altran

PRAXIS

THE CENTRE FOR EMBEDDED AND CRITICAL SYSTEMS

[SPARK]

SPARK Update – Ada Europe 2012

Stuart Matthews

Contents

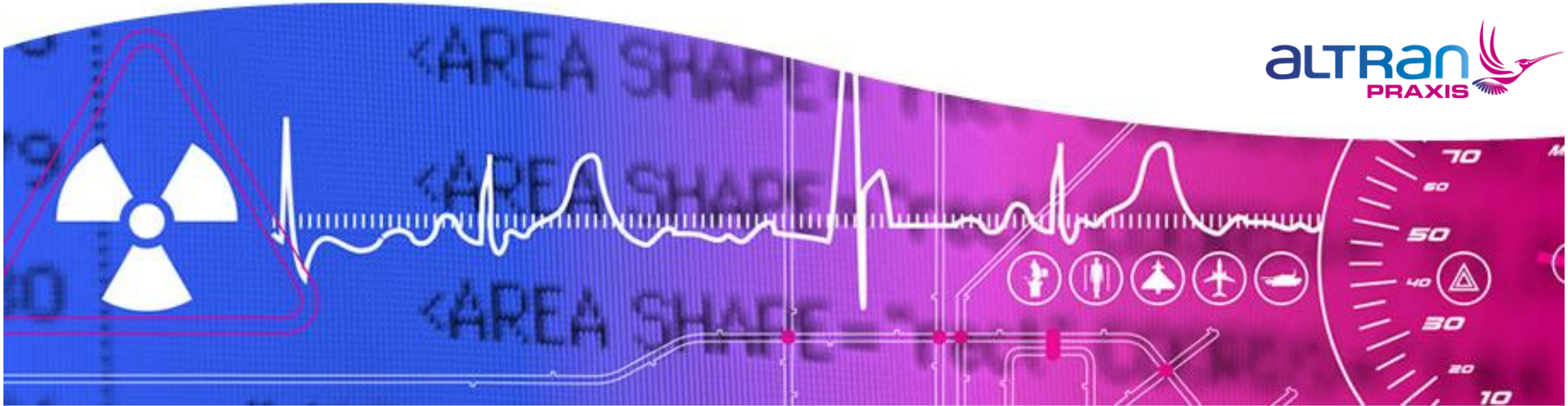
Corprate **Markets** **Services** **Technology** **News** **Careers** **Home**

Overview
Essential
Praxis
Altran

OVERVIEW

Altran Praxis delivers engineering, technology and innovation for the world's embedded and critical systems

- Our expertise and focus is on embedded and critical systems, where there are demanding safety, security and innovation requirements
- Our Engineering Services
  - Systems, Software, Safety, Innovation, Security and training
- Our Markets
  - All have embedded and critical requirements
- Our Accreditation
  - ISO9001, and also market specific accreditations
- Our Offices
  - Bath, Sophia Antipolis, Bangalore, London, Loughborough



altran PRAXIS

Corporate | Markets | Services | Technology | News | Careers | Home

- Aerospace & Defence
- Rail
- Nuclear
- Air Traffic Management
- Automotive
- Medical
- Security

MARKETS

Characteristics of Altran Praxis' markets

- Novel ideas required
- Rely on Critical systems
- Safety and Security need to de demonstrated
- Legacy systems need to be taken beyond their life expectancy
- Global markets, requiring solutions that both work locally and can be applied globally

Altran Praxis has a strong pedigree in all of these markets

- Trusted by manufacturers, suppliers, operators and regulators alike

SPARK Background

- What is SPARK?...
  - Programming language
  - Static verification toolset
- History:
  - 20+ years since first definition of language
  - Southamption Uni, PVL, Praxis, Altran Praxis
  - AdaCore – Altran Praxis Partnership
- SPARK Pro
  - Language evolution
  - IDE-integration
  - Access via GNAT Tracker portal

SPARK - The SPADE Ada Kernel

An Outline of the Language

by Bernard Carré and Trevor Jennings
Department of Electronics and Computer Science,
University of Southampton

March 1987

PARTNERSHIP

SPARK Pro – Product Update

Recent Releases – 10.0 & 10.1

- Generics – Phase 1
- Examiner dynamic tables
- Automatic selection of flow analysis level
- SPARKBridge and other improvements to proof tools
- KCG Language Profile

Generics

- Re-usable components that can be implemented and analysed once, but instantiated many times
- Phase 1 – Generic Subprograms included in Release 10.1 (Dec 2011)
- Provides access to Ada.Unchecked_Conversion
- Phase 2 – Generic Packages – scheduled for Release 11.0 (Q4 2012)
- Generics will support Ada Container Library packages in future releases

Generic Subprogram - Example

- Library-level declaration:

```
generic
   type T1 is range <>;
   type T2 is range <>;
   --# check T1 ’ Last * T1 ’ Last <= T2 ’ Last and
   --# T1 ’ First * T1 ’ First <= T2 ’ Last and
   --# T1 ’ First * T1 ’ First >= T2 ’ First ;

function Square (X : T1) return T2;
--# return R => R = T2 (X * X);
```

Generic Subprogram (cont.)

- An example instantiation with stronger function constraints:

```
type Actual_T1 is range 0 .. 10;
type Actual_T2 is range 0 .. Actual_T1'Last *
       Actual_T1'Last;

function My_Square
--# pre X > 1;
--# return R => R = T2 (X * X) and R >= 4;
is new Square (T1 => Actual_T1 , T2 => Actual_T2 );
```

Generic Package - Example

```
generic
      type T1 is private;
package Stack
--# own State : Stack_Type;
--# initializes State;
is
  --# type Stack_Type is abstract;
  --# function Is_Empty (S : Stack_Type) return Boolean;
procedure Pop (Item : out T1);
--# global in out State;
--# derives Item, State from State;
--# pre not Is_Empty (State);
…
end Stack;
```

Examiner Dynamic Tables

- Flow analyser and VCG heaps are dynamic in Release 10.1
- No more ‘megaspark’ or custom versions
- Faster Examiner start-up time

Auto-Selection of Flow Analsyis Level

- New command-line option -flow=auto
- Examiner switches automatically between data and information flow depending on presence of derives annotation
- Allows mixed analysis in single run eg:
  – Information flow at lower levels
  – Data flow only at higher levels
  – Partitions with different integrity levels & different flow analysis requirements

SPARKBridge & Proof Tools

- Simplifier tactics & efficiency continually enhanced
- SPARKBridge now provides a gateway for the use of alternative theorem provers
  - Based on Victor from Edinburgh University
  - Allows use of SAT solvers such as Alt-Ergo, CVC3, Yices, Z3
- An open-source interface to Isabelle – "SPARK/HOL" is also available

KCG Language Profile

- Language profile suitable for use with SCADE's KCG code generator
- Enables parent package to access public child
- Allows data flow errors to be delegated to proof system rather than Examiner (VC generation not yet implemented)
- Future extensions:
  - Auto-generate cut-points
  - Strengthen default loop invariant
  - Array slices

Release 11.0

- Release 11.0 scheduled for early Q4 2012
- Major features:
  - Generic Packages
  - SPARKBridge fully supported
  - Proof Functions
  - #assume annotation
  - Riposte (Beta release)

Proof Functions

- Proof functions can now be annotated with preconditions and return annotations
- They can also be refined
- We expect this change to eliminate most axiomatic Simplier user rules
- The Examiner now fully models function calls in all proof contexts ie. VC generation takes account of return annotation and pre-conditions
- Example …

Proof Function: Example

```
--# function Contains (V : Integer ;
--#                    T : Tuple )
--#                    return Boolean ;

--# function Contains_Both (A : Integer ;
--#                    B : Integer ;
--#                    T : Tuple )
--#                    return Boolean ;
--# pre A /= B;
--# return Contains (A, T) and Contains (B, T);
```

- Refined in package body …

```
--# function Contains (V : Integer ;
--#                    T : Tuple )
--#                    return Boolean ;
--# return T. First = V or T. Second = V;
```

#assume Annotation

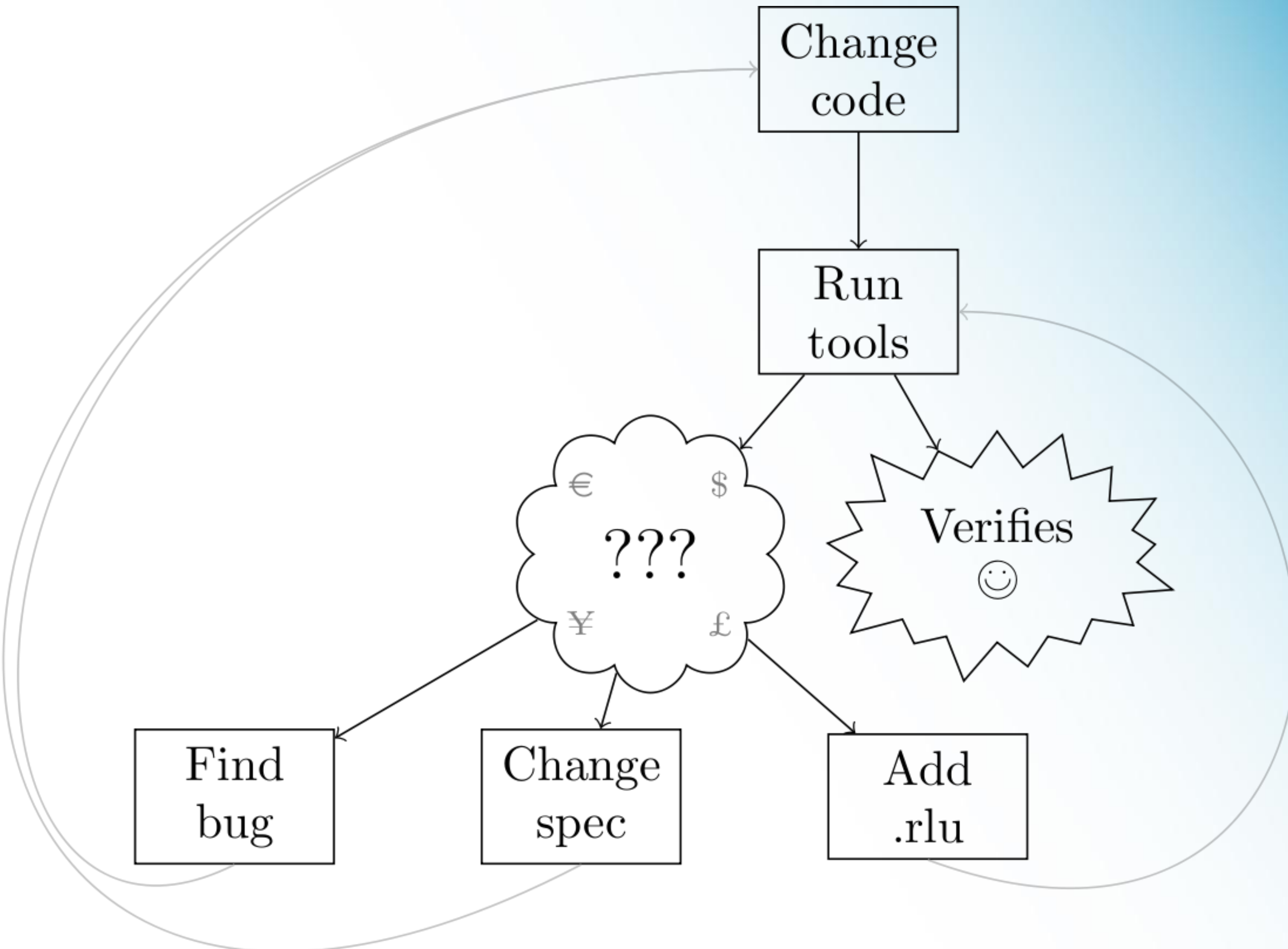
- Use to replace user rules or manual proof reviews
- Like a check statement, except that there will be no VCs generated to show that it is true.
- Example:

```
--# accept W, 444 , "We increment the uptime counter once every
second .",
--# "The operational procedure requires that the system is",
--# "rebooted at least once every 3 years - as the uptime ",
--# "is stored in a signed 64 bit integer this means the ",
--# "counter can never overflow in the lifetime of the system .";
--# assume ( Clock .T’ Last = 2**63 - 1) -> (T < Clock .T’ Last );
T := T + 1;
--# end accept;
```

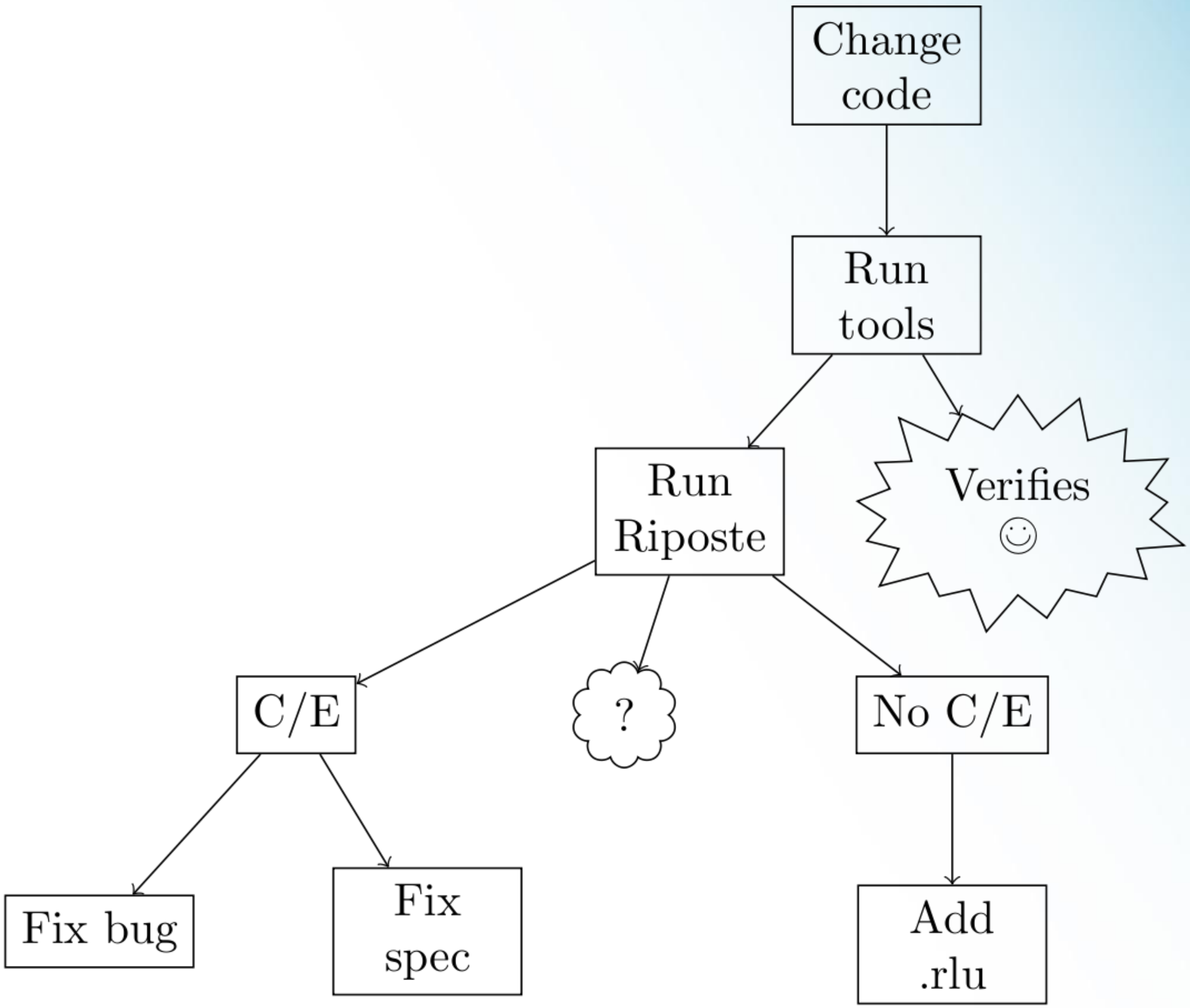
Riposte

- Counter-example generation tool
- Result of a KTP project:
  - Partnership with Bath University
  - Part-funded by the Technology Strategy Board
- Improves productivity by distinguishing false VCs from incomplete proofs
- Generates counter-examples (variable bindings) for false VCs

Current SPARK Proof Workflow



Future Workflow



Example 1 – Run Time Exceptions

- Attempt to prove absence of run time exceptions in the following function:

```
function Example_1 (X : in Integer) return Integer
--# return abs (X);
is
begin
   return abs (X);
end Example_1;
```

- Riposte finds the one case in which taking the absolute value of a 2's complement integer gives an overflow …

Example 1 – Riposte output

```
*** Found a counter-example to function_example_1_1, conclusion C2:
     (For path(s) from start to run-time check associated with statement
of line 30:)
H2: x >= integer__first
H3: x <= integer__last
   ->
C2: abs(x) <= integer__last

This conclusion is false if:
   x = integer__first
*** VC function_example_1_1 - COUNTER_EXAMPLE
*** VC function_example_1_2 - PROVEN
```

```
*** Found a counter-example to function_example_1_1, conclusion C2:
     (For path(s) from start to run-time che
of line 30:)
H2: x >= -2147483648
H3: x <= 2147483647
   ->
C2: abs(x) <= 2147483647

This conclusion is false if:
   x = -2147483648
*** VC function_example_1_1 - COUNTER_EXAMPLE
*** VC function_example_1_2 - PROVEN
```

`'riposte -n'` replaces symbolic constants with numbers

Example 2 – Functional Correctness

```
type Rec_T is record
   A : Integer;
   B : Integer;
   C : Integer;
end record;

procedure Example_2 (R : in out Rec_T)
--# derives R from R;
--# post R = R~[A => R~.C;
--#                C => R~.A]
--#  and R /= R~;
Is
   Tmp : Integer;
Begin
   Tmp := R.A;
   R.A := R.C;
   R.C := Tmp;
end Example_2;
```

Example 2 – Riposte output

```
*** Found a counter-example to procedure_example_2_4, conclusion C2:
    (For path(s) from start to finish:)
H2: fld_c(r) >= integer__first
H3: fld_c(r) <= integer__last
H4: fld_b(r) >= integer__first
H5: fld_b(r) <= integer__last
H6: fld_a(r) >= integer__first
H7: fld_a(r) <= integer__last
   ->
C2: not upf_c(upf_a(r,fld_c(r)),fld_a(r)) = r

This conclusion is false if:
   r := rec_t'(
      a => 0
      b => 0
      c => 0
   )
*** VC procedure_example_2_4 - COUNTER_EXAMPLE
```

Example 3

- Riposte can show that the post condition in the following function does not hold:

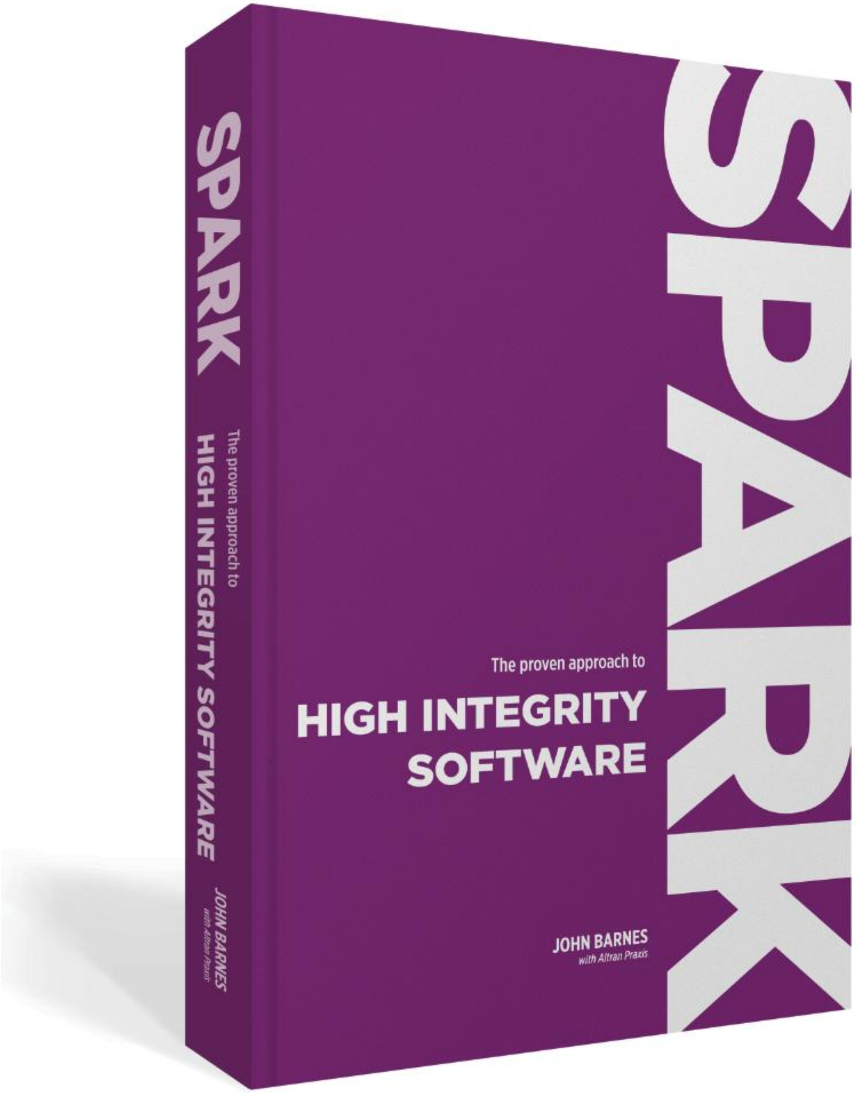
```
function Example_3 (A : in U64;
                    B : in U64)
                    return U64
--# pre A > 17 and B > 19;
--# return X => X > 0;
is
begin
  return 73 xor (A * B);
end Example_3;
```

- Riposte gives the following counter-example …

Example 3 – Riposte output

```
*** Found a counter-example to function_example_3_1, conclusion C1:
    (For path(s) from start to finish:)
H1: a > 17
H2: b > 19
H3: a >= 0
H4: a <= u64__last
H5: b >= 0
H6: b <= u64__last
   ->
C1: bit__xor(73, (a * b) mod 2 ** 64) > 0

This conclusion is false if:
   a = 15617650143032034577
   b = 17670193526088673209
*** VC function_example_3_1 - COUNTER_EXAMPLE
```



NEW BOOK COMING SUMMER 2012

SPARK is a programming language and static verification technology designed specifically for the development of high integrity software. First designed over 20 years ago, SPARK has established a track record of use in embedded and critical systems across a diverse range of industrial domains where safety and security are paramount.

This third edition of the SPARK book is a major update which reflects more recent additions to the SPARK language including tasking and generics.

From basic principles through to the use of advanced proof techniques, John Barnes provides both an informal introduction and a reference guide for those wishing to develop high integrity software using SPARK.

For more information please visit www.sparkada.com/book

SPARK Training Programme

- Primary Courses:
  - Software Engineering with SPARK
  - Advanced SPARK Program Design and Verification
- Advanced Courses:
  - Secure Software Development with SPARK *New*
  - Introduction to the Proof Checker
  - Concurrent Software Design with RavenSPARK
- Special Courses:
  - SPARK - Two Day Overview
  - Refresh Your SPARK
- Tailored Courses

Secure Software Development with SPARK

- Audience: SPARK users who wish to learn how to exploit the SPARK language and verification tools in the development of high-security software.
- Prerequisites: Software Engineering with SPARK course or fluency with the SPARK language and the Examiner.
- Training Method: A one-day course (which can follow Software Engineering with SPARK directly), combining presentations, exercises and practical work.

Course Content

- Security basics - policy, threats, and value
- System versus Software Security
- Safety and Security Properties
- SPARK in the high-grade and MILS environments
- Verification of Security Properties in SPARK
  - Ada language features supporting security
  - Information flow
  - Robustness and "crash proofing“
  - Defence against buffer overflow and other undefined behaviour
  - Validating inputs
  - Error handling and the role of defensive programming
  - Application-specific security properties
  - Comparison with CWE, SANS “Top 25”, ISO/SC22/WG23 PLV lists
- Case study: the Tokeneer ID Station

SPARK Training Delivery

- On-site training
- Public courses
  - Altran Praxis, Bath
  - September 2012
- www.altran-praxis.com/trainingSpark.aspx

Further Information

- Ada Europe Tutorials – Friday AM & PM
  - AM: The Benefits of Using SPARK for High-assurance Software
  - PM: The Use of Proof and Generics in SPARK

- The SPARK Book!

- Product information: www.adacore.com/sparkpro

Altran Praxis Limited

20 Manvers Street
Bath BA1 1PX
United Kingdom
Telephone: +44 (0) 1225 466991
Facsimile: +44 (0) 1225 469006
Website: www.altran-praxis.com

Email: stuart.matthews@altran-praxis.com