Teaching formal methods through Frama-C & SPARK

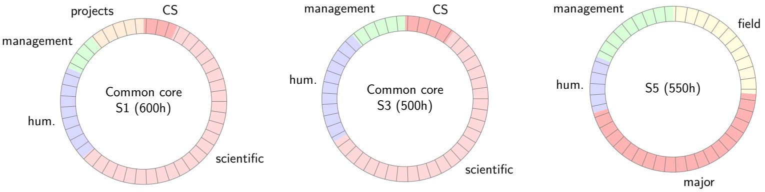
Frama-C and SPARK day 2019

Christophe Garion and Jérôme Hugues (and others)
ISAE-SUPAERO – DISC/IpSC

Outline

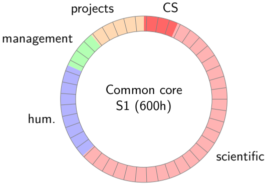
ISAE-SUPAERO engineering program

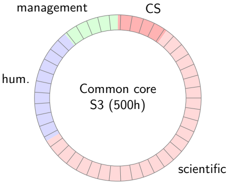
ISAE-SUPAERO is one of the leading French “Grandes Écoles”, mainly focused on **aerospace**, albeit offering other possibilities.

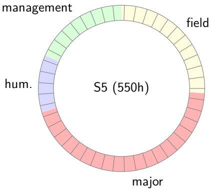


ISAE-SUPAERO engineering program

ISAE-SUPAERO is one of the leading French “Grandes Écoles”, mainly focused on **aerospace**, albeit offering other possibilities.



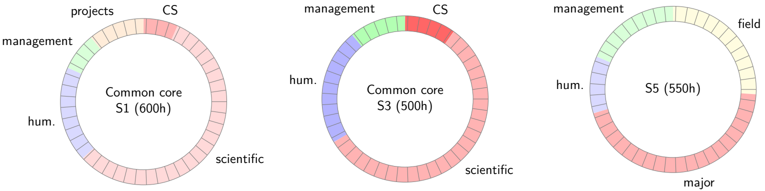




- 40h lecture on Algorithms and Programming in C: algorithms, C programming, data structures (linked lists, BST, binary heaps, graphs)

ISAE-SUPAERO engineering program

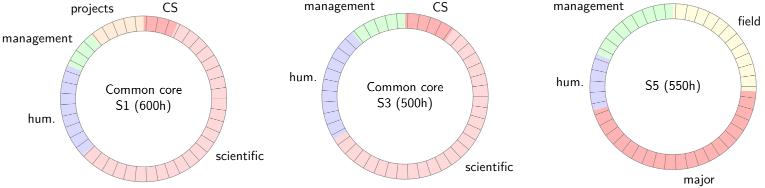
ISAE-SUPAERO is one of the leading French "Grandes Écoles", mainly focused on **aerospace**, albeit offering other possibilities.



- 40h lecture on Object Oriented Design and Programming in Java
- 10h lecture on Integer Linear Programming in S3

ISAE-SUPAERO engineering program

ISAE-SUPAERO is one of the leading French “Grandes Écoles”, mainly focused on **aerospace**, albeit offering other possibilities.

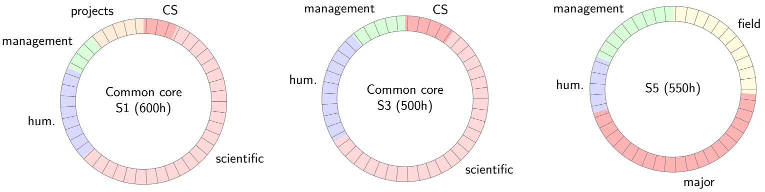


S2 and S4 are dedicated to projects and 30h elective courses e.g.

- functional and logic programming languages
- implementation of control systems
- systems architecture

ISAE-SUPAERO engineering program

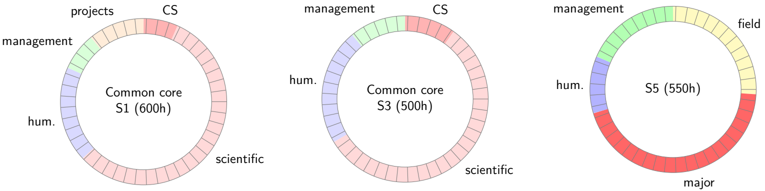
ISAE-SUPAERO is one of the leading French "Grandes Écoles", mainly focused on **aerospace**, albeit offering other possibilities.



Most students do a gap year between S4 and S5 with various experiences: academic, internships, personal projects.

ISAE-SUPAERO engineering program

ISAE-SUPAERO is one of the leading French "Grandes Écoles", mainly focused on **aerospace**, albeit offering other possibilities.



- field of application (140h): aircraft operations & design, space systems, energy, autonomous systems, decision systems, complex systems modeling & simulation
- **major of expertise** (240h) e.g. **critical system architecture**

Teaching formal methods at SUPAERO?

Why?

- as the main industrial sector of SUPAERO is **aerospace**, it seems legitimate
- the students in the **critical system architecture** major should be exposed to formal methods
- it gives more visibility to **CS as a science**

Teaching formal methods at SUPAERO?

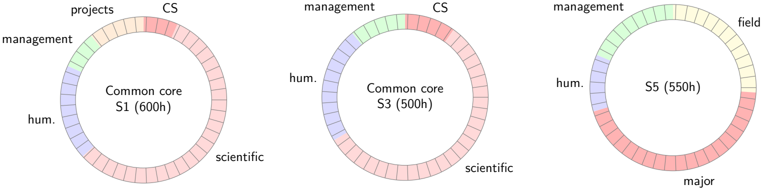
Why?

- as the main industrial sector of SUPAERO is **aerospace**, it seems legitimate
- the students in the **critical system architecture** major should be exposed to formal methods
- it gives more visibility to **CS as a science**

Difficulties?

- the "average" student has only be exposed to **90h of Computer Science** before the last year
- other scientific courses in the common core mainly use **continuous mathematics**
- **(almost) no background in useful mathematics** for formal methods: mathematical logic, calculability theory, SAT/SMT solving etc.

Two experiments



- **SPARK by Example** with two 2nd year students during semester 4
- "classic" **formal methods lecture** in critical system architecture major

Outline

Learning how to prove programs with SPARK

How to learn how to prove complex programs with SPARK?

```
function Inc (X : Integer) return Integer with
  Pre  => X < Integer'Last - 1,
  Post => Inc'Result = X + 1,
  SPARK_Mode is
  begin
     return X + 2 - 1;
  end Inc;
```

how to go there?

Dross, Claire and Yannick Moy (2017). "Auto-Active Proof of Red-Black Trees in SPARK". In: **NASA Formal Methods** .

Available material for learning

For the moment, there are several resources for learning SPARK:

- SPARK 2014 User's Guide by AdaCore
  - ➥ requires familiarity with Ada and some previous knowledge on formal verification
- *Building High Integrity Applications with SPARK* by John McCormick and Peter Chapin
  - ➥ focuses on programming rather than verifying with SPARK
- Introduction to SPARK by AdaCore, an interactive tutorial available on `https://learn.adacore.com/`

Available material for learning

For the moment, there are several resources for learning SPARK:

- SPARK 2014 User's Guide by AdaCore
  - ➥ requires familiarity with Ada and some previous knowledge on formal verification
- *Building High Integrity Applications with SPARK* by John McCormick and Peter Chapin
  - ➥ focuses on programming rather than verifying with SPARK
- Introduction to SPARK by AdaCore, an interactive tutorial available on `https://learn.adacore.com/`

Our impression

Still need a "recipe" document that shows how to develop and prove SPARK programs through classic CS algorithms.

In the C world

There is of course a platform for deductive verification of C programs specified by ACSL, namely Frama-C.

Good references are also available:

- ACSL Frama-C implementation
- Frama-C user manual
- WP plugin manual

In the C world

There is of course a platform for deductive verification of C programs specified by ACSL, namely Frama-C.

Good references are also available:

- ACSL Frama-C implementation
- Frama-C user manual
- WP plugin manual

Jens Gerlach and al. at Fraunhofer Institute have produced a guide, "ACSL by Example":

- specification, implementation and proof of classic CS algorithms extracted from the C++ *Standard Template Library*
- see `https://fraunhoferfokus.github.io/acsl-by-example/`

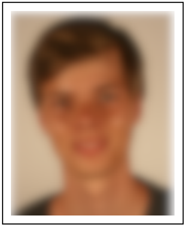
SPARK by Example: the contract

Idea

- provide a booklet in the spirit of "ACSL by Example" in which students can find classical algorithms and learn SPARK "hands-on"
- start from each function presented in "ACSL by Example"
- write a SPARK version of this function, first by translating the C function signature and then by trying to "SPARKify" the function
- compare both approaches

Guinea pigs: our students

Fortunately, we have plenty of students that can be used as guinea pigs to experiment with SPARK ☺



Léo Creuse

Joffrey Huguet

- some background knowledge in theoretical CS (automata, propositional logic), functional programming (Caml) and maths
- no previous knowledge of formal methods, Ada nor SPARK
- small introduction to Floyd-Hoare logic and how to specify programs in SPARK

Experiment

Objective

Will Léo and Joffrey be able to implement, prove and document all algorithms from *ACSL by Example* in SPARK with the 2018 Community Edition of SPARK during a 5-months internship?

Experiment

Objective

Will Léo and Joffrey be able to implement, prove and document all algorithms from *ACSL by Example* in SPARK with the 2018 Community Edition of SPARK during a 5-months internship?

Answer

Yes, they did it in less than 3 months!

Creuse, Léo et al. (2018).
"SPARK by Example: an introduction to formal verification through the standard C++ library".
In: **Proceedings of HILT 2018** .

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms
4. **mutating** algorithms: copy an array, swap values, replace value etc.
   - ➥ first chapter with significant differences between *ACSL by Example* and *SPARK by Example*
   - ➥ using **lemmas functions** in proofs

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms
4. **mutating** algorithms: copy an array, swap values, replace value etc.
   - ➥ first chapter with significant differences between *ACSL by Example* and *SPARK by Example*
   - ➥ using **lemmas functions** in proofs
5. **numeric** algorithms
   - ➥ focuses on **overflow** errors

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms
4. **mutating** algorithms: copy an array, swap values, replace value etc.
   - ➥ first chapter with significant differences between *ACSL by Example* and *SPARK by Example*
   - ➥ using **lemmas functions** in proofs
5. **numeric** algorithms
   - ➥ focuses on **overflow** errors
6. **heap**: a classical implementation of a binary heap with an array
   - ➥ the most difficult chapter

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms
4. **mutating** algorithms: copy an array, swap values, replace value etc.
   - ➥ first chapter with significant differences between *ACSL by Example* and *SPARK by Example*
   - ➥ using **lemmas functions** in proofs
5. **numeric** algorithms
   - ➥ focuses on **overflow** errors
6. **heap**: a classical implementation of a binary heap with an array
   - ➥ the most difficult chapter
7. **sorting** algorithms: quick chapter

Algorithms proved

Algorithms presented in *ACSL by Example* and *SPARK by Example* are extracted from the C++ *Standard Template Library* (STL):

1. **non-mutating** algorithms: find first occurence of an element in an array, count the number of occurrences of an element in an array etc.
2. **maxmin** algorithms return the maximum and minimum value of an array
3. **binary search** algorithms
4. **mutating** algorithms: copy an array, swap values, replace value etc.
   - ➥ first chapter with significant differences between *ACSL by Example* and *SPARK by Example*
   - ➥ using **lemmas functions** in proofs
5. **numeric** algorithms
   - ➥ focuses on **overflow** errors
6. **heap**: a classical implementation of a binary heap with an array
   - ➥ the most difficult chapter
7. **sorting** algorithms: quick chapter
8. **classic sorting**: selection sort, insertion sort, heap sort

Result

- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...

Result

- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...
- ... but with help of the community (thanks Claire, Yannick and people from the `spark2014-discuss` mailing list!)

Result

- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...
- ... but with help of the community (thanks Claire, Yannick and people from the `spark2014-discuss` mailing list!)
- only SMT solvers were used to prove all algorithms (no need to learn Coq for instance)

Result

- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...
- ... but with help of the community (thanks Claire, Yannick and people from the `spark2014-discuss` mailing list!)
- only SMT solvers were used to prove all algorithms (no need to learn Coq for instance)
- two difficult points:

Result

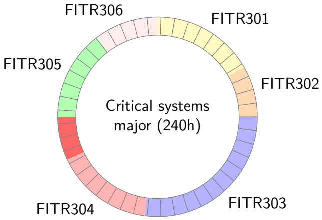
- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...
- ... but with help of the community (thanks Claire, Yannick and people from the `spark2014-discuss` mailing list!)
- only SMT solvers were used to prove all algorithms (no need to learn Coq for instance)
- two difficult points:
  - using **lemmas** through ghost functions to help automatic provers when proving complex functions
    - ➥ you have to discover the mathematical proof

Result

- it is possible to prove (relatively) **complex algorithms** without previous knowledge of formal methods...
- ... but with help of the community (thanks Claire, Yannick and people from the `spark2014-discuss` mailing list!)
- only SMT solvers were used to prove all algorithms (no need to learn Coq for instance)
- two difficult points:
  - using **lemmas** through ghost functions to help automatic provers when proving complex functions
    - ➥ you have to discover the mathematical proof
  - understanding how **SMT solvers work**
    - ➥ quantifiers nesting
    - ➥ understand **triggers**
    - ➥ understand counterexamples

Outline

Content of the critical systems major



FITR301	Network and Computer Architecture
FITR302	Security
FITR303	Real-time Systems
FITR304	Model-Based Engineering
FITR305	Distributed Systems
FITR306	Conferences

FITR304 “Model-Based Engineering” is a 55h lecture with two parts:

- a 38h part on SysML and SCADE
- a **17h slot** for formal methods for validation...

Content of the formal method part

1. **introduction lecture**: what are formal methods, industrial use, programming languages semantics
2. the students choose **one particular formal method** through a track (4 students per track):
   - model checking (J. Brunel – ONERA)
   - abstract interpretation (P.-L. Garoche – ONERA)
   - deductive methods with SPARK (C. Dross – AdaCore)
   - deductive methods with Frama-C (C. Garion – ISAE-SUPAERO)
3. for each track, 6 2h sessions mixing theoretical concepts and labs
   ➥ each track has a specific project to do
4. each student group has 30 minutes to present to the other groups the principles of the technique they used, their result, what was difficult etc.
5. a 2h industrial feedback made by S. Duprat (ATOS) on how (aerospace) industry uses formal methods

Frama-C tracks content

A very classic presentation:

- what is a proof? Formal systems for prop. logic and FOL
- Floyd-Hoare logic
- manual annotation of small algorithms (factorial, GCD etc.) to understand weakest-preconditions
- Frama-C and WP plugin presentation
- gradual hands-on labs to discover Frama-C/WP from basics to axiomatization, pointers, memory separation etc.

➥ **top-down presentation:** from theory to practise

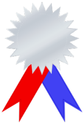
SPARK track content

Claire has a more incremental approach using stronger and stronger levels of verification.

> AdaCore and Thales (2018).
> **Implementation Guidance for the Adoption of SPARK**.
> https://www.adacore.com/books/implementation-guidance-spark.

stone level
valid SPARK

bronze level
init. + data flow



silver level
AoRTE

gold level
contracts

➥ **bottom-up presentation**

Associated projects

Two (similar) projects are done in both tracks.

- Frama-C track: develop a tiny library on strings

```
int strlen(const char *str);
void strsubstring(char *dst, const char *src, int start, int length);
void strappend(char *dst, const char *src);
```

  An (incomplete) axiomatization for `strlen` is given to students. They have to **specify**, **implement** and **prove** the three functions.

- SPARK track: prove a small part of `Ada.Strings.Fixed` GNAT library

```
function Index
  (Source : String;
   Set    : Maps.Character_Set;
   Test   : Membership := Inside;
   Going  : Direction  := Forward) return Natural;

...
```

  Students have to **specify** and **prove** 12 functions.

Students feedback on deductive tracks

Pros

- students complete both projects
- it is cool for them to prove programs
- industrial feedback is important

Students feedback on deductive tracks

Pros

- students complete both projects
- it is cool for them to prove programs
- industrial feedback is important

Cons

- in such a small amount of time, top-down approach is not efficient
  - ➥ better to quickly use Frama-C/SPARK and present theory when needed
- it is not cool for them to write specifications
- they lack theoretical background for complex specifications

Outline

Conclusion

It is possible for non-experts to use Frama-C/SPARK to prove "relatively complex" programs.

But they sometimes lack knowledge/background to:

- understand how SMT solvers work and why they may fail
- understand what is decidable or not
- write complex specifications

Conclusion

It is possible for non-experts to use Frama-C/SPARK to prove "relatively complex" programs.

But they sometimes lack knowledge/background to:

- understand how SMT solvers work and why they may fail
- understand what is decidable or not
- write complex specifications

Industrial feedback by S. Duprat and also C. Dross is important to confort students that these techniques are used in real life.

Conclusion

It is possible for non-experts to use Frama-C/SPARK to prove "relatively complex" programs.

But they sometimes lack knowledge/background to:

- understand how SMT solvers work and why they may fail
- understand what is decidable or not
- write complex specifications

Industrial feedback by S. Duprat and also C. Dross is important to confort students that these techniques are used in real life.

Some ideas:

- begin with Why3 and WhyML instead of "real" programming languages
- add more formal methods with TLA+ in the distributed algorithms course
- create a S4 30h optional course on reliable software

Thanks for your attention

Coffee is just waiting for you, but you can ask questions!