AdaCore

The Technology Basis for AvesTerra: An Overview of Ada and SPARK

Presented by Steve Baird

MSST 2018

Overview

- What is Ada?
- What is SPARK?
- Levels of Assurance and SPARK
- Industrial Experience
- Features that Matter
- Conclusions

What Is Ada?

- Ada is a high-level programming language intended for critical long-lived applications where reliability, maintainability, portability and performance are essential
- Originally designed in the early 1980s, with several subsequent revisions and enhancements
  - Current version of the language is known as Ada 2012
- Available on a wide range of native and embedded platforms
- Roughly comparable to C++ in expressive power but with many more checks, both at compile time and run time

Summary of Ada Language Features (1)

- Reliable General-Purpose Sequential Programming
  - Structured control facilities
  - Data structuring, strong typing
  - Exception handling
  - Code modules (subprograms)
  - Contract-based programming (pre- and postconditions)
- Programming in the Large
  - Packages
  - Data abstraction
  - Generic templates
  - General separate compilation facility
  - Object-Oriented Programming
    - No need for automatic garbage collection

Summary of Ada Language Features (2)

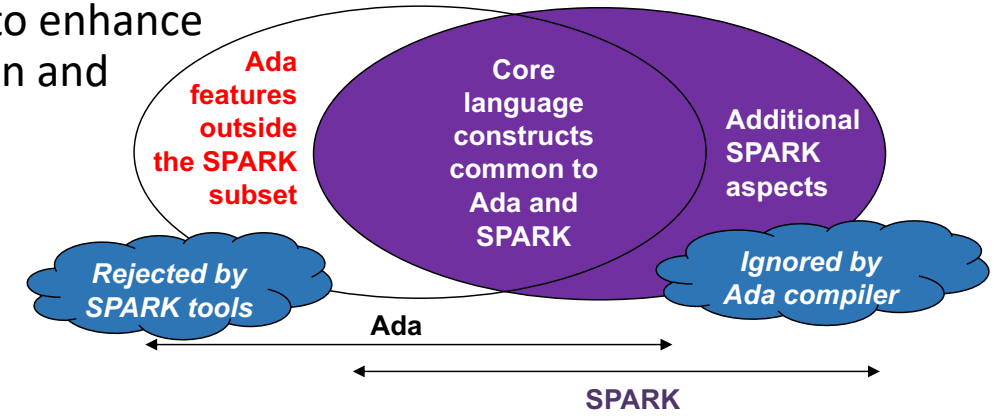
- Concurrent Programming
  - Tasking and associated statements
  - State-based mutual exclusion (protected objects/types)
  - Multicore support
  - Ravenscar profile
- Interfacing with Other Languages
  - Importing / exporting subprograms, data objects
  - Standard support for interfacing with C, C++, Fortran, COBOL
- Low-Level Programming
  - Unchecked conversions
  - Data layout control
  - Machine-dependent types

Summary of Ada Language Features (3)

- Predefined Environment
  - Input/Output
  - Character and string handling
  - Numerics
  - Execution Environment Interfacing
  - Containers
  - Internationalization
- Specialized Needs Annexes
  - Systems Programming
  - Real-Time Systems
  - Distributed Systems
  - Information Systems
  - Numerics
  - Safety/Security (High-Integrity Systems)

What Is SPARK?

- A programming language ("SPARK 2014")
  - A subset of Ada 2012 amenable to modular static verification
  - Additional features to enhance program specification and facilitate analysis



- A set of program analysis/proof tools
  - SPARK subset enforcement
  - Flow analysis
  - Functionality proof

SPARK Language Summary

- Design principles
  - Include as much of the Ada language as is possible / practical to analyze
  - Eliminate sources of ambiguity / implementation dependence
  - Add aspects that facilitate formal analysis
- Included Ada features
  - Program structure
  - Contract-based programming
    - Pre- and postconditions
    - Scalar ranges, type/subtype predicates
  - Most data types
  - Object-Oriented Programming
  - Ravenscar tasking profile
- Excluded Ada features
  - Access types (pointers)
  - Side effects in functions / expressions
  - Problematic aliasing of names
  - goto statement
  - Exception handling
  - Most tasking features

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));
```

Levels of Software Assurance

Stone Level

Strong semantic coding standard

Program respects all the SPARK language legality rules

Enforces safer use of language features:

- Restricted concurrency (Ravenscar profile)
- Expressions and functions without side-effects

Forbids language features that make analysis difficult:

- Unrestricted pointers
- Exception handlers

Bronze Level

Initialization and correct data flow

Program passes SPARK flow analysis without violations

Detects programming errors:

- Read of uninitialized data
- Problematic aliasing between parameters
- Data race between concurrent tasks

Checks user specifications:

- Data read or written
- Flow of information from inputs to outputs

Silver Level

Absence of run-time errors

Program passes SPARK proof without violations

Detects programming errors:

- Divide by zero
- Array index out of bounds
- Integer, fixed-point and floating-point overflow
- Integer, fixed-point and floating-point range violation
- Explicit exception raised
- Violation of Ceiling Priority Protocol

Gold Level

Proof of key integrity properties

Program passes SPARK proof without violations

Checks user specifications:

- Type invariants (weak and strong)
- Preconditions
- Postconditions

Checks correct use of OO wrt Liskov Substitution Principle

Platinum Level

Proof of full functional correctness

Program passes SPARK proof without violations

Checks complete user specifications:

- Type invariants (weak and strong)
- Preconditions
- Postconditions

Checks loop termination (loop variant)

Industrial Practice

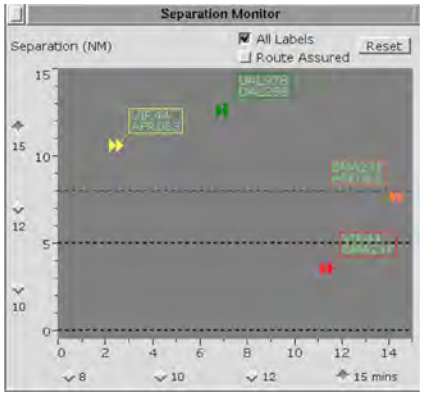
Established Practice at Altran UK

Software Integrity Level		SPARK Software Assurance Level			
DAL	**SIL**	**Bronze**	**Silver**	**Gold**	**Platinum**
A	4				
B	3				
C	2				
D	1				
E	0				

Past Projects at Altran UK

SHOLIS: 1995
DEFSTAN 00-55 SIL4
First Gold

C130J: 1996 - now
Bronze (Lockheed Martin) and Gold (UK RAF and BAE Systems)



iFACTS: 2006 - now
Silver (NATS)

Adoption Experiments at Thales

Use case 1: porting to new platform context: 300 klocs radar software target: Stone level significant manual refactoring (several days) on the way to completion on 300 klocs	Use case 2: demonstrate compliance to LLR context: small numerical function target: Gold level difficulties in expressing suitable contracts property was not proved automatically
Use case 3: identify and fix weakness context: 100s slocs code generator target: Gold level half a day to reach Silver property related to inner memory bounds two days to reach Gold	Use case 4: guarantee safety properties context: 7 klocs command & control target: Gold level one day to reach Silver property expressed as automaton four days to reach Gold

Adoption Guidelines with Thales

For every level, we present:

- Benefits, Impact on process, Costs and limitations
- Setup and tool usage
- Violation messages issued by the tool
- Remediation solutions

Guidance was put to test:

- During adoption experiments at Thales
- On the example (SPARK tool) presented in last section

Features that Matter

Stone Level – Large Language Subset

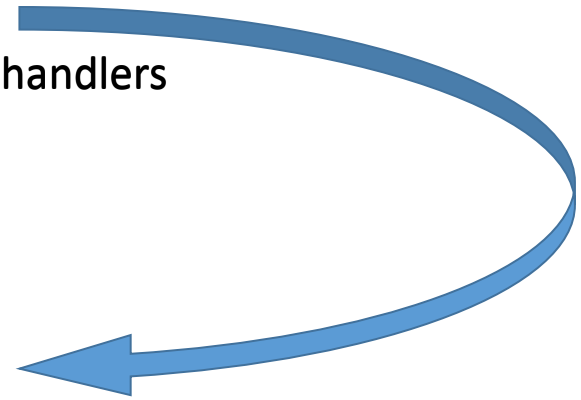
`SPARK_Mode => On`

- Ada types, expressions, statements, subprograms

`SPARK_Mode => Off`

- Ada pointers
- Ada exception handlers

- Ada generics
- Ada object orientation
- Ada concurrency
- Ada pointers



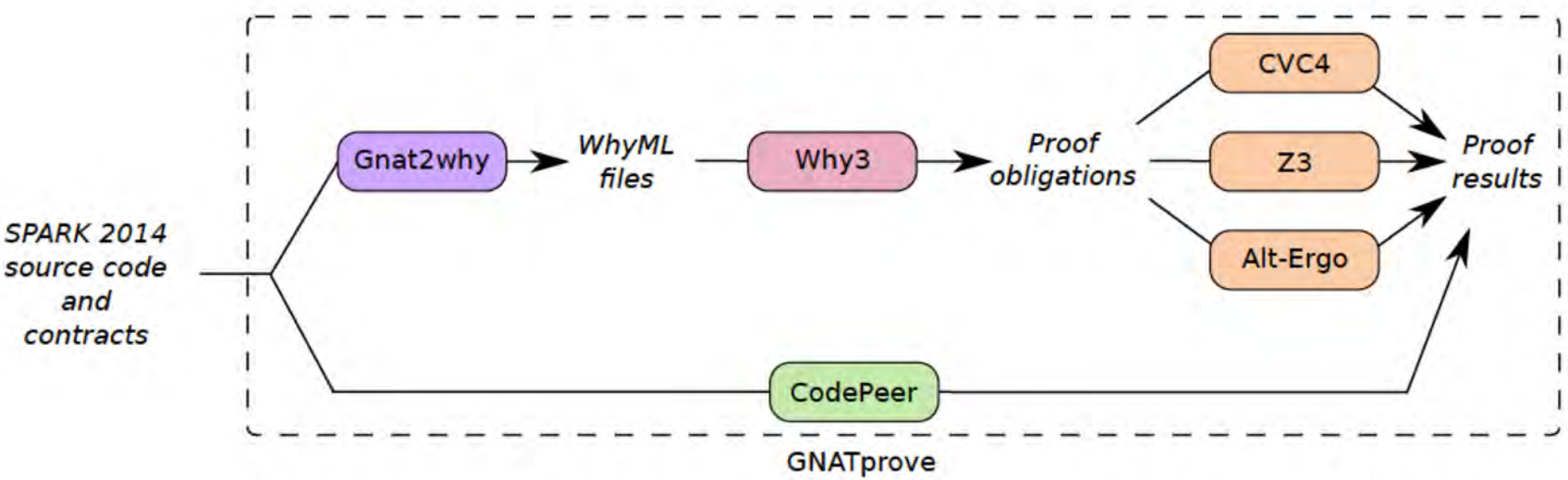
work in progress to include safe Rust-like pointers in SPARK

Bronze/Silver Level – Generation of Contracts

Example: SPARKSkein Skein cryptographic hash algorithm (Chapman, 2011)
target: Silver level

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

Silver/Gold Level – Combination of Provers



Silver/Gold Level – Combination of Provers

Example: Safe bounds on trajectory computation (submitted to VSTTE 2017)
target: Gold level

```
procedure Compute_Speed (N         : Frame;
                         Factor    : Ratio_T;
                         Old_Speed : Float64;
                         New_Speed : out Float64)
with Global => null,
     Pre    => N < Frame'Last and then
               Invariant (N, Old_Speed),
     Post   => Invariant (N + 1, New_Speed);
```

```
Delta_Speed := Drag + Factor * G * Frame_Length;
New_Speed   := Old_Speed + Delta_Speed;
```

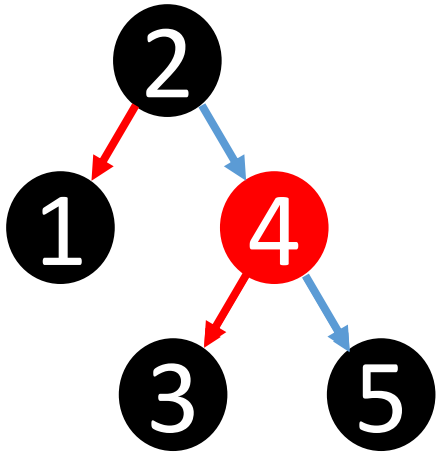
VC	CVC4	Alt-Ergo	Z3	CodePeer	AE_fpa	Colibri
Delta_Speed in -Bound .. Bound				1	3	0
In_Bounds (High_Bound(N))				1	1	
In_Bounds (Low_Bound(N))			0	1	2	
Float64(N_Bv) * Bound + Bound = (Float64(N_Bv) + 1.0) * Bound			42			0
Float64(N) * Bound + Bound = (Float64(N) + 1.0) * Bound		44		1	25	0
Float64(N) * (-Bound) Bound = (Float64(N) + 1.0) * (-Bound)				1		0
T(1) = 1.0	0	0		1	0	0
Float64(N) + 1.0 = Float64(N + 1)	0	1			1	0
New_Speed >= Float64 (N) * (-Bound) Bound	27					0
New_Speed >= Float64 (N + 1) * (-Bound)			1			0
New_Speed <= Float64 (N) * Bound + Bound	26					0
New_Speed <= Float64 (N + 1) * Bound			1			0
Post-condition	20	0			1	

Gold/Platinum Level – Auto-Active Verification

Example: Functional correctness of red-black trees (NFM 2017)
target: Platinum level

Auto-Active = portmanteau of **Auto**matic and inter**Active**

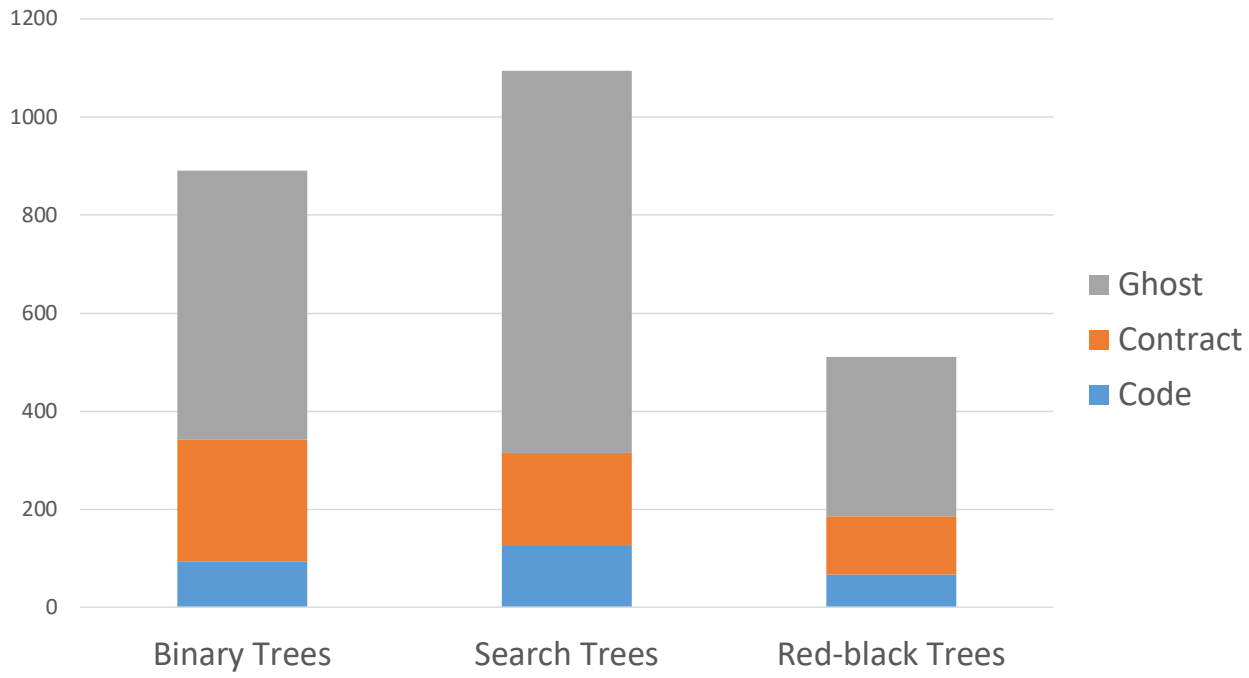
supported by **ghost** code: contracts, loop invariants, intermediate assertions, lemma procedures



ghost code used to:

- define model of data used in specifications
- prove intermediate lemmas (e.g. for inductive proofs)
- provide witness for property (e.g. for transitivity relation)

Gold/Platinum Level – Auto-Active Verification



Conclusion

Levels of Software Assurance

From strong semantic coding standard to full functional correctness

Every level implicitly builds on the lower levels

Lower levels require lower costs/efforts

Good match from DAL/SIL to Bronze-Silver-Gold-Platinum

Adoption greatly facilitated by detailed level-specific guidance

Catchy names are easy to remember!

SPARK Resources

SPARK toolset
http://www.adacore.com/sparkpro http://libre.adacore.com/

SPARK adoption guidance
www.adacore.com/knowledge/technical-papers/implementation-guidance-spark

SPARK blog and resources (User's Guide)
http://www.spark-2014.org

SPARK online training
http://u.adacore.com