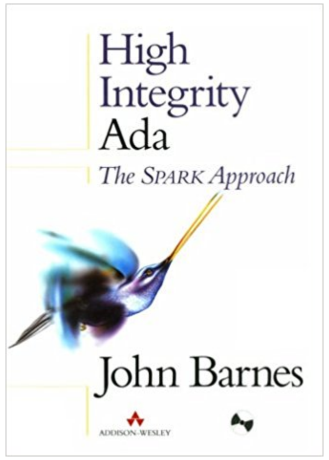
SPARK Language: Historical Perspective & FOSS Development

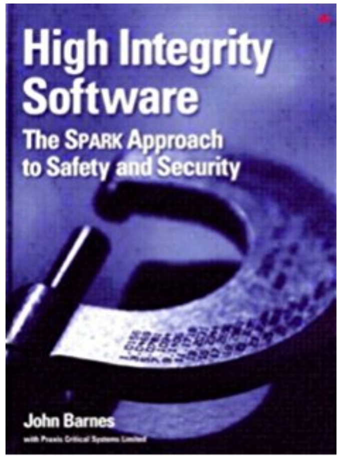
Yannick Moy – SPARK Product Manager – AdaCore


High Integrity Ada

The SPARK Approach

John Barnes

ADDISON-WESLEY


High Integrity Software

The SPARK Approach to Safety and Security

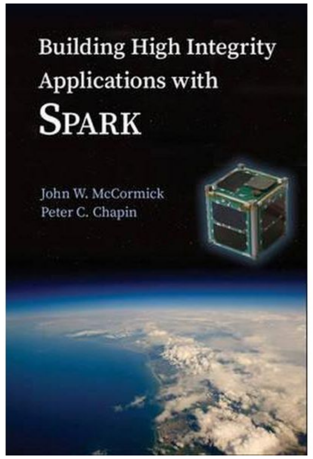
John Barnes

with Praxis Critical Systems Limited

SPARK

The proven approach to HIGH INTEGRITY SOFTWARE

JOHN BARNES


Building High Integrity Applications with SPARK

John W. McCormick

Peter C. Chapin

Implementation Guidance for the Adoption of SPARK

AdaCore THALES

…

Historical Perspective

HI-LITE
Simplifying the use of formal methods

1987
SPARK
PVL

2008
AdaCore altran
PARTNERSHIP

SPARK2014

1997
C130J

2005
Tokeneer

2011
iFACTS

2013
Muen

Are We There Yet? 20 Years of Industrial Theorem Proving with SPARK, Chapman and Schanda, Altran, ITP 2014

From SPARK 2005…

HI-LITE
Simplifying the use of formal methods

```
function Add (X , Y : My_Int) return My_Int;
--# return Sum => (X + Y < 10_000 and Sum = X + Y) or
--#                (X + Y >= 10_000 and Sum = 10_000);

function Mult (X , Y : My_Int) return My_Int;
--# return Prod => (X * Y < 10_000 and Prod = X * Y) or
--#                 (X * Y >= 10_000 and Prod = 10_000);

function Price_Of_Item (It : Item) return Sat.My_Int;
--# return Sat.Mult (It.Price, It.Number);

function Price_Of_Basket (Bk : Basket) return Sat.My_Int;
--# return Price => for all It in Positive range Bk'Range =>
--#                    (Price >= Price_Of_Item (Bk (It)));
```

http://www.open-do.org/projects/hi-lite/a-lighter-introduction-to-hi-lite/

...to SPARK 2014

HI-LITE
Simplifying the use of formal methods

```
function Add (X, Y : My_Int) return My_Int with
  Contract_Cases => (X + Y < 10_000  => Add'Result = X + Y,
                     X + Y >= 10_000 => Add'Result = 10_000);

function Mult (X, Y : My_Int) return My_Int with
  Contract_Cases => (X * Y < 10_000  => Mult'Result = X * Y,
                     X * Y >= 10_000 => Mult'Result = 10_000);

function Price_Of_Item (It : Item) return Sat.My_Int with
  Post => Price_Of_Item'Result = Sat.Mult (It.Price, It.Number);

function Price_Of_Basket (Bk : Basket) return Sat.My_Int with
  Post => (for all It in Positive range Bk'Range =>
             Price_Of_Basket'Result >= Price_Of_Item (Bk (It)));
```

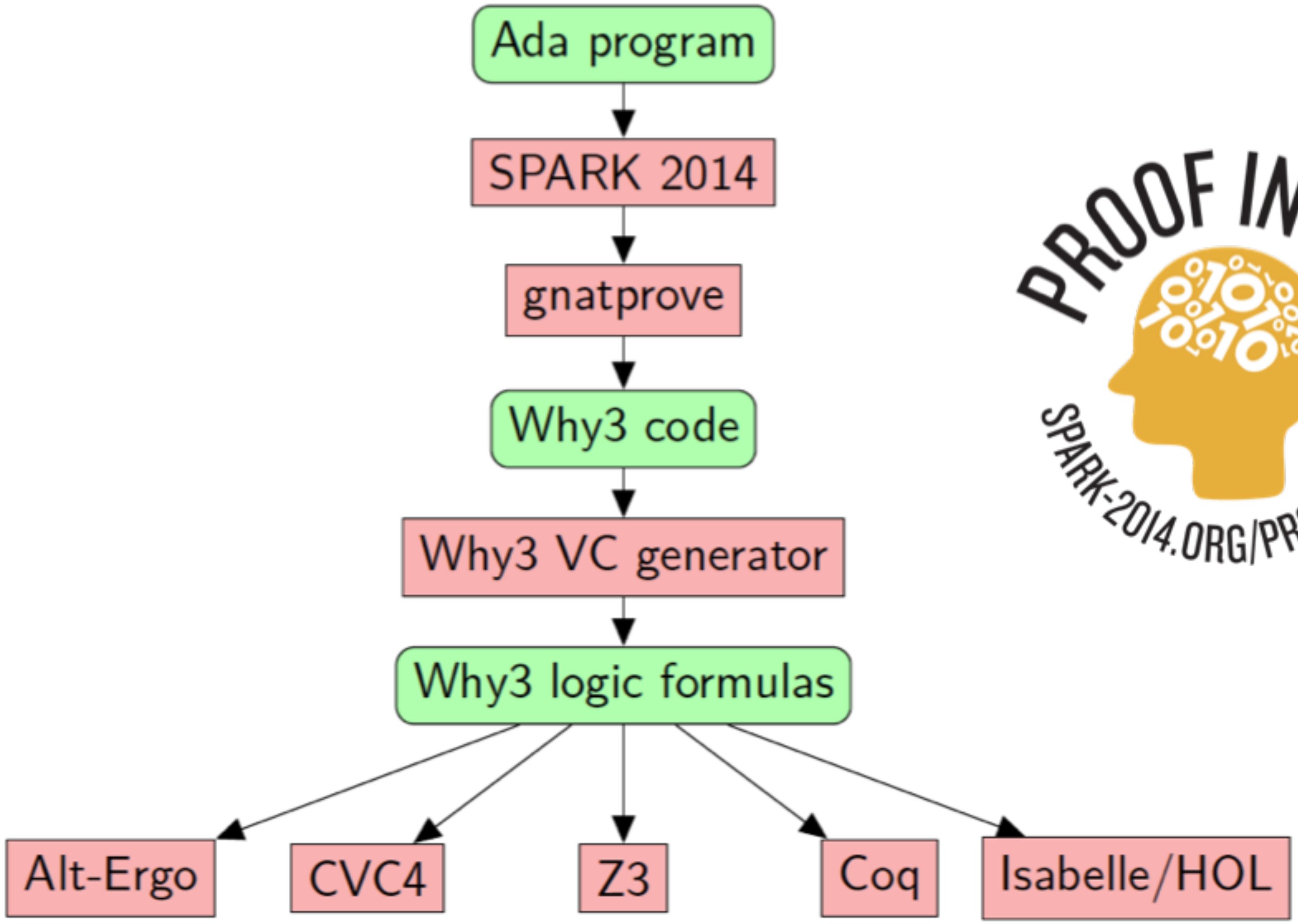
…to SPARK 2014

HI-LITE
Simplifying the use of formal methods

```
function Add (X, Y : My_Int) return My_Int with
  Contract_Cases ⇒ (X + Y < 10_000  ⇒ Add'Result = X + Y,
                    X + Y ≥ 10_000 ⇒ Add'Result = 10_000);

function Mult (X, Y : My_Int) return My_Int with
  Contract_Cases ⇒ (X * Y < 10_000  ⇒ Mult'Result = X * Y,
                    X * Y ≥ 10_000 ⇒ Mult'Result = 10_000);

function Price_Of_Item (It : Item) return Sat.My_Int with
  Post ⇒ Price_Of_Item'Result = Sat.Mult (It.Price, It.Number);

function Price_Of_Basket (Bk : Basket) return Sat.My_Int with
  Post ⇒ (for all It in Positive range Bk'Range ⇒
             Price_Of_Basket'Result ≥ Price_Of_Item (Bk (It)));
```

… in a monospace font with ligatures like FiraCode

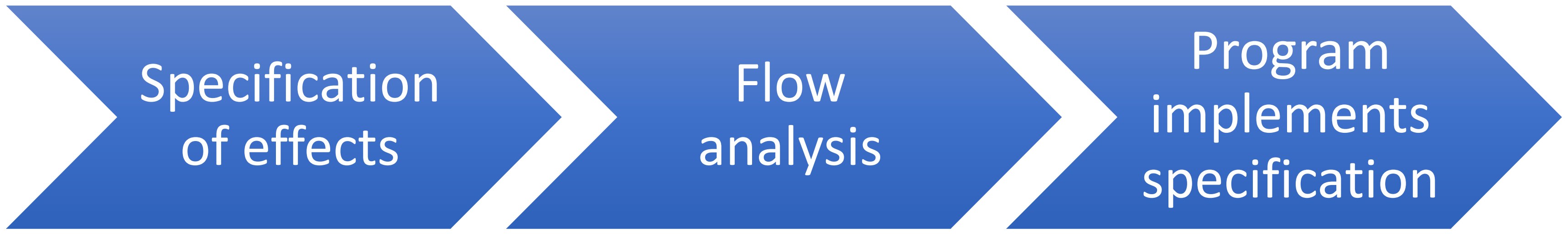
SPARK Open Source Ecosystem



PROOF IN USE
SPARK-2014.ORG/PROOFINUSE

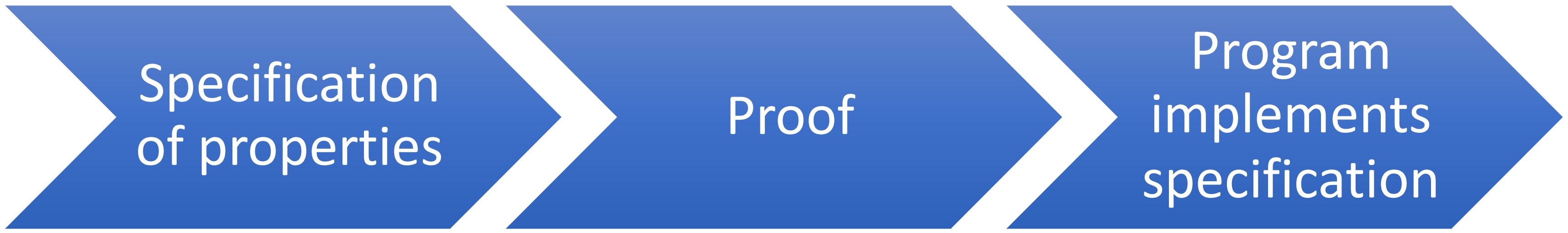
SPARK Flow Analysis

```
procedure Stabilize (Mode    :  in Mode_T;
                     Success : out Boolean)
  with Global => (Input  => (Accel, Giro),
                  In_Out => Rotors);
```



SPARK Proof

```
procedure Stabilize (Mode    :  in Mode_T;
                     Success : out Boolean)
  with Pre  => Mode /= Off,
       Post => (if Success then
                  Delta_Change (Rotors'Old, Rotors));
```



Main Objectives for SPARK 2014

Functional contracts can be executed, tested, debugged

Ada subset supported is as large as possible

User needs no annotation to start proving code

User needs few annotations to fully prove code

Manual proof of formulas is not needed

Contracts can be executed, tested, debugged

Use the Ada 2012 preconditions (aspect Pre) and postconditions (aspect Post)

CONTRACTS = CODE

→ Needed quantified-expressions in Ada 2012

```
(for [some/all] V in Low .. High => Property(V))
```

→ Needed expression-functions in Ada 2012

```
function Property (V : T) return Boolean is (…);
```

Ada subset supported is as large as possible

Only exclude features that make formal verification impossible:

1. Pointers (but references and addresses are OK)
2. Exceptions (but raising one is OK)

Support in particular all types (except access and tagged), no restriction on control flow, recursion, generics

Initial version of SPARK 2014 did not yet support OO programming, concurrency, data invariants, but…

Ada subset supported is expanding

Support for OO programming in 2015, based on Liskov Substitution Principle

Support for concurrency in 2016, based on Ravenscar

Support for type predicates in 2016 and for type invariants in 2017

Support for safe ownership (Rust-like) pointers in 2019-2020

User needs no annotation to start proving

Subprogram signature defines a default functional contract:

- Precondition: inputs (parameters and global variables) in their types
- Postcondition: outputs (parameters and global variables) in their types

Global variables read/written generated by the tool when not provided

User needs fewer annotations to fully prove code

Proof is mostly modular

→ Preconditions and postconditions needed to analyze calls

Inlining mechanisms to do without annotations:

- Inlining of internal subprograms with no contracts
- Unrolling of simple for-loops

Factorization of annotations with data invariants

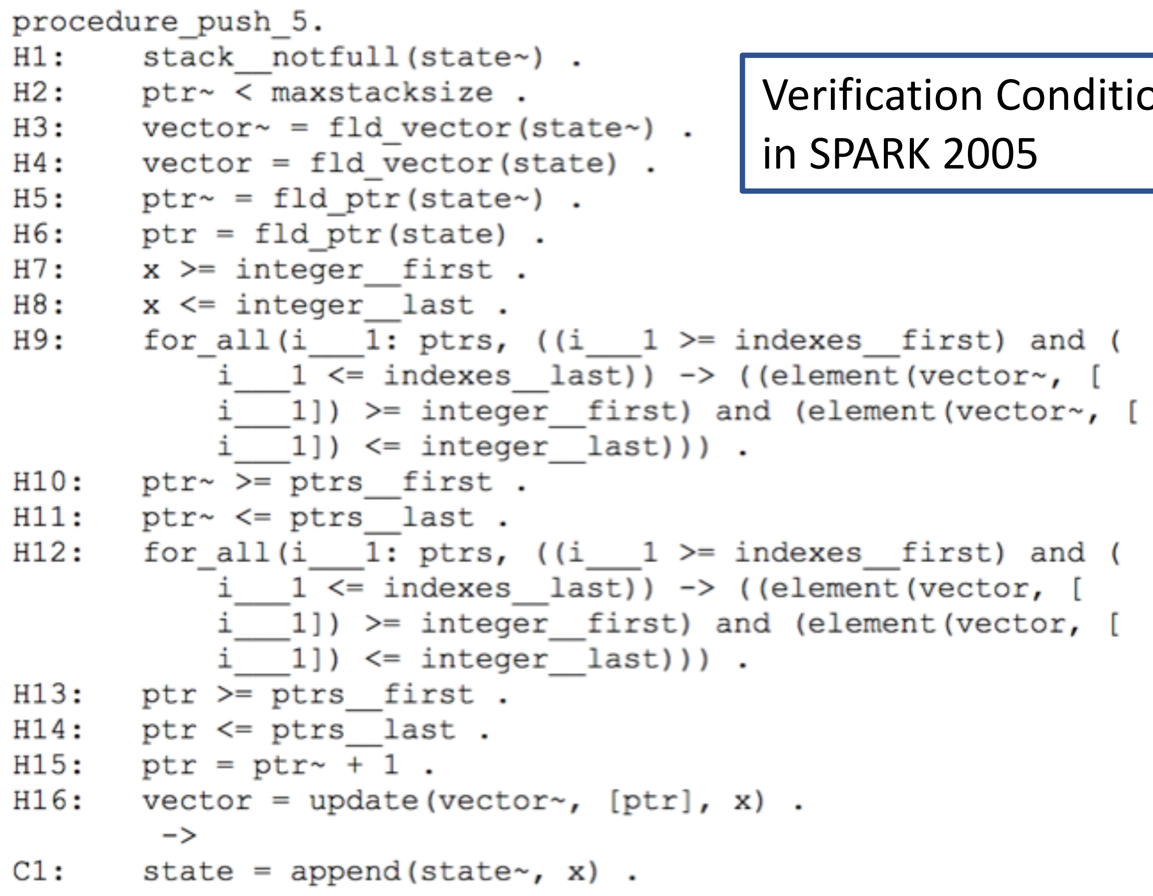
Better generation of formulas → fewer loop invariants, no cutpoints

User really needs few^{er} annotations!

Example: SPARKSkein Skein cryptographic hash algorithm (Chapman, 2011)
http://www.spark-2014.org/entries/detail/sparkskein-from-tour-de-force-to-run-of-the-mill-formal-verification

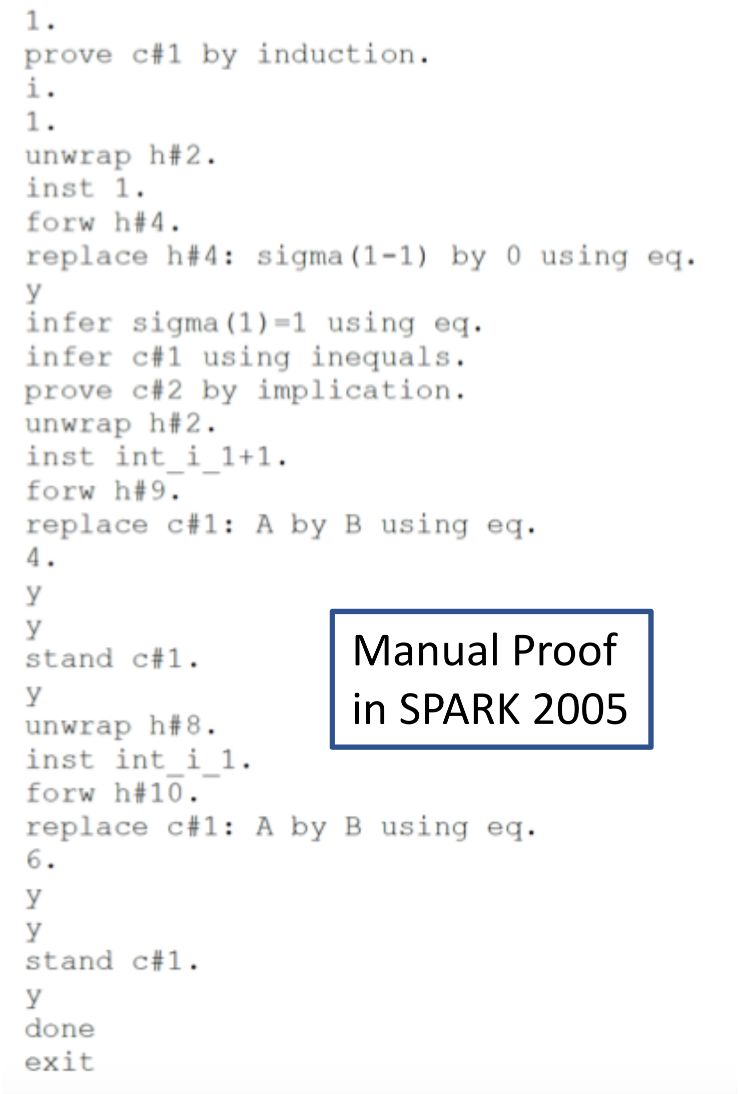
initial version (SPARK 2005)	current version (SPARK 2014)
41 non-trivial contracts for effects and dependencies	1 – effects and dependencies are generated
31 conditions in preconditions and postconditions on internal subprograms	0 – internal subprograms are inlined
43 conditions in loop invariants	1 – loop frame conditions are generated
23 annotations to prevent combinatorial explosion	0 – no combinatorial explosion

Manual proof of formulas was needed



```
procedure_push_5.
H1:    stack__notfull(state~) .
H2:    ptr~ < maxstacksize .
H3:    vector~ = fld_vector(state~) .
H4:    vector = fld_vector(state) .
H5:    ptr~ = fld_ptr(state~) .
H6:    ptr = fld_ptr(state) .
H7:    x >= integer__first .
H8:    x <= integer__last .
H9:    for_all(i___1: ptrs, ((i___1 >= indexes__first) and (
           i___1 <= indexes__last)) -> ((element(vector~, [
           i___1]) >= integer__first) and (element(vector~, [
           i___1]) <= integer__last))) .
H10:   ptr~ >= ptrs__first .
H11:   ptr~ <= ptrs__last .
H12:   for_all(i___1: ptrs, ((i___1 >= indexes__first) and (
           i___1 <= indexes__last)) -> ((element(vector, [
           i___1]) >= integer__first) and (element(vector, [
           i___1]) <= integer__last))) .
H13:   ptr >= ptrs__first .
H14:   ptr <= ptrs__last .
H15:   ptr = ptr~ + 1 .
H16:   vector = update(vector~, [ptr], x) .
        ->
C1:    state = append(state~, x) .
```

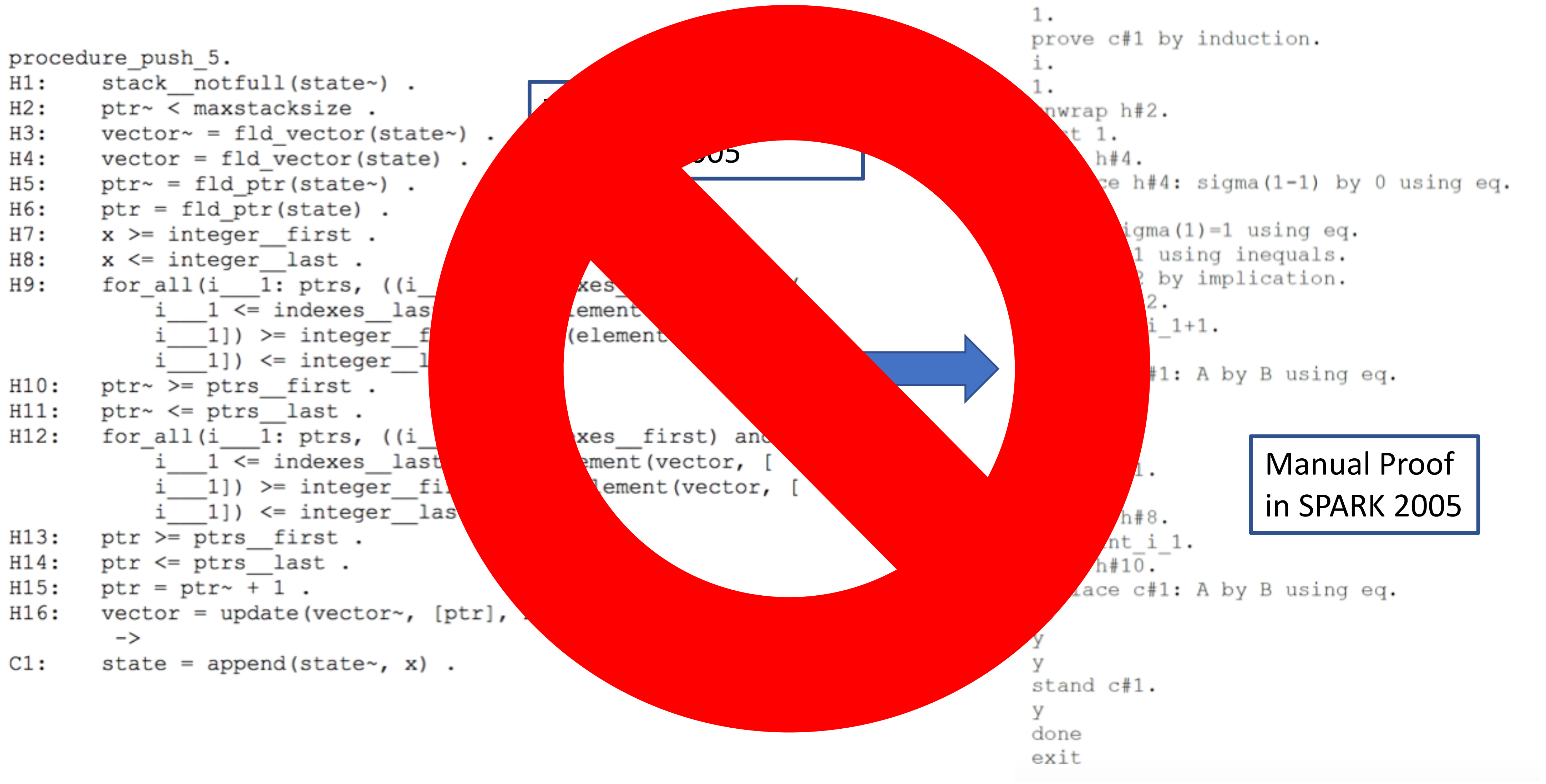
Verification Condition in SPARK 2005



```
1.
prove c#1 by induction.
i.
1.
unwrap h#2.
inst 1.
forw h#4.
replace h#4: sigma(1-1) by 0 using eq.
y
infer sigma(1)=1 using eq.
infer c#1 using inequals.
prove c#2 by implication.
unwrap h#2.
inst int_i_1+1.
forw h#9.
replace c#1: A by B using eq.
4.
y
y
stand c#1.
y
unwrap h#8.
inst int_i_1.
forw h#10.
replace c#1: A by B using eq.
6.
y
y
stand c#1.
y
done
exit
```

Manual Proof in SPARK 2005

Manual proof of formulas was needed



```
procedure_push_5.
H1:    stack__notfull(state~) .
H2:    ptr~ < maxstacksize .
H3:    vector~ = fld_vector(state~) .
H4:    vector = fld_vector(state) .
H5:    ptr~ = fld_ptr(state~) .
H6:    ptr = fld_ptr(state) .
H7:    x >= integer__first .
H8:    x <= integer__last .
H9:    for_all(i___1: ptrs, ((i_        xes_
           i___1 <= indexes__las        ement
           i___1]) >= integer__f        (element
           i___1]) <= integer__l
H10:   ptr~ >= ptrs__first .
H11:   ptr~ <= ptrs__last .
H12:   for_all(i___1: ptrs, ((i_        xes__first) and
           i___1 <= indexes__last       ement(vector, [
           i___1]) >= integer__fi       lement(vector, [
           i___1]) <= integer__las
H13:   ptr >= ptrs__first .
H14:   ptr <= ptrs__last .
H15:   ptr = ptr~ + 1 .
H16:   vector = update(vector~, [ptr],
        ->
C1:    state = append(state~, x) .
```

005

```
1.
prove c#1 by induction.
i.
1.
nwrap h#2.
t 1.
 h#4.
e h#4: sigma(1-1) by 0 using eq.

igma(1)=1 using eq.
1 using inequals.
 by implication.
2.
i_1+1.

#1: A by B using eq.


1.

h#8.
nt_i_1.
h#10.
ace c#1: A by B using eq.

y
y
stand c#1.
y
done
exit
```

Manual Proof in SPARK 2005

Manual proof of formulas is not needed

Use of state-of-the-art SMT solvers: Alt-Ergo, CVC4, Z3

- Why3 platform adapts each formula for each prover
- Mix of arithmetic and quantified properties natively understood by these provers

Encoding of data in logic tailored for automatic proof by SMT solvers

- Encoding not tailored for manual proof

User control over proof strategy (provers combination, timeout)

FOSS Projects in SPARK

Aida library

https://github.com/joakim-strandberg/aida_2012

Library suitable for use in SPARK code, mostly coded in SPARK:

- Bounded strings
- Bounded hash maps, vectors
- UTF8 support
- XML SAX & DOM parsers
- JSON SAX & DOM parsers
- Directories, stream & textual input-output

Certyflie drone software

https://github.com/AdaCore/Certyflie

Rewrite of the original Crazyflie firmware in SPARK:

- FreeRTOS replaced by Ravenscar
- C stabilization and communication code rewritten in SPARK (AoRTE proof)

Demo feature: free-fall detection and landing

Used for prototyping, teaching and research

- Sogilis using it for prototyping
- Jérôme Hugues (ISAE-Supaero) using it for teaching/research

PolyORB-HI

https://github.com/OpenAADL/polyorb-hi-ada

High-integrity middleware for code generation from AADL:

- marshalling and unmarshalling facilities
- messages management
- patterns for periodic, sporadic tasks etc.

Proof of AoRTE + functional contracts (see Frama-C & SPARK Day 2017 - https://frama-c.com/FCSD17.html)

Pulsar drone autopilot

https://www.hionos.com/#pulsar

No public code repository yet

- Part of ongoing funded research project CAP2018
- Should be available by end of 2018

Autopilot developed with agile process targeting civil avionics certification (DO-178C level A)

SPARK used for proving some of the functionalities + AoRTE

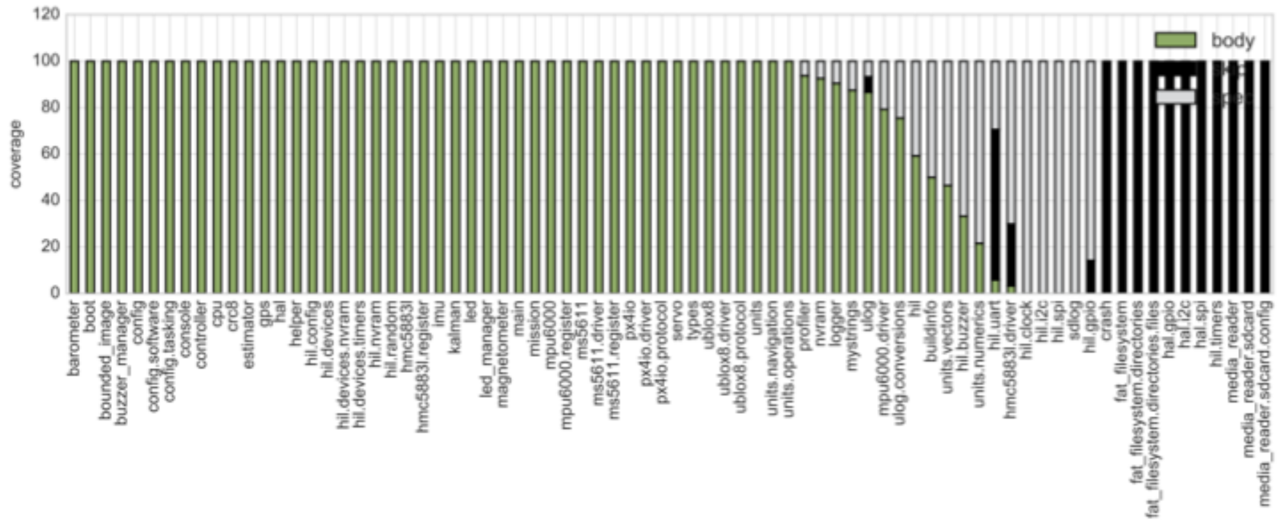
StratoX glider software

https://github.com/tum-ei-rcs/StratoX

Firmware to control an unmanned fixed-wing glider model

Proof of AoRTE + functional contracts (see Frama-C & SPARK Day 2017 - https://frama-c.com/FCSD17.html)

Unit SPARK coverage:



Tokeneer biometric enclave

https://www.adacore.com/tokeneer

https://github.com/AdaCore/spark2014/tree/master/testsuite/gnatprove/tests/tokeneer

Demo project done by Altran for NSA in 2005, open-sourced in 2008

- All project artifacts & statistics collected and available
- Code fully annotated with contracts, even if not needed anymore

Goal of achieving very high level of security (EAL 5)

Muen separation kernel

https://muen.sk/

Developed since 2013 at University of Rapperswil (Switzerland) with secunet (Germany)

Runs on Intel x86/64 platform

First version in 2015: 3000 sloc SPARK, 300 sloc assembly

Just released version 0.9

- Project website served by MirageOS on Muen!

Muen separation kernel

The Muen Separation Kernel is the world's first Open Source microkernel that has been formally proven to contain no runtime errors at the source code level.

Originally written in SPARK 2005. Then fully migrated to SPARK 2014.

Muen vs Meltdown/Spectre

Muen not vulnerable to Meltdown: *Meltdown is defended by our design decision to have a simple architecture which only utilizes a single isolation mechanism: hardware virtualization.*

https://groups.google.com/forum/#!topic/muen-dev/1ILwIz8h-kM

Muen little vulnerable to Spectre: *The Muen kernel is affected by Spectre (one indirect jump in debug build, one indirect access after range check). The observed issues can be fixed with small local changes and no architectural modifications.*

https://groups.google.com/forum/m/#!topic/muen-dev/4tC3MbPxTOQ

SPARK Community Resources

SPARK Community Releases

Every year in June - https://www.adacore.com/community

SPARK will be bundled with GNAT in the Community release 2018

Current differentiator between SPARK Pro and SPARK Discovery:

- Provers CVC4 and Z3 not shipped in SPARK Discovery
- Static analyzer CodePeer not shipped in SPARK Discovery
- As a result, counterexamples not available, and proof less automatic

Installation of CVC4 and Z3 documented in SPARK User’s Guide

- Section “Installing CVC4 and Z3 for SPARK Discovery”

SPARK Learning Resources

AdaCore University – 5-module class on SPARK

u.adacore.com → will move to new Ada/SPARK learning website

This + advanced 5-module class on SPARK on AdaCoreU GitHub:

https://github.com/AdaCoreU

Blog http://www.spark-2014.org/ → will move to AdaCore blog in 2018

Online SPARK RM, SPARK User's Guide, distributed examples, booklet

SPARK Learning Community Resources

SPARK by Example - https://github.com/yoogx/spark_examples

by researchers Jérôme Hugues and Christophe Garion

Similar to ACSL-by-Example by Fraunhofer for Frama-C

(not to be confused with "GNATprove by Example" section of SPARK UG)

Introduction to SPARK - https://www.rcs.ei.tum.de/spark2014-intro/

by researcher Martin Becker

You're developing your own material? Let us know!

SPARK Community Events

SPARK and Frama-C Days 2018 at NIST (near Washington DC)

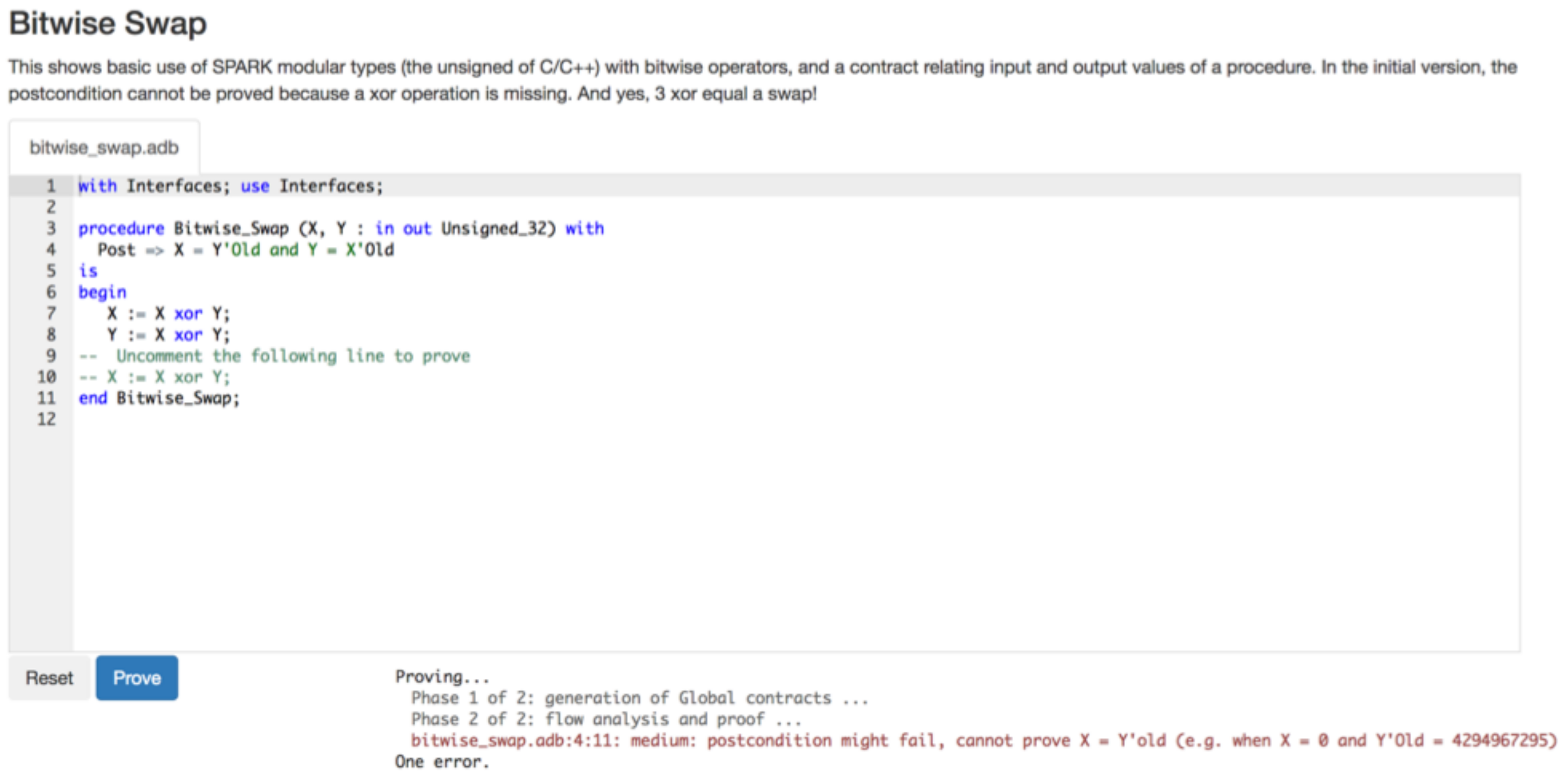
- Keynotes by Dave Wheeler, Rustan Leino, David Cok
- Talks, tutorials
- June 27-28

Presentations at conferences

- Alexander Senier at BOB Conference 2018 (Berlin, February 23) "What happens when we use what's best for a change? » (also in Embedded, mobile and automotive devroom)
- Ada Europe (Lisbon, June 18-22)

Online proof with SPARK

https://cloudchecker.r53.adacore.com/



Bitwise Swap

This shows basic use of SPARK modular types (the unsigned of C/C++) with bitwise operators, and a contract relating input and output values of a procedure. In the initial version, the postcondition cannot be proved because a xor operation is missing. And yes, 3 xor equal a swap!

bitwise_swap.adb

```
with Interfaces; use Interfaces;

procedure Bitwise_Swap (X, Y : in out Unsigned_32) with
  Post => X = Y'Old and Y = X'Old
is
begin
   X := X xor Y;
   Y := X xor Y;
--  Uncomment the following line to prove
-- X := X xor Y;
end Bitwise_Swap;
```

```
Proving...
  Phase 1 of 2: generation of Global contracts ...
  Phase 2 of 2: flow analysis and proof ...
  bitwise_swap.adb:4:11: medium: postcondition might fail, cannot prove X = Y'old (e.g. when X = 0 and Y'Old = 4294967295)
One error.
```

What’s your FOSS Project in SPARK?