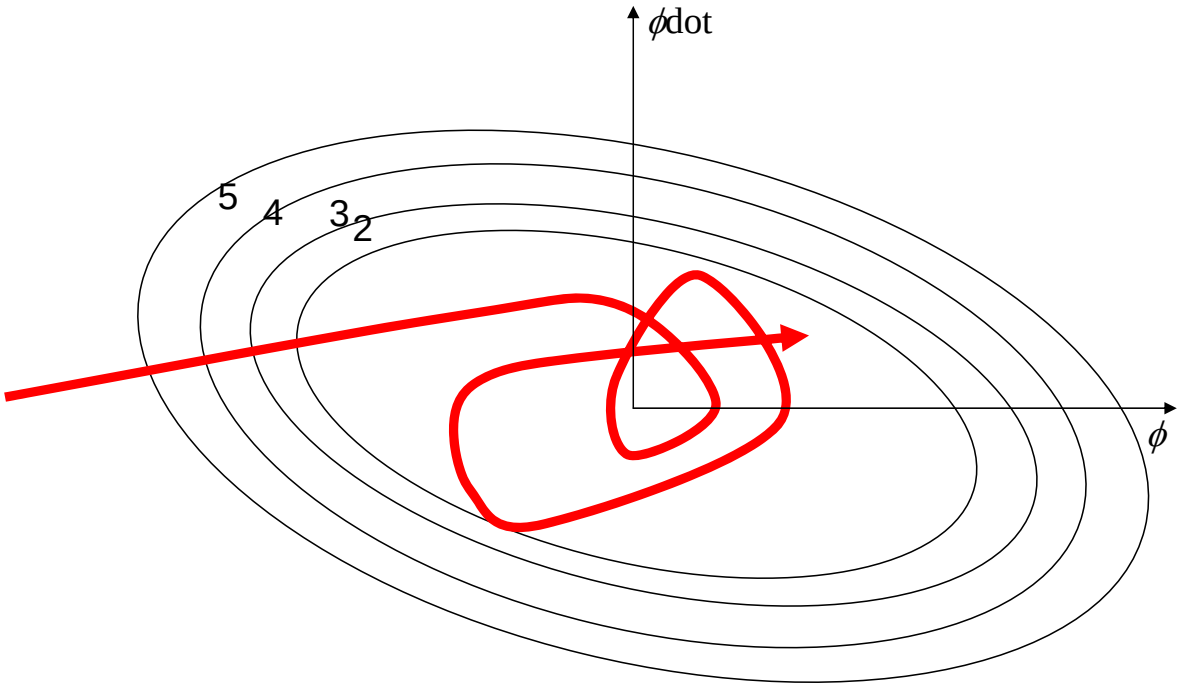

Credible autocoder (a la Rinard)



# Desirable attributes of “system proofs”

- Must be expressive enough to tell nontrivial statements about system
- Must speak the language of system representation, eg: “IEEE Transactions on Automatic Control proofs” written in natural language (one wonders…), “Simulink proofs” expressed in Simulink, “Program proofs” expressed in formal languages.
- Must be “elementary enough” to be easily checked wherever necessary.

# Lyapunov functions and invariant ellipses



# Back to the Example

The control-systemic way:

$$\begin{aligned} x_{c,k+1} &= \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k) \\ u_k &= -\begin{bmatrix} 564.48 & 0 \end{bmatrix} x_{c,k} + 1280\, \mathbf{SAT}(y_k) \end{aligned}$$

Assume the controller state is initialized at $x_{c,0} = 0$

What range of values could be reached by the state $x_{c,k}$ and the control variable $u_k$?

There is a variety of options, including computation of -1 norms.

A Lyapunov-like proof (from Boyd *et al.*, Poola*):*

The ellipsoid $\mathcal{E}_P = \left\{ x \in \mathbf{R}^2 \mid x^T P x \leq 1 \right\}.$ $P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}.$

is invariant. None of the entries of *x* exceeds 7 in size.

# A proof for control people

$\forall t,\ x^T P x \leq 1$ is equivalent to $x_k^T P x_k \leq 1 \Rightarrow x_{k+1}^T P x_{k+1} \leq 1$

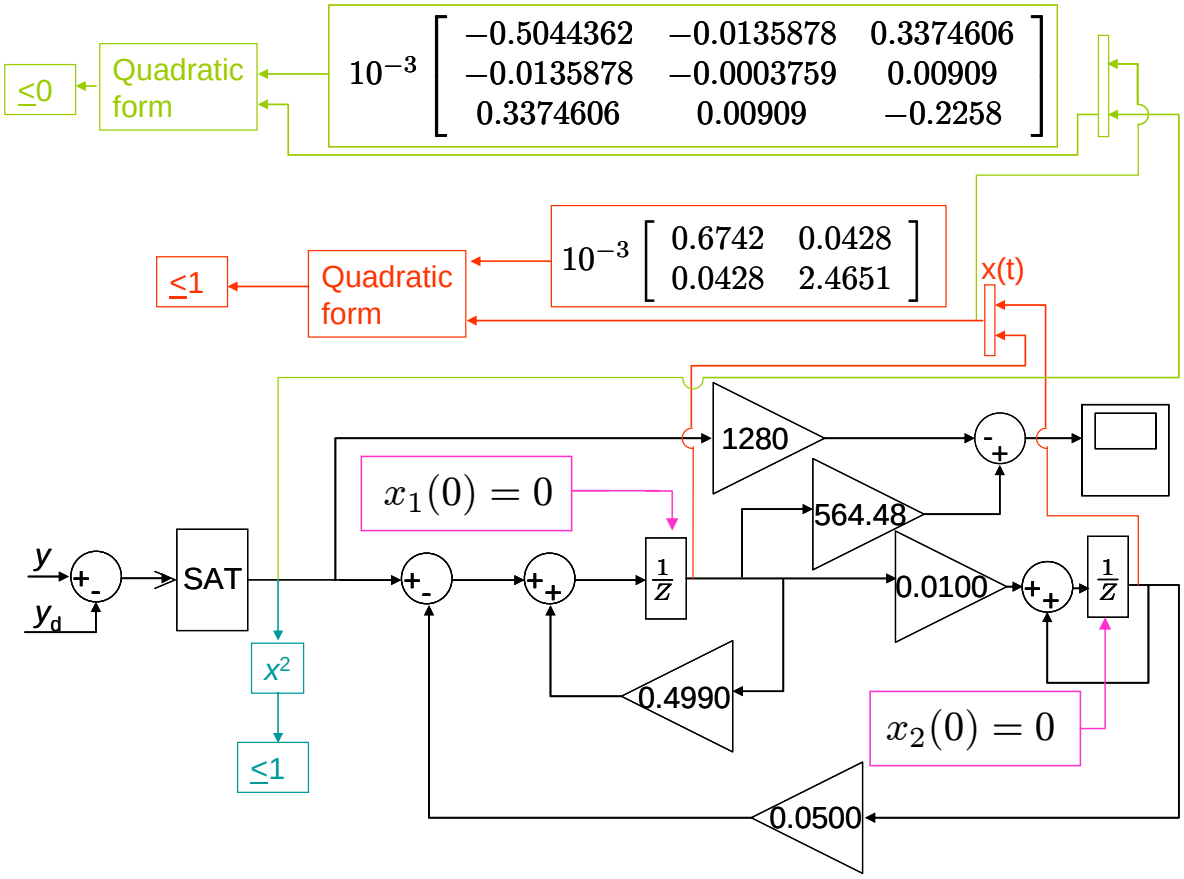
Or $(Ax + Bw)^T P (Ax + Bw) \leq 1$ whenever $x^T P x \leq 1$ and $w^2 \leq 1$

True if there exists $\mu$ such that $(Ax+Bw))^T P(Ax+Bw) - \mu x^T P x - (1-\mu)w^2 < 0$, (*) a tautology.

Indeed a linear combination of (*) and $x^T P x \leq 1$ and $w^2 \leq 1$ yields the desired property.

$P$ that works is $P = 10^{-3} \begin{bmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{bmatrix}$, with $\mu = 0.9991$ and tautology (*) is $10^{-3} \begin{bmatrix} x \\ w \end{bmatrix}^T \begin{bmatrix} -0.5044362 & -0.0135878 & 0.3374606 \\ -0.0135878 & -0.0003759 & 0.00909 \\ 0.3374606 & 0.00909 & -0.2258 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \leq 0.$

# Simulink, Discrete Time Formal Semantics



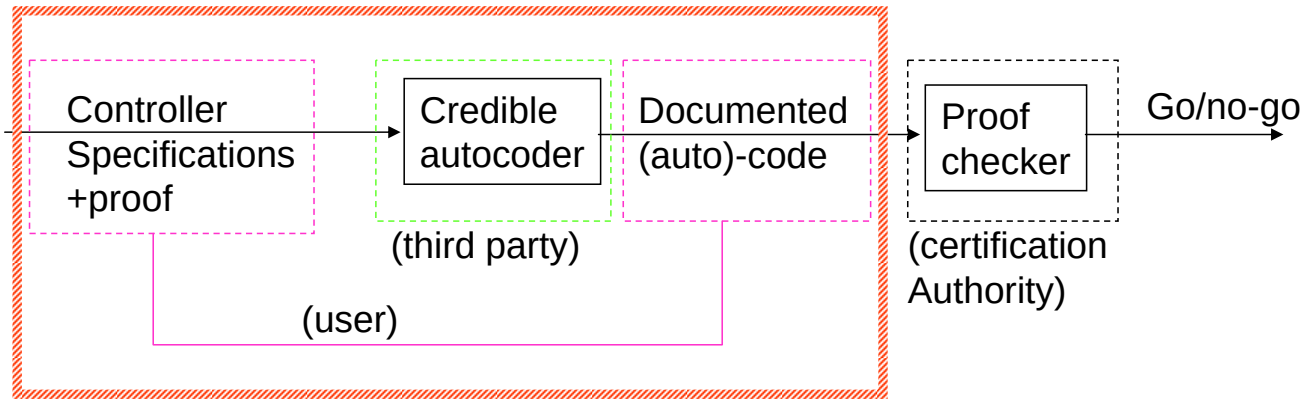
# Commented code

$\{\text{true}\}$
```
1:  A = [0.4990, -0.0500; 0.0100, 1.0000];
```
$\{\text{true}\}$
```
2:  C = [-564.48, 0];
```
$\{\text{true}\}$
```
3:  B = [1;0];D=1280
```
$\{\text{true}\}$
```
4:  x = zeros(2,1);
```
$\{x \in \mathcal{E}_P\}$
```
5:  while 1
```
$\{x \in \mathcal{E}_P\}$
```
6:  y = fscanf(stdin,"%f")
```
$\{x \in \mathcal{E}_P\}$
```
7:  y = max(min(y,1),-1);
```
$\{x \in \mathcal{E}_P,\ y^2 \leq 1\}$
```
8:  u = C*x+D*y;
```
$\{x \in \mathcal{E}_P,\ u^2 \leq 2(CP^{-1}C^T + D^2),\ y^2 \leq 1\}$
```
9:  fprintf(stdout,"%f\n",u)
```
$\{x \in \mathcal{E}_P,\ y^2 \leq 1,\ (Ax+By)^T P(Ax+By) - 0.01x^T Px - 0.99y^2 \leq 0\}$

*skip*

$\{Ax + By \in \mathcal{E}_P,\ y^2 \leq 1\}$
```
10:  x = A*x + B*y;
```
$\{x \in \mathcal{E}_P\}$
```
11:  end
```

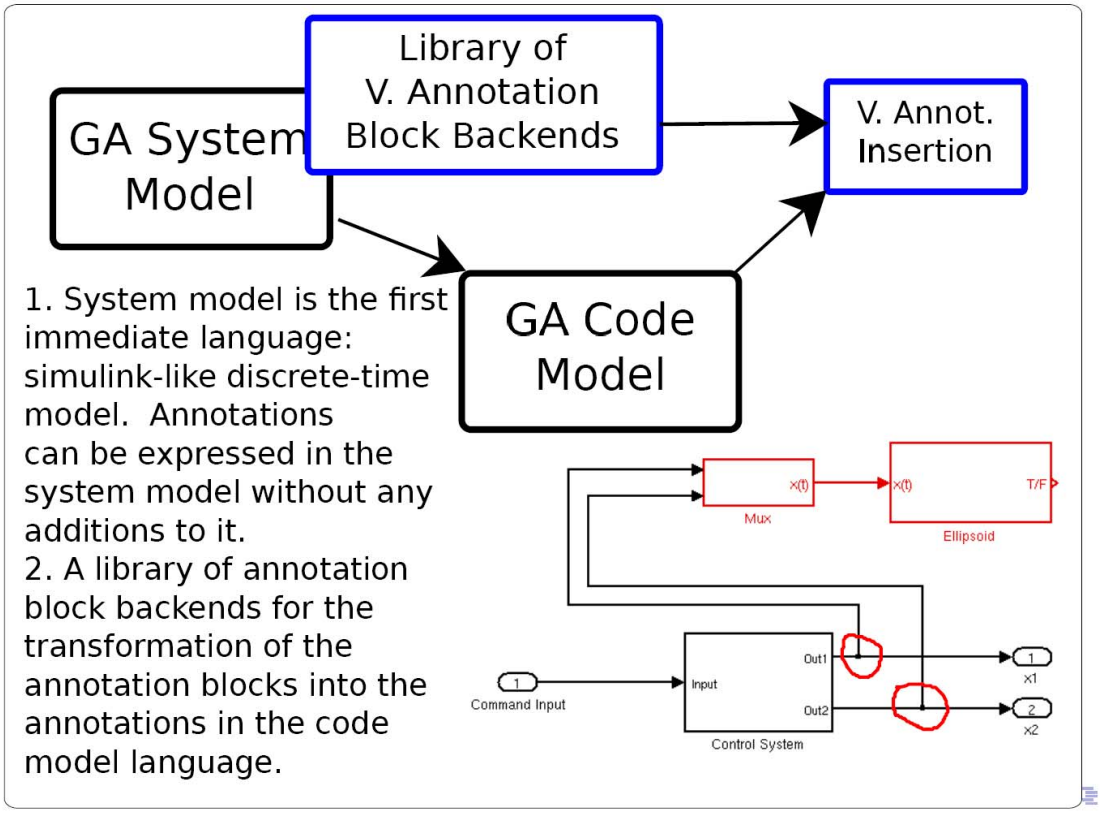
# Front End: Formal comment writing



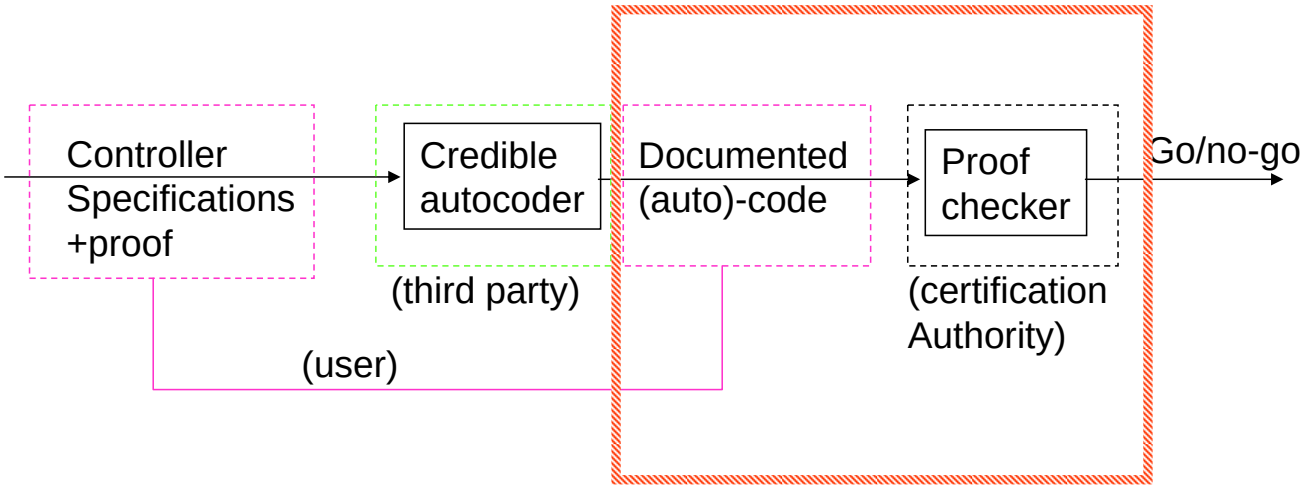

- ANSI/ISO C Specification Language (ACSL) can be used to formally comment C programs and can be handled by Frama-C.
- Start from Simulink
- End with commented C code

# A prototype front-end built on Gene-Auto

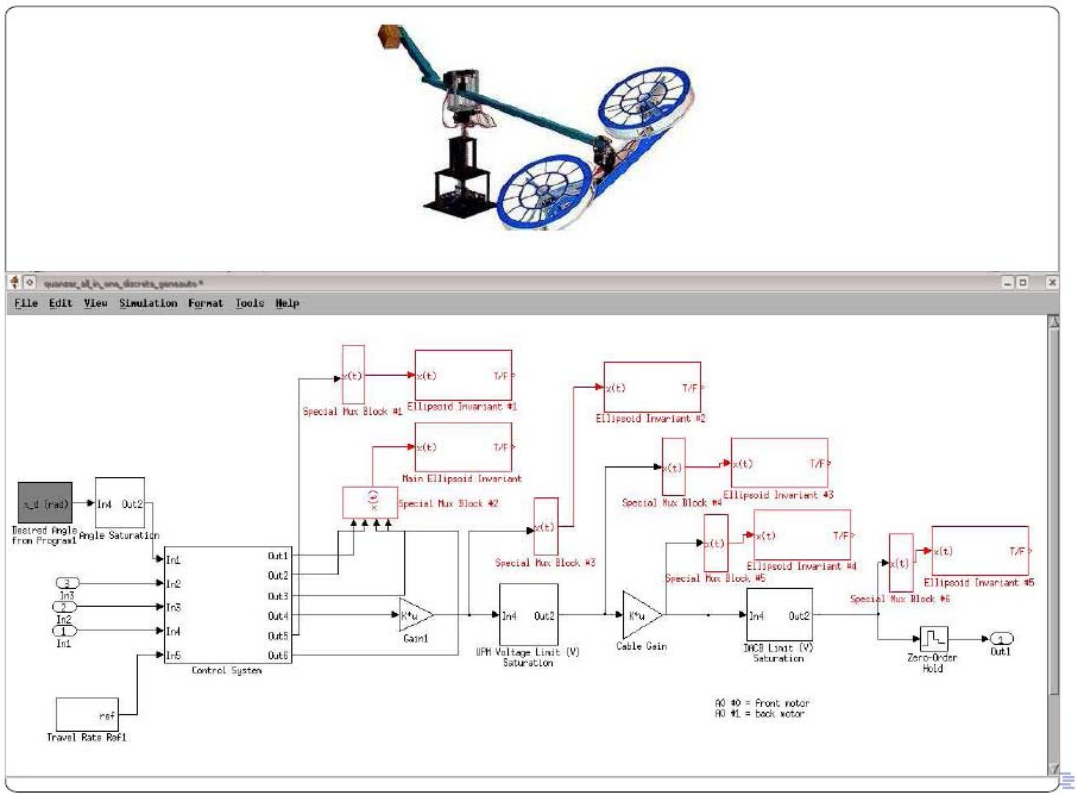
- Thank you Marc Pantel, Arnaud Dieumegard, Andres Toom



# Back End: Verification of Code Semantics

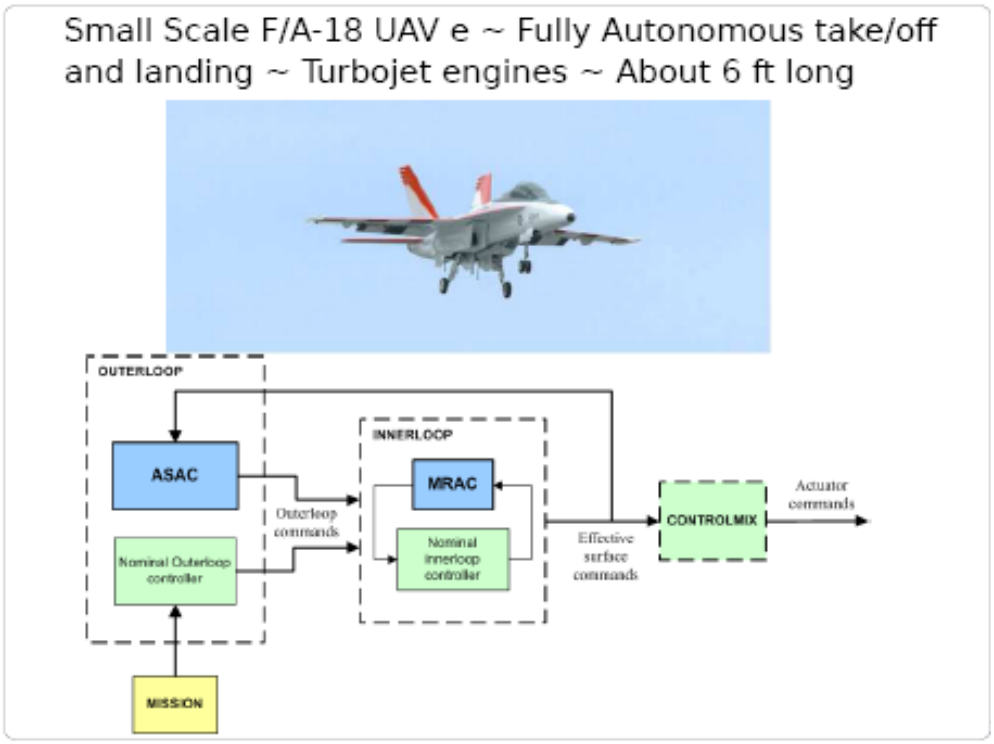


# A physical example: 3 DOF helicopter



# And it still works!!!

# F-18 replica from Rockwell-Collins




http://www.youtube.com/watch?v=QJkIONTzbNM

# Complexity Increase vs Quanser: Sets of Gains

## Quanser

- Quanser: operates in only one flight condition hence only has one set of gains.

## F/A-18 UAV

- F/A-18: operates in a range of flight conditions hence has an infinite set of gains.
- Offline: gains are picked a priori at some finite set of points in the range of flight conditions (altitude, speed).
- Online: gains during runtime are computed by interpolation on the pre-defined offline gains.
- We call this gain-scheduling. The F/A-18 controllers are gain-scheduled at 110 different points.





# # Of Stability Analysis

## Stability Analysis

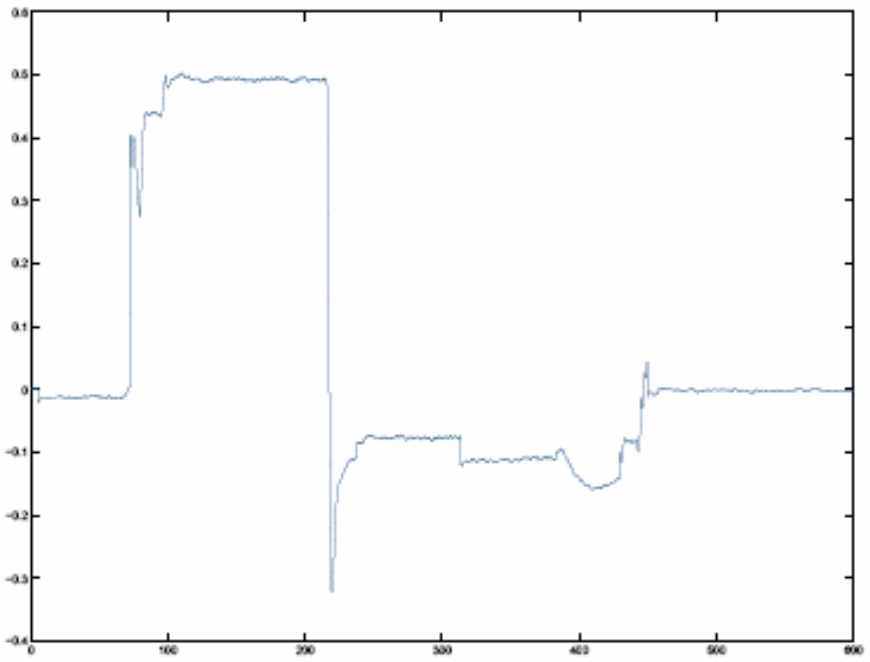
- Need to generate a stability proof for each controller mode (33) for each set of gains (110).
- Total number of configurations is 3630.





# Pitch Controller (Longitudinal)

## Pitch Dynamics



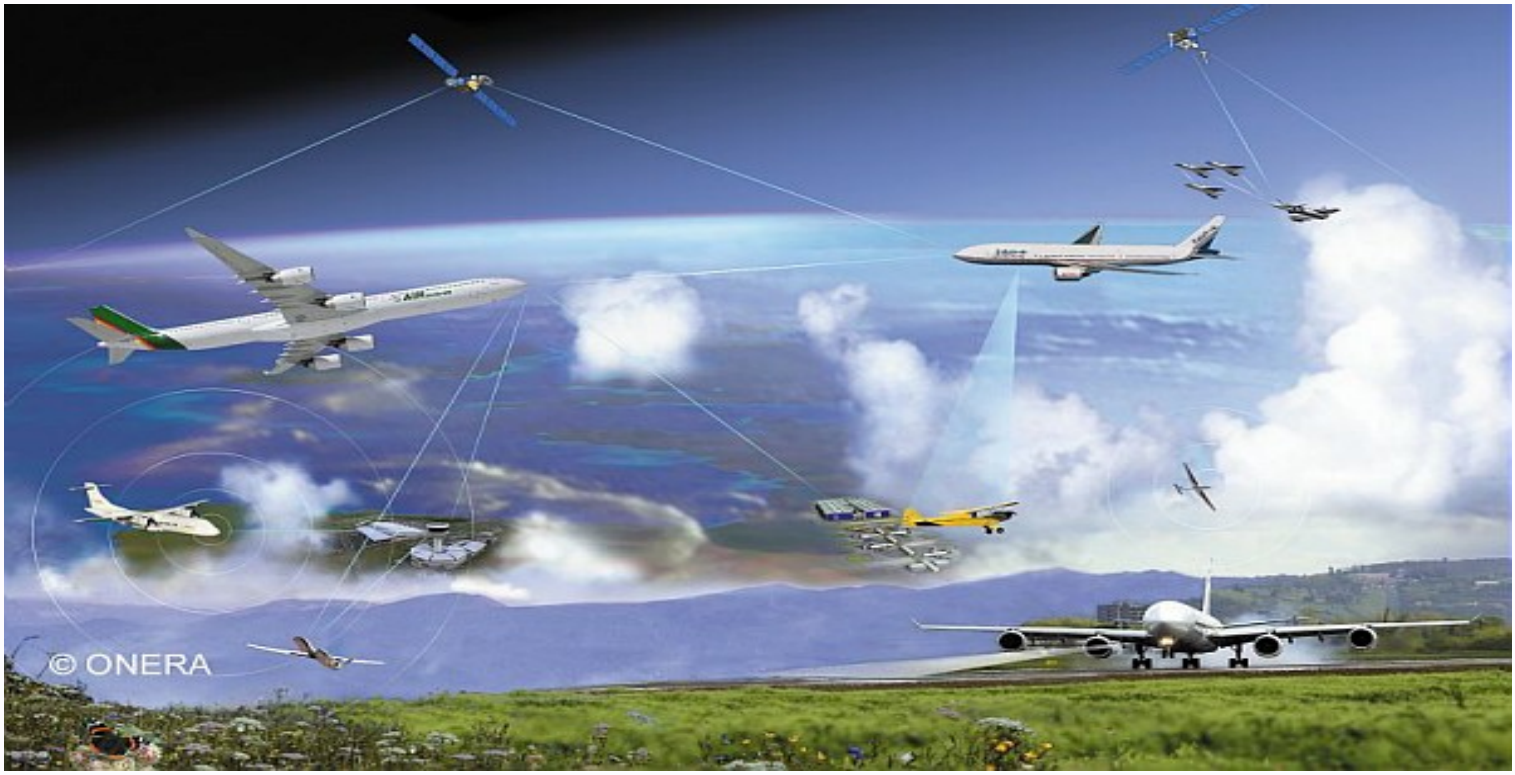
# Application to Collision avoidance TCAS / last resort safety net

# Conclusion

- It is possible to generate safety-critical control code from specifications, all-equipped with semantics and proofs.
- With that, code-level analyses are possible, and much easier than analyses from code alone.

# Acknowledgements


- Army Research Office
- Dutton/Ducoffe professorship at Georgia Tech
- Fondation STAE Toulouse
- Ecole Nationale de l'Aviation Civile
- Institut National Polytechnique de Toulouse
- National Science Foundation
- NASA
- ONERA DCSD and DTIM
- *Fernando Alegre, Arnaud Dieumegard, Alwyn Goodloe, Heber Herencia, Pierre-Loic Garoche, Romain Jobredeaux, Sam Owre,* ***Marc Pantel****, Pierre Roux, Andres Toom, Arnaud Venet, Tim Wang.*




# Analyzing control command software: the need for non linear invariants

*Formal methods for Aerospace Applications - FMCAD'12 Tutorial*

Pierre-Loïc Garoche – Onera

Mon October 22nd 2012

# Content

- **Need for non linear invariants**
- **Proving stability at code level**
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- **Conclusion**

# Content

- **Need for non linear invariants**
- **Proving stability at code level**
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- **Conclusion**

# NON LINEAR INVARIANTS IN CONTROL COMMAND SOFTWARE

Properties of controllers:

- open-loop stability
- close-loop stability
- tracking
- ...

(Most | All) of them can be expressed as invariants over the system's variables.

$\Longrightarrow$ Lyapunov functions

## OPEN-LOOP STABILITY

A controller is open-stable

$\triangleq$ its output stay bounded for any bounded input.

Example: $in_0 \in [-1, 1], in_1 \in [-1, 1]$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.6227 & 0.3871 & 0.0102 & 0.3064 \\ -0.3407 & 0.9103 & -0.3388 & 0.0649 \\ 0.0918 & -0.0265 & -0.7319 & 0.2669 \\ 0.2643 & -0.1298 & -0.9903 & 0.3331 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0.3064 & 0.1826 \\ -0.0054 & 0.6731 \\ +0.0494 & 1.6138 \\ -0.0531 & 0.4012 \end{pmatrix} \begin{pmatrix} in_0 \\ in_1 \end{pmatrix}$$

Proof?

## OPEN-LOOP STABILITY

A controller is open-stable

$\triangleq$ its output stay bounded for any bounded input.

Example: $in_0 \in [-1, 1], in_1 \in [-1, 1]$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.6227 & 0.3871 & 0.0102 & 0.3064 \\ -0.3407 & 0.9103 & -0.3388 & 0.0649 \\ 0.0918 & -0.0265 & -0.7319 & 0.2669 \\ 0.2643 & -0.1298 & -0.9903 & 0.3331 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0.3064 & 0.1826 \\ -0.0054 & 0.6731 \\ +0.0494 & 1.6138 \\ -0.0531 & 0.4012 \end{pmatrix} \begin{pmatrix} in_0 \\ in_1 \end{pmatrix}$$

Proof?

- a Lyapunov function exists ! (Which one ? Existential proof)

## OPEN-LOOP STABILITY

A controller is open-stable

$\triangleq$ its output stay bounded for any bounded input.

Example: $in_0 \in [-1, 1], in_1 \in [-1, 1]$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.6227 & 0.3871 & 0.0102 & 0.3064 \\ -0.3407 & 0.9103 & -0.3388 & 0.0649 \\ 0.0918 & -0.0265 & -0.7319 & 0.2669 \\ 0.2643 & -0.1298 & -0.9903 & 0.3331 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0.3064 & 0.1826 \\ -0.0054 & 0.6731 \\ +0.0494 & 1.6138 \\ -0.0531 & 0.4012 \end{pmatrix} \begin{pmatrix} in_0 \\ in_1 \end{pmatrix}$$

Proof?

- a Lyapunov function exists ! (Which one ? Existential proof)
- $0.14 \times x_3^2 - 0.22 \times x_3 \times x2 + 0.07 \times x_3 \times x_1 - 0.03 \times x_3 \times x_0 + 0.13 \times x_2^2 - 0.08 \times x_2 \times x_1 + 0.02 \times x_2 \times x_0 + 0.06 \times x_1^2 - 0.04 \times x_1 \times x_0 + 0.05 \times x_0^2$ is a Lyapunov function (Constructive proof)

## OPEN-LOOP STABILITY

A controller is open-stable

$\triangleq$ its output stay bounded for any bounded input.

Example: $in_0 \in [-1, 1], in_1 \in [-1, 1]$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.6227 & 0.3871 & 0.0102 & 0.3064 \\ -0.3407 & 0.9103 & -0.3388 & 0.0649 \\ 0.0918 & -0.0265 & -0.7319 & 0.2669 \\ 0.2643 & -0.1298 & -0.9903 & 0.3331 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0.3064 & 0.1826 \\ -0.0054 & 0.6731 \\ +0.0494 & 1.6138 \\ -0.0531 & 0.4012 \end{pmatrix} \begin{pmatrix} in_0 \\ in_1 \end{pmatrix}$$

Proof?

- a Lyapunov function exists ! (Which one ? Existential proof)
- $0.14 \times x_3^2 - 0.22 \times x_3 \times x2 + 0.07 \times x_3 \times x_1 - 0.03 \times x_3 \times x_0 + 0.13 \times x_2^2 - 0.08 \times x_2 \times x_1 + 0.02 \times x_2 \times x_0 + 0.06 \times x_1^2 - 0.04 \times x_1 \times x_0 + 0.05 \times x_0^2$ is a Lyapunov function (Constructive proof)

**Theorem**

Any stable linear controller admits a quadratic Lyapunov function

## Objective: reasoning about non linear properties

**Analyzing controllers**

1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

## Objective: reasoning about non linear properties

**Analyzing controllers**

1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.

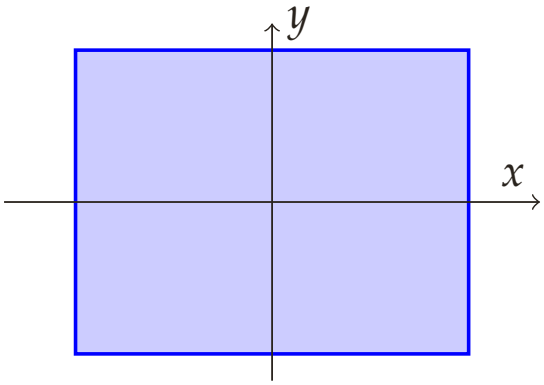
## OBJECTIVE: REASONING ABOUT NON LINEAR PROPERTIES

**Analyzing controllers**

1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.



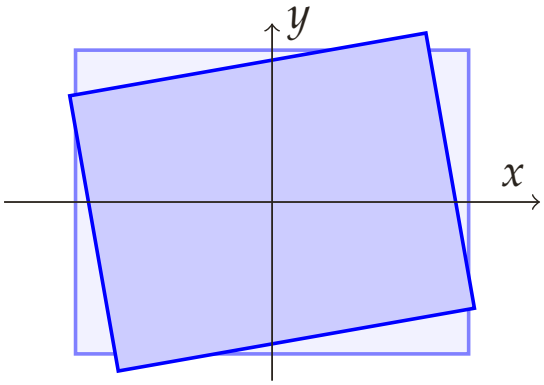
## OBJECTIVE: REASONING ABOUT NON LINEAR PROPERTIES

**Analyzing controllers**

1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.



## OBJECTIVE: REASONING ABOUT NON LINEAR PROPERTIES

**Analyzing controllers**

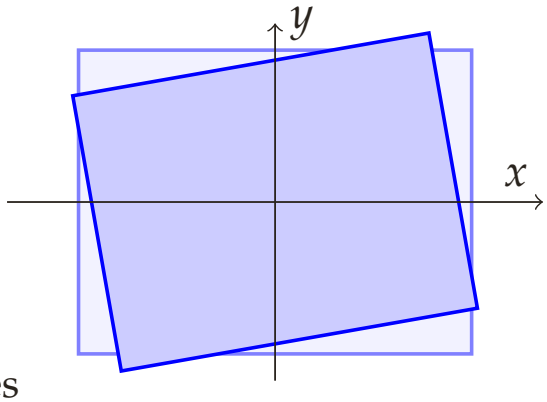
1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.

Few non linear proposals:

- Feret's digital filters in Astrée
  work on a subclass of open-stable linear system: second order filters
- Template domains instrumented with Policy Iteration
  existing instantiation limited to quadratic templates



## OBJECTIVE: REASONING ABOUT NON LINEAR PROPERTIES

**Analyzing controllers**

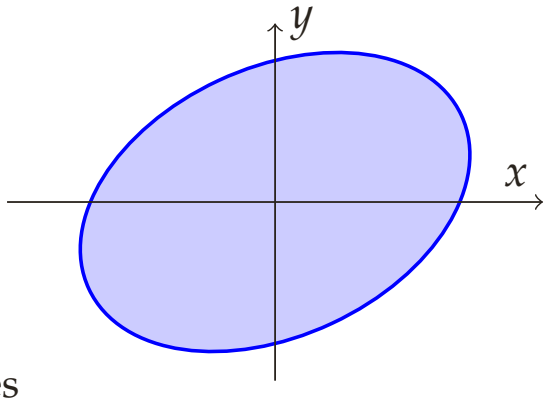
1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.

Few non linear proposals:

- Feret's digital filters in Astrée
  work on a subclass of open-stable linear system: second order filters
- Template domains instrumented with Policy Iteration
  existing instantiation limited to quadratic templates



## OBJECTIVE: REASONING ABOUT NON LINEAR PROPERTIES

**Analyzing controllers**

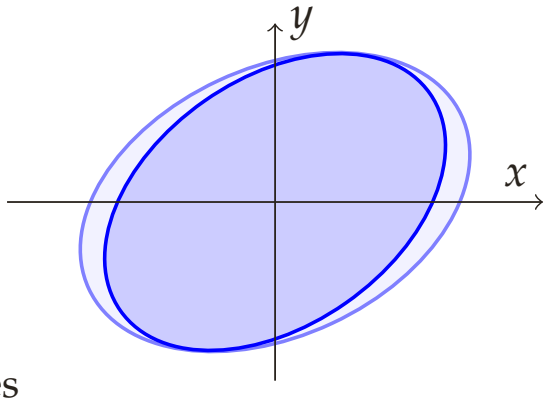
1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

*Linear invariants* – intervals, octagons, polyhedra – commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.

Few non linear proposals:

- Feret's digital filters in Astrée
  work on a subclass of open-stable linear system: second order filters
- Template domains instrumented with Policy Iteration
  existing instantiation limited to quadratic templates



## Objective: reasoning about non linear properties

**Analyzing controllers**

1. Abstract interpretation
2. SMT-based model checking (k-induction)
3. Deductive methods (Weakest precondition)

SMT based model-checking: encodes the system states in SMT and the operational semantics as SMT predicates: I(x) and T(x,y)

Deductive methods manipulate logical expressions (eg. SMT) and transform them according to the program semantics.

Both techniques reason on formulas based on the program/model axiomatisation in SMT.
Nowadays very few non linear reasoning at SMT level

# Content

- Need for non linear invariants
- **Proving stability at code level**
- Non linear invariant synthesis
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- Conclusion

# PROVING STABILITY AT CODE LEVEL

- input: C code annotated with ACSL describing quadratic invariants
- goal: prove the validity of each Hoare triple

File Edit Options Buffers Tools C Help

```
void discrete_timeg_no_plant_compute(t_discrete_timeg_no_plant_io *_io_, t_discrete_timeg_no_plant_state *_state_) {
    REAL VaMux_1[1];
    REAL VaMux_2[2];
    REAL A11;
    REAL A12;
    REAL A21;
    REAL A22;
    REAL C11;
    REAL C12;
    REAL D11;
    REAL Integrator_1;
    REAL Integrator_2;
    REAL Sum;
    REAL Sum1;
    REAL Sum3;
    REAL Sum4;
    /*@
            assert in_ellipsoidQ(QMat_0,vect_of_2_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory));
    */

    discrete_timeg_no_plant_input = _io_->input;
    /*@
            assert in_ellipsoidQ(QMat_1,vect_of_2_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory));
    */

    Integrator_1 = _state_->Integrator_1_memory;
    /*@
            assert in_ellipsoidQ(QMat_2,vect_of_3_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory,Integrator_1));
    */

    C11 = 564.48 * Integrator_1;
    /*@
            assert in_ellipsoidQ(QMat_3,vect_of_4_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory,Integrator_1,C11));
    */

    A21 = 0.01 * Integrator_1;
    /*@
            assert in_ellipsoidQ(QMat_4,vect_of_5_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory,Integrator_1,C11,A21));
    */

    Integrator_2 = _state_->Integrator_2_memory;
    /*@
            assert in_ellipsoidQ(QMat_5,vect_of_6_scalar(_state_->Integrator_1_memory,_state_->Integrator_2_memory,Integrator_1,C11,A21,Integrator_2));
```

## An ellipsoid-aware Hoare logic

- To use ellipsoids to formally specify bounded input, bounded state stability in.

## An ellipsoid-aware Hoare logic

- To use ellipsoids to formally specify bounded input, bounded state stability in.
- Typically, an instruction $S$ would be annotated in the following way:

$$\{x \in \mathcal{E}_P\}\ y = Ax + b\ \{y - b \in \mathcal{E}_Q\} \tag{1}$$

where the pre- and post- conditions are predicates expressing that the variables belong to some ellipsoid, with $\mathcal{E}_p = \{x : \mathbb{R}^n | x^T P^{-1} x \leqslant 1\}$ and $Q = APA^T$.

## An ellipsoid-aware Hoare logic

The mathematical theorem that guarantees the relations is :

**Theorem**

If $M$, $Q$ are invertible matrices, and
$(x - c)^T Q^{-1}(x - c) \leqslant 1$ and
$y = Mx + b$
then
$(y - b - Mc)^T (MQM^T)^{-1}(y - b - Mc) \leqslant 1$

We will refer to it as the *ellipsoid affine combination theorem*.

## Deductive methods

- Predicate transformer + external automatic decision procedure (e.g. SMT solver)
- Weakest precondition computation: $Pre \implies WP(Code, Post)$
- expressiveness of predicate vs. power of decision procedures

In our case:

- code: linear controller
- Post, Pre: quadratic expressions

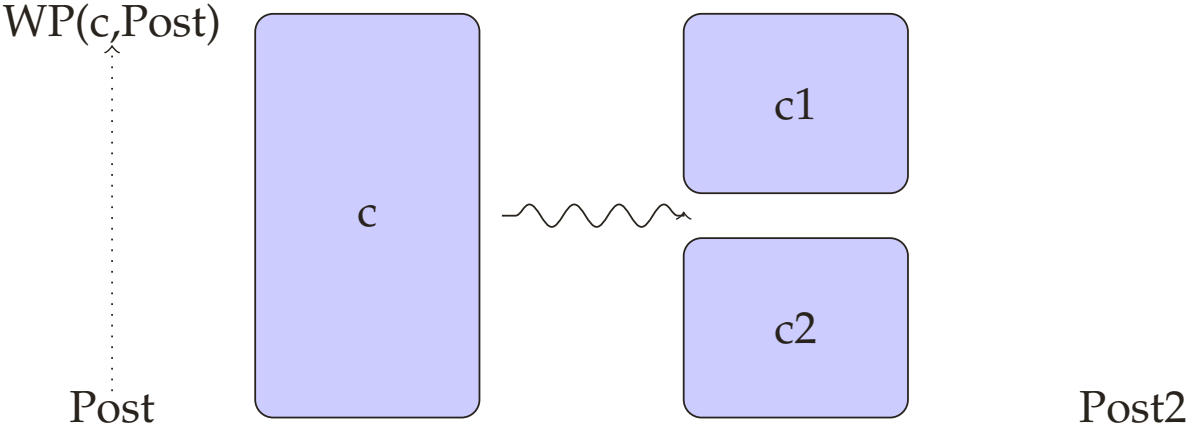
## DEDUCTIVE METHODS

- Predicate transformer + external automatic decision procedure (e.g. SMT solver)
- Weakest precondition computation: $Pre \implies WP(Code, Post)$
- expressiveness of predicate vs. power of decision procedures

In our case:

- code: linear controller
- Post, Pre: quadratic expressions

To ease the process, we can split proofs



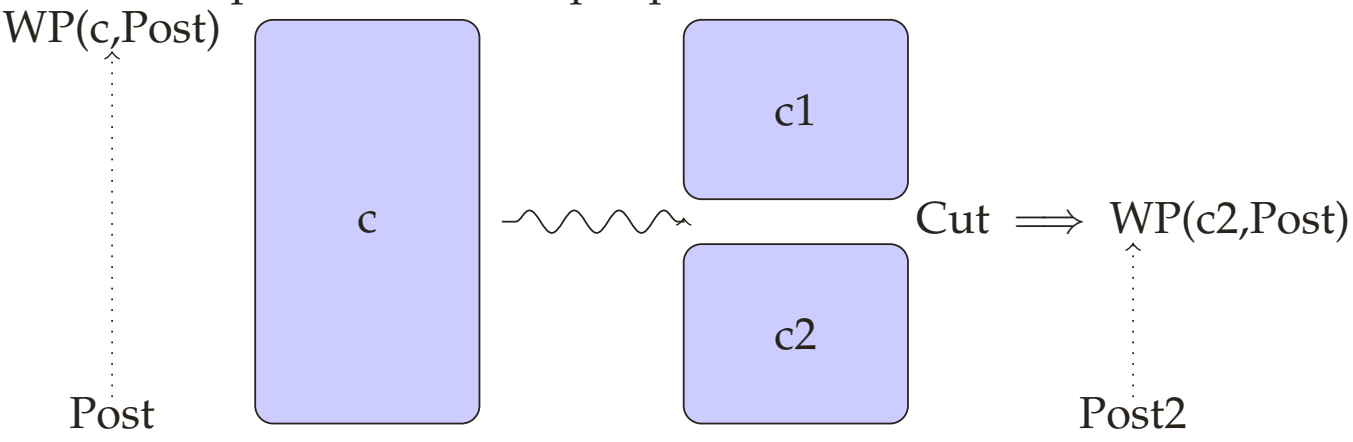
## DEDUCTIVE METHODS

- Predicate transformer + external automatic decision procedure (e.g. SMT solver)
- Weakest precondition computation: $Pre \implies WP(Code, Post)$
- expressiveness of predicate vs. power of decision procedures

In our case:

- code: linear controller
- Post, Pre: quadratic expressions

To ease the process, we can split proofs



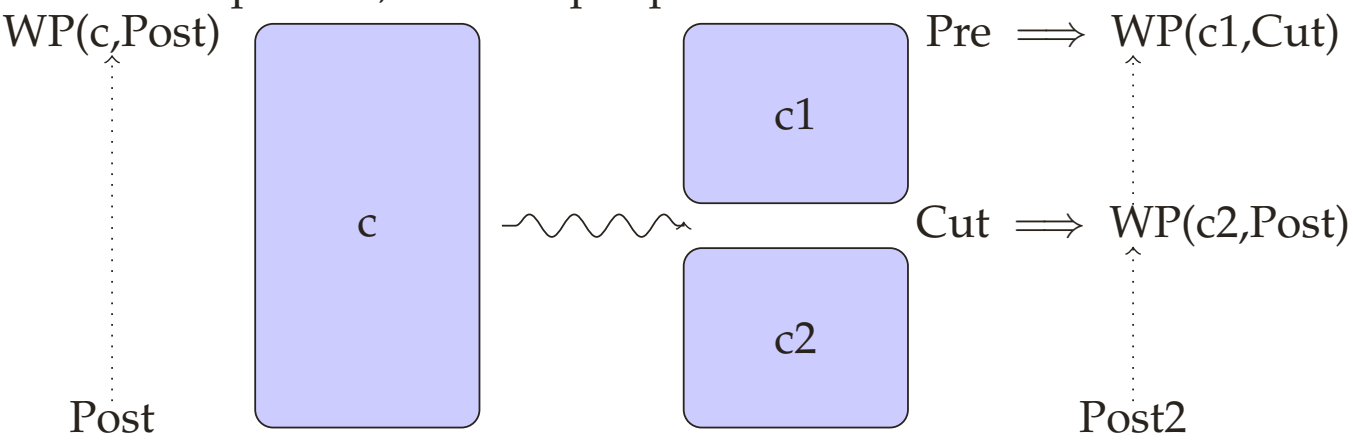
## DEDUCTIVE METHODS

- Predicate transformer + external automatic decision procedure (e.g. SMT solver)
- Weakest precondition computation: $Pre \implies WP(Code, Post)$
- expressiveness of predicate vs. power of decision procedures

In our case:

- code: linear controller
- Post, Pre: quadratic expressions

To ease the process, we can split proofs



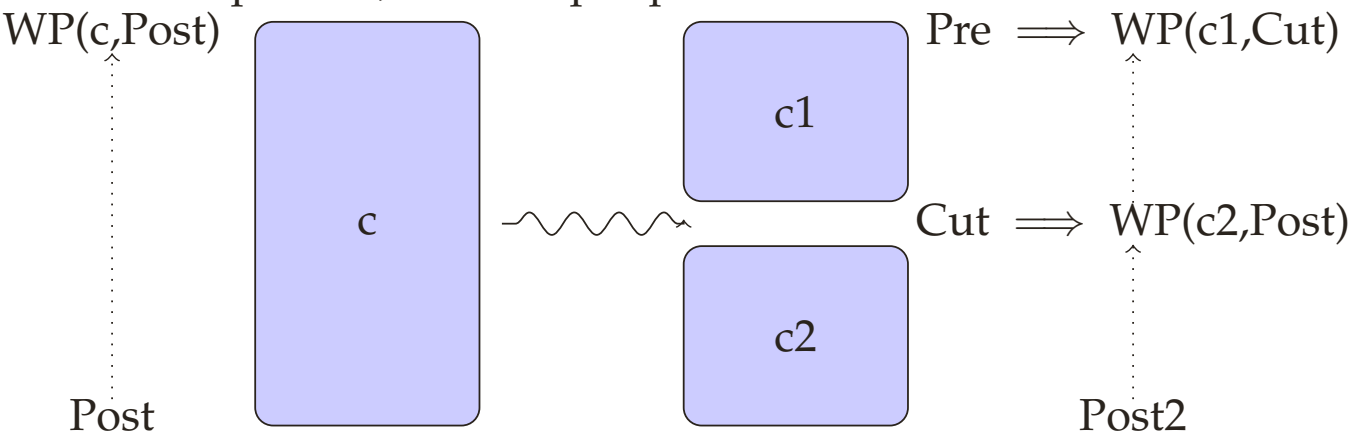
## DEDUCTIVE METHODS

- Predicate transformer + external automatic decision procedure (e.g. SMT solver)
- Weakest precondition computation: $Pre \implies WP(Code, Post)$
- expressiveness of predicate vs. power of decision procedures

In our case:

- code: linear controller
- Post, Pre: quadratic expressions

To ease the process, we can split proofs




In-the-middle annotations act as proof cuts.

## AN ELLIPSOID-AWARE LOGIC

Extension of ACSL to manipulate

- matrices, vectors
- properties over matrices
- ellipsoids
- link between C variables and matrices/vectors

ACSL

```
//@ type matrix; type vector
```

## AN ELLIPSOID-AWARE LOGIC

Extension of ACSL to manipulate

- matrices, vectors
- properties over matrices
- ellipsoids
- link between C variables and matrices/vectors

ACSL

```
//@ type matrix; type vector
```

ACSL

```
@ logic real mat_select(matrix A, integer i, integer j);
@ logic integer mat_row(matrix A);
@ logic integer mat_col(matrix A);
```

## AN ELLIPSOID-AWARE LOGIC

Extension of ACSL to manipulate

- matrices, vectors
- properties over matrices
- ellipsoids
- link between C variables and matrices/vectors

inverse of a matrix $A$, mat_inverse$(A)$ is defined using the predicate is_invertible$(A)$ as follows:

ACSL

```
/*@ axiom mat_inv_select_i_eq_j:
@ ∀matrixA, integer i, j;
@ is_invertible(A) && i == j ==>
@ mat_select(mat_mult(A, mat_inverse(A)), i, j) = 1
@
@ axiom mat_inv_select_i_dff_j:
@ ∀matrixA, integer i, j;
@ is_invertible(A) && i! = j ==>
@ mat_select(mat_mult(A, mat_inverse(A)), i, j) = 0
@*/
```

## AN ELLIPSOID-AWARE LOGIC

Extension of ACSL to manipulate

- matrices, vectors
- properties over matrices
- ellipsoids
- link between C variables and matrices/vectors

Complex constructions or relations can be defined as uninterpreted predicates.

- The following predicate is meant to express that vector $x$ belongs to $\mathcal{E}_P$:

ACSL

```
//@ predicate in_ellipsoid(matrix P, vector x);
```

## AN ELLIPSOID-AWARE LOGIC

Extension of ACSL to manipulate

- matrices, vectors
- properties over matrices
- ellipsoids
- link between C variables and matrices/vectors

Complex constructions or relations can be defined as uninterpreted predicates.

- The following predicate is meant to express that vector $x$ belongs to $\mathcal{E}_P$:

ACSL
```
//@ predicate in_ellipsoid(matrix P, vector x);
```

- mat_of_array or vect_of_array, is used to associate an ACSL matrix type to a C array.

ACSL
```
//@ logic matrix mat_of_array{L}(float *A, integer row,
integer col);
```

## A PVS LIBRARY FOR ELLIPSOIDS

A NASA PVS Library to manipulate

- matrices, vectors
- ellipsoids
- affine combination of ellipsoids (thm1)
- S-procedure (thm2)

PVS

```
Mapping:TYPE= [# dom: posnat, codom: posnat, mp:
[Vector[dom]->Vector[codom]] #]
```

PVS

```
L(n,m)(f) = (# rows:=m, cols:=n, matrix:=λ(j,i):
f`mp(e(n)(i))(j) #)
T(n,m)(A) = (# dom:=n, codom:=m, mp:=λ(x,j): Σ_{i=0}^{A`cols-1}(λ(i):
A`matrix(j,i)*x(i) #))
```

PVS

```
Matrix_inv(n):TYPE = {A: Square | squareMat?(n)(A) and
bijective?(n)(T(n,n)(A))}
```

PVS

```
inv(n)(A) = L(n,n)(inverse(n)(T(n,n)(A)))
```

## A PVS LIBRARY FOR ELLIPSOIDS

A NASA PVS Library to manipulate

- matrices, vectors
- ellipsoids
- affine combination of ellipsoids (thm1)
- S-procedure (thm2)

PVS

```
Vector_no_param: TYPE = [# length: posnat, vect:
vectors[length].Vector #]
```

PVS

```
in_ellipsoid?(P: Matrix, x:Vector_no_param ):
MACRO bool =
IF x'length = P'cols AND P'cols=P'rows
THEN ((x'vect)*(P*(x'vect)) <=1)
ELSE FALSE
ENDIF
```

## A PVS LIBRARY FOR ELLIPSOIDS

A NASA PVS Library to manipulate

- matrices, vectors
- ellipsoids
- affine combination of ellipsoids (thm1)
- S-procedure (thm2)

PVS

```
ellipsoid_affine_comb: LEMMA ∀ (n:posnat, Q, M:
SquareMat(n), x, y, b, c: Vector[n]):
bijective?(n)(T(n,n)(Q)) AND bijective?(n)(T(n,n)(M))
AND (x-c)*(inv(n)(Q)*(x-c))⩽ 1
AND y=M*x + b
IMPLIES
(y-b-M*c)*(inv(n)(M*(Q*transpose(M)))*(y-b-M*c))⩽ 1
```

# Using Frama-C/Jessie/Why do generate PVS proof objective

File Edit Options Buffers Tools C Help

```
/*@ logic matrix Ac = mat_of_2x2_scalar(0.449,-0.05,0.01,1.);
  @ logic matrix P = mat_of_2x2_scalar(1.5325,10.0383,10.0383,507.2450);
  @ logic matrix Q = mat_mult(mat_inv(transpose(Ac)),mat_mult(P,mat_inv(Ac))); */

/*@ requires (\valid(xc + (0..1)));
  @ requires (\valid(yc + (0..1)));
  @ requires in_ellipsoid(P,vect_of_array(xc,2));
  @ ensures in_ellipsoid(Q,vect_of_array(yc,2));*/
void inst2(float* xc, float* yc)
{
  yc[0]= 0.449*xc[0] + -0.05*xc[1];
  yc[1]= .01*xc[0] + 1.*xc[1];
}
```

--:---  pattern_canonical.c  Bot L15  SVN-95  (C/l Abbrev)----------

Generating PVS PO with Frama-C/Jessie/Why:

- www.frama-c.com (open source C code analysis framework)
- axiomatize C semantics into Why (Jessie)
- WP computation (Why)
- PVS backend to express PO

File Edit Options Buffers Tools Help

```
% Why obligation from file "pattern_canonical.c", line 12, characters 12-47:
inst2_ensures_default_po_1: LEMMA
  FORALL (xc: pointer[floatP]) :
  FORALL (yc: pointer[floatP]) :
  FORALL (floatP_alloc_table: alloc_table[floatP]) :
  FORALL (floatP_floatM: memory[floatP,
  real]) :
  (offset_min[floatP](floatP_alloc_table, xc) <= 0 AND
  (offset_max[floatP](floatP_alloc_table, xc) >= 1 AND
  (offset_min[floatP](floatP_alloc_table, yc) <= 0 AND
  (offset_max[floatP](floatP_alloc_table, yc) >= 1 AND
  in_ellipsoid(P_0, vect_of_array(xc, 2, floatP_floatM))))) IMPLIES
  FORALL (result: real) :
  result = select[real, floatP](floatP_floatM, shift[floatP](xc, 0)) IMPLIES
  FORALL (result0: real) :
  result0 = select[real, floatP](floatP_floatM, shift[floatP](xc, 1)) IMPLIES
  FORALL (result1: real) :
  result1 = select[real, floatP](floatP_floatM, shift[floatP](xc, 0)) IMPLIES
  FORALL (result2: real) :
  result2 = select[real, floatP](floatP_floatM, shift[floatP](xc, 1)) IMPLIES
  FORALL (floatP_floatM0: memory[floatP, real]) :
  not_assigns[floatP,
  real](floatP_alloc_table, floatP_floatM, floatP_floatM0,
        pset_range[floatP](pset_singleton[floatP](yc), 0, 1)) AND
  (select[real, floatP](floatP_floatM0, shift[floatP](yc, 0)) =
   0.449 * result + -0.05 * result0 AND
  select[real, floatP](floatP_floatM0, shift[floatP](yc, 1)) =
  0.01 * result1 + 1.0 * result2) IMPLIES
  in_ellipsoid(Q, vect_of_array(yc, 2, floatP_floatM0))
```

--:---  pattern_canonical_why.pvs  93% L584  (Text Fill)----------

## THEORY INTERPRETATION: MAPPING PVS CONCEPTS

Theory interpretation is a logical technique for relating one axiomatic theory to another.

PVS

```
IMPORTING acsl_theory{{ matrix := Matrix,
vector := Vector_no_param,
vect_length := LAMBDA (v:Vector_no_param): v'length,
mat_row := LAMBDA (M:Matrix): M'rows,
mat_col := LAMBDA (M:Matrix): M'cols,
mat_mult := *,
in_ellipsoid := in_ellipsoid?
mat_inv := LAMBDA (M:Matrix): IF square?(M) THEN IF
bijective?(M'rows)(T(M'rows,M'rows)(M))
THEN inv(M'rows)(M)
ELSE M
ENDIF
ELSE M ENDIF }}
```

## REFORMULATED PO

PVS

```
in_ellipsoid?(P_0, vect_of_array(xc, 2, floatP_floatM)))))
IMPLIES
in_ellipsoid?(Q, vect_of_array(yc, 2, floatP_floatM0))
```

PVS

```
vect_of_array(yc, 2, floatP_floatM0)'vect =
Ac * vect_of_array(xc, 2, floatP_floatM)'vect
```

For both POs,

- we must first interpret the uninterpreted types and to prove the properties that are defined axiomatically.

## REFORMULATED PO

PVS

```
in_ellipsoid?(P_0, vect_of_array(xc, 2, floatP_floatM)))))
IMPLIES
in_ellipsoid?(Q, vect_of_array(yc, 2, floatP_floatM0))
```

PVS

```
vect_of_array(yc, 2, floatP_floatM0)'vect =
Ac * vect_of_array(xc, 2, floatP_floatM)'vect
```

For both POs,

- we must first interpret the uninterpreted types and to prove the properties that are defined axiomatically.
- We must then discharge the verification conditions. This is done by using PVS and our linear algebra extension of it.

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,
- proving the stability of the control code using ellipsoid affine combinations.

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,
- proving the stability of the control code using ellipsoid affine combinations.
- We have defined an ACSL extension to describe predicates over the code, as well as a PVS library able to manipulate these predicates.
- Theory interpretation maps proof obligations generated from the code to their equivalent in this PVS library.

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,
- proving the stability of the control code using ellipsoid affine combinations.
- We have defined an ACSL extension to describe predicates over the code, as well as a PVS library able to manipulate these predicates.
- Theory interpretation maps proof obligations generated from the code to their equivalent in this PVS library.
- This mapping allows to discharge POs using the ellipsoid affine combination theorem implemented in PVS.

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,
- proving the stability of the control code using ellipsoid affine combinations.
- We have defined an ACSL extension to describe predicates over the code, as well as a PVS library able to manipulate these predicates.
- Theory interpretation maps proof obligations generated from the code to their equivalent in this PVS library.
- This mapping allows to discharge POs using the ellipsoid affine combination theorem implemented in PVS.
- Linear algebra PVS libraries can be used for the formal specification of control theory properties

## Summary

- We have described a global approach to validate stability properties of C code implementing controllers.
- Our approach requires the code to be annotated by Hoare triples,
- proving the stability of the control code using ellipsoid affine combinations.
- We have defined an ACSL extension to describe predicates over the code, as well as a PVS library able to manipulate these predicates.
- Theory interpretation maps proof obligations generated from the code to their equivalent in this PVS library.
- This mapping allows to discharge POs using the ellipsoid affine combination theorem implemented in PVS.
- Linear algebra PVS libraries can be used for the formal specification of control theory properties

# Content

- Need for non linear invariants
- Proving stability at code level
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- Conclusion

## Providing tools to manipulate/synthesize non linear invariants

Mathematical tools exist to deal with non linear arithmetic:

- Linear systems admit quadratic invariants: use semi definite programming with LMIs
- Polynomial systems admitting polynomial invariants: use Bernstein polynomials
- More complex systems: other tools

## PROVIDING TOOLS TO MANIPULATE/SYNTHESIZE NON LINEAR INVARIANTS

Mathematical tools exist to deal with non linear arithmetic:

- Linear systems admit quadratic invariants: use semi definite programming with LMIs
- Polynomial systems admitting polynomial invariants: use Bernstein polynomials
- More complex systems: other tools

Be careful: most of them rely on floating point computations
$\Longrightarrow$ find ways to validate the result

# Content

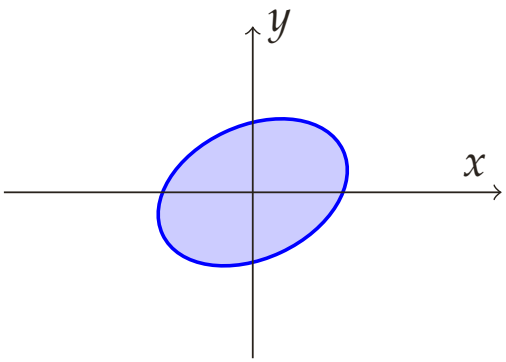
- Need for non linear invariants
- Proving stability at code level
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- Conclusion

## QUADRATIC INVARIANTS

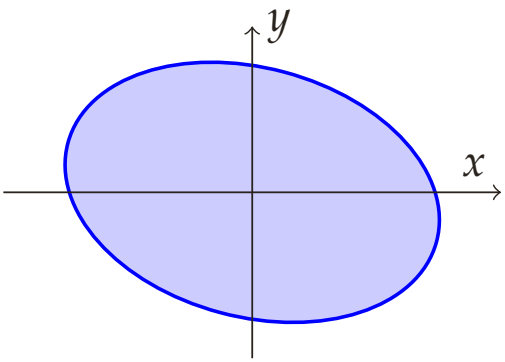
Goal: characterize an ellipsoid such that it is a Lyapunov function for the system
Control theorists rely on LMI and semi definite programming to generate Lyapunov functions.
Let's do the same.

**Overall Method**

1. First determine the *shape* of the ellipsoid by choosing a matrix $P$ such that $A^T P A - P \prec 0$.
2. then find the smallest possible *ratio* $\lambda$ such that $x^T P x \leqslant \lambda$ is an invariant.




is better than



## Methodology - Multiple approaches

### Definition (Semidefinite Programming)

Minimize a linear objective function of variables $y_i$
under constraint

$$A_0 + \sum_{i=1}^{k} y_i A_i \succeq 0$$

where the $A_i$ are known matrices
and "$P \succeq 0$" means $x^T P\, x \geqslant 0$ for all vector $x$.

### Heuristics

1. Minimizing Condition Number: finding the roundest possible solution

$$I \preceq P \preceq rI.$$

2. Preserving the shape: minimizing $r$ s.t.

$$A^T P\, A - rP \preceq 0.$$

3. Handle the inputs (avoid the scale phase)

# Rounding Errors

Actual computations are carried out with floating point numbers leading to *rounding errors*.

**Example**

```
int i = 0;
float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%.0f\n", x);
```

## Rounding Errors

Actual computations are carried out with floating point numbers leading to *rounding errors*.

**Example**

```
int i = 0;
float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%.0f\n", x);
```

Gives 100958!

## Rounding Errors

Actual computations are carried out with floating point numbers leading to *rounding errors*.

**Example**

```
int i = 0;
float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%.0f\n", x);
```

Gives 100958!

We have to distinguish two problems:

- rounding errors *in the analyzed program;*
- and rounding errors *in the analyzer* itself.

## Rounding Errors in the Program

Knowing the precision of the floating point system used
and the dimensions of matrices $A$ and $B$ of the analyzed system,
we can compute two reals $a$ and $b$ such that if

$$(Ax + Bu)^T P (Ax + Bu) \leqslant \lambda$$

then

$$\mathrm{fl}(Ax + Bu)^T P \,\mathrm{fl}(Ax + Bu) \leqslant a^2\lambda + 2ab\sqrt{\lambda} + b^2$$

with $\mathrm{fl}(e)$ the computation of $e$ in any order
and with any IEEE754 rounding mode
(in practice $a$ is near from 1 and $b$ from 0).

## SOUNDNESS OF THE RESULT

- Checking the soundness of the result basically amounts to *checking positive definiteness* of a matrix.
- This is done by carefully bounding the rounding errors in a *Cholesky decomposition*.
- Hence an efficient soundness check (in $O(n^3)$ for an $n \times n$ matrix).

## Experimental Results

| | Shape | Bounds | | Valid. |
|---|---|---|---|---|
| Ex. 1 n=2, 1 input | 0.07 | [140.4; 189.9] | 0.40 | 0.01 |
| | 0.16 | [22.2; 26.5] | 0.28 | 0.01 |
| | 0.23 | [16.2; 17.6] | 0.20 | 0.01 |
| Ex. 2 n=4, 1 input | 0.09 | [18.1; 25.2; 24.3; 33.7] | 0.40 | 0.01 |
| | 0.27 | [6.3; 7.7; 2.2; 3.4] | 0.27 | 0.02 |
| | 0.40 | [1.7; 2.0; 2.2; 2.5] | 0.21 | 0.01 |
| Ex. 3 lead-lag controller n=2, 1 input | 0.07 | $\perp$ | $\perp$ | $\perp$ |
| | 0.17 | [36.2; 36.1] | 0.33 | 0.01 |
| | 0.20 | [38.8; 20.3] | 0.20 | 0.01 |
| Ex. 4 LQG regulator n=3, 1 input | 0.09 | [1.2; 0.9; 0.5] | 0.32 | 0.02 |
| | 0.19 | [0.9; 0.9; 0.9] | 0.26 | 0.01 |
| | 0.24 | [0.7; 0.4; 0.3] | 0.22 | 0.02 |

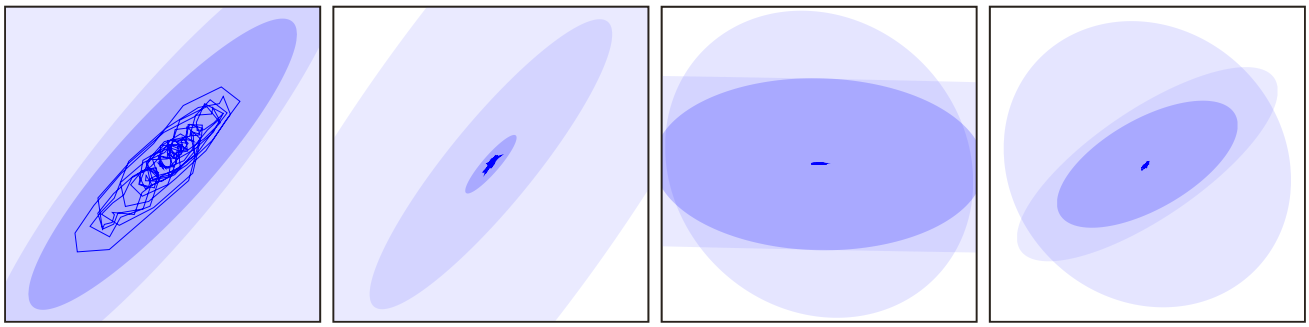
Analysis times (in s) and bounds compared for the three heuristics.

## Experimental Results, continued

| | Shape | Bounds | | Valid. |
|---|---|---|---|---|
| Ex. 5 coupled | 0.09 | [9.8; 8.9; 11.0; 16.8] | 0.43 | 0.03 |
| mass system | 0.24 | [5.7; 5.6; 6.4; 10.1] | 0.33 | 0.03 |
| n=4, 2 inputs | 0.48 | [5.0; 4.9; 4.8; 4.7] | 0.22 | 0.03 |
| Ex. 6 Butterworth | 0.10 | [7.5; 8.7; 6.1; 7.0; 6.5] | 0.38 | 0.03 |
| low-pass filter | 0.32 | [3.6; 5.0; 4.7; 8.1; 8.9] | 0.29 | 0.02 |
| n=5, 1 input | 0.78 | [2.3; 1.1; 1.9; 2.0; 2.9] | 0.24 | 0.03 |
| Ex. 7 Dampened | 0.07 | [1.7; 2.1] | 0.23 | 0.01 |
| oscillator | 0.15 | [2.0; 2.0] | 0.20 | $\perp$ |
| n=2, no input | 0.27 | [1.5; 1.5] | 0.16 | 0.01 |
| Ex. 8 Harmonic | 0.08 | [1.5; 1.5] | 0.23 | 0.01 |
| oscillator | 0.24 | [1.5; 1.5] | 0.20 | $\perp$ |
| n=2, no input | 0.15 | [1.5; 1.5] | 0.16 | 0.01 |

Analysis times (in s) and bounds compared for the three heuristics.

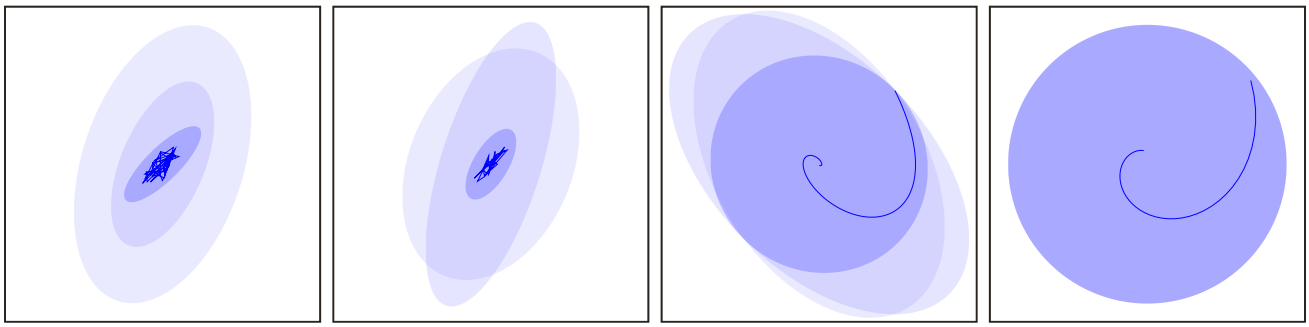
## Experimental Results, continued



(a) Ex. 1 (b) Ex. 2 (c) Ex. 3 (d) Ex. 4

Figure: Comparison of obtained ellipsoids by the three methods from lighter to darker, plus a random simulation trace ((b) and (d), being of dimension greater than 2, are cuts along planes containing the origin and two vectors of the canonical base, to show how the three different templates compare together).

## Experimental Results, end



(a) Ex. 5 (b) Ex. 6 (c) Ex. 7 (d) Ex. 8

Figure: Comparison of obtained ellipsoids by the three methods from lighter to darker, plus a random simulation trace (a) and (b), being of dimension greater than 2, are cuts along planes containing the origin and two vectors of the canonical base, to show how the three different templates compare together).

# Content

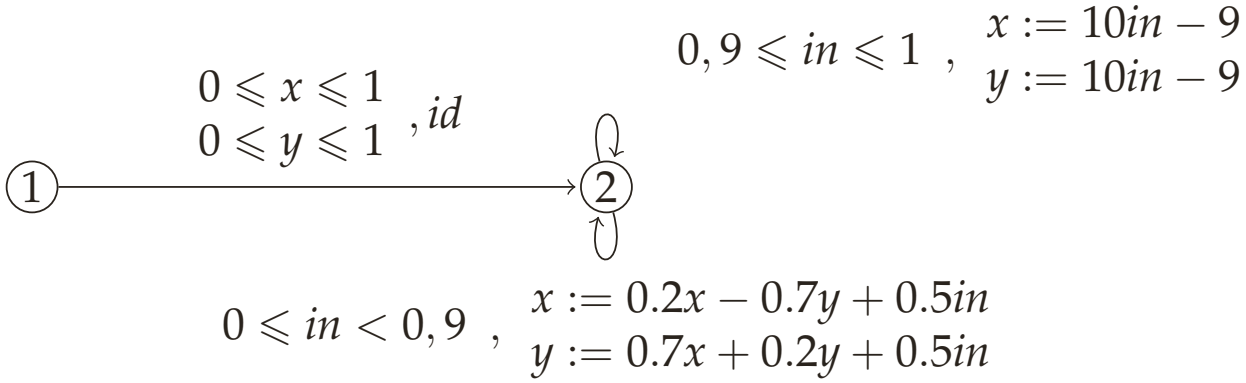
- Need for non linear invariants
- Proving stability at code level
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- Conclusion

## LINEAR SYSTEMS WITH GUARDS: EXTENSION TO POLICY ITERATION

Real systems are not purely linear, they use saturations.

**Extension to policy iterations**

- Build the control flow graph of the program
- Rely on previous approach to compute a set of appropriate templates
- Iterate on program policies with synthesized templates.



# Content

- Need for non linear invariants
- Proving stability at code level
- **Non linear invariant synthesis**
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - ***Beyond linear systems: polynomial controllers***
- Conclusion

## BEYOND LINEAR SYSTEMS

Controllers are rarely linear:

- use of trigonometric functions, exponential ...
- use of polynomials

In static analysis, most complex domains only deal with quadratic properties, using semi definite programming.

Proposal: use Bernstein polynomials to bound polynomial templates.

# Polynomial template domains

## Definition

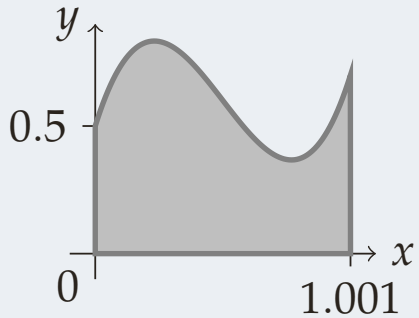
Given a set $P = \{ p_1, \ldots, p_k \}$ of $k \in \mathbb{N}$ polynomials over $n \in \mathbb{N}$ variables, an abstract value is defined as a tuple $(b_1, \ldots, b_k) \in \bar{\mathbb{R}}^k$.

## Concretization

$$\gamma(b_1, \ldots, b_k) = \left\{ (x_1, \ldots, x_n) \in \mathbb{R}^n \;\middle|\; \begin{array}{l} p_1(x_1, \ldots, x_n) \leqslant b_1 \\ \wedge \ldots \wedge \\ p_k(x_1, \ldots, x_n) \leqslant b_k \end{array} \right\}.$$

## Example

$$\begin{array}{rcl} x & \leqslant & 1.001 \\ -x & \leqslant & 0 \\ y & \leqslant & 0.833 \\ -y & \leqslant & 0 \\ y - 6x^3 + 9x^2 - 3.2x & \leqslant & 0.5 \end{array}$$




$(1.001, 0, 0.833, 0, 0.5)$ with $P = \{ x, -x, y, -y, y - 6x^3 + 9x^2 - 3.2x \}$.

# Bernstein polynomials as a polynomial template domain engine

## Approach: express program semantics as an optimization problem

$$\max \{p\,(x_1, \ldots, x_n) \mid q_1\,(x_1, \ldots, x_n) \leqslant b_1 \ \wedge \ \ldots \ \wedge \ q_k\,(x_1, \ldots, x_n) \leqslant b_k\}$$

## Bernstein polynomials

- Bernstein basis:

$$B_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}$$

- Polynomials: every polynomial can be expressed in Bernstein basis

$$p = \sum_{i=0}^{n} b_{p,i} B_{n,i}.$$

- Bound properties: For any polynomial $p$, for all $x \in [0, 1]$,

$$\min \left\{ b_{p,i} \;\middle|\; 0 \leqslant i \leqslant n \right\} \leqslant p(x) \leqslant \max \left\{ b_{p,i} \;\middle|\; 0 \leqslant i \leqslant n \right\}.$$

## A TEMPLATE ABSTRACT DOMAIN

Express each program construct as an optimization problem.

- Guards

$$[\![ r(x_1, \ldots, x_n) \leqslant 0 ]\!]^\sharp \ (b_1, \ldots, b_k) = (b'_1, \ldots, b'_k)$$

with, for $i \in [\![1, k]\!]$: $b'_i = \max \left\{ p_i\,(x_1, \ldots, x_n) \;\middle|\; \begin{array}{l} p_1\,(x_1, \ldots, x_n) \leqslant b_1 \\ \wedge \ \ldots \ \wedge \\ p_k\,(x_1, \ldots, x_n) \leqslant b_k \wedge \\ r\,(x_1, \ldots, x_n) \leqslant 0 \end{array} \right\}$

- Assignments

$$[\![ x_{i_0} := r(x_1, \ldots, x_n) ]\!]^\sharp \ (b_1, \ldots, b_k) = (b'_1, \ldots, b'_k)$$

with, for $i \in [\![1, k]\!]$:

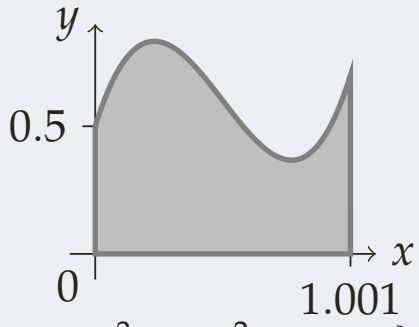
$b'_i = \max \left\{ p_i[x_{i_0} \leftarrow r(x_1, \ldots, x_n)]\,(x_1, \ldots, x_n) \;\middle|\; \begin{array}{l} p_1\,(x_1, \ldots, x_n) \leqslant b_1 \\ \wedge \ \ldots \ \wedge \\ p_k\,(x_1, \ldots, x_n) \leqslant b_k \end{array} \right\}$

## EXAMPLE

x := 0; y := ?(0, 0.5);
**while** x $\leqslant$ 1 **do**
    y := y + 0.001 $\times$ (18x$^2$ $-$ 18x + 3);
    x := x + 0.001;
    **if** y $\leqslant$ 0 **then** y := 0 **else** y := y **fi**
**od**

### Loop invariant at loop head

$$
\begin{array}{rcl}
x & \leqslant & 1.001 \\
-x & \leqslant & 0 \\
y & \leqslant & 0.833 \\
-y & \leqslant & 0 \\
y - 6x^3 + 9x^2 - 3.2x & \leqslant & 0.5
\end{array}
$$




$(1.001, 0, 0.833, 0, 0.5)$ with $P = \{\, x, -x, y, -y, y - 6x^3 + 9x^2 - 3.2x \,\}$.

## SOUNDNESS OF THE RESULT

All computation were done in floats

PVS NASA library for Bernstein polynomials:
Checking that the floating point result is a sound maximum value in real computations.

See NASA Langley Grizzly

# Content

- Need for non linear invariants
- Proving stability at code level
- Non linear invariant synthesis
  - *Quadratic invariants for linear systems*
  - *Linear systems with guards*
  - *Beyond linear systems: polynomial controllers*
- Conclusion

## SUMMARY

Identified need for avionics software: non linear reasoning
Proposals:

- Proof-assistant that is able to discharge proofs about ellipsoids
  - Backend of the Geneauto translation from Simulink + Proof to C code
  - Targeting a fully automatic proof replay at C code level
- Abstract domains building non linear abstract values
  - quadratic invariants for linear controllers (automatic)
  - quadratic invariants for linear controllers with guards (automatic)
  - polynomial invariants for polynomial controllers (need to be provided with templates)

Thanks a lot to all people that participated to these works: Adrien Champion, Rémi Delmas, Éric Féron, Heber Herencia-Zapana, Romain Jobredeaux, Temesghen Kahsai, Steve Miller, Sam Owre, Pierre Roux, Cesare Tinelli, Lucas Wagner, Tim Wang, Mike Whalen.