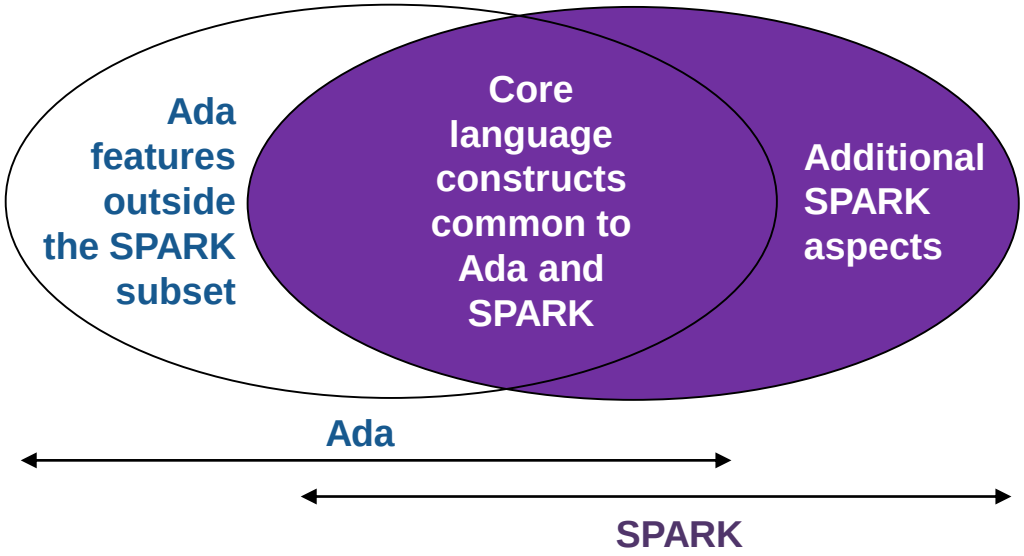
SPARK 2014: Overview

Claire Dross and Martyn Pike

University.adacore.com

SPARK 2014 – What is it ?

- **A programming language**
  - A subset of Ada, designed for static verification
  - Additional features to enhance program specification



- **A set of program verification tools**

SPARK 2014 – What do the tools do ?

- **Performs static verification of Ada source code. It can check that:**
  - a program is “well-formed”
  - it performs its intended functionality
- **Static verification can take several forms:**
  - static-semantic checking (strong typing, visibility…)
  - flow analysis (initialization of variables, data dependencies…)
  - integrity and functional verification (absence of runtime error, functional correctness…)

SPARK 2014 – Key Tools

- **GNAT compiler**
  - checks conformance of source with Ada 2012 and SPARK 2014
  - compiles source into executable
- **GNATprove**
  - performs additional SPARK 2014 checks
  - performs flow analysis
  - performs integrity and functional verification using formal proof
- **GNAT Programming Studio (GPS)**
- **GNATbench plugin of Eclipse**

SPARK 2014 – A trivial example

```
procedure Increment
          (X : in out Integer)

  with Global  => null,

       Depends => (X => X),

       Pre     => X < Integer'Last,
       Post    => X = X'Old + 1;

procedure Increment
          (X : in out Integer)
is
begin
   X := X + 1;
end Increment;
```

data dependencies ✓

flow dependencies ✓

functionality ✓

absence of runtime error ✓

SPARK 2014 – The Programming Language

- **SPARK design principles:**
  - Exclude language features difficult to specify/verify
  - Eliminate sources of ambiguity

- **Excluded from SPARK:**
  - access types and allocators
  - expressions (including function calls) with side-effects
  - aliasing of names
  - goto statements
  - controlled types
  - handling of exceptions (raise statements can be used in a program but proof will attempt to show that they cannot be executed)

SPARK 2014 – Limitations – no side-effects in expressions

- **The analysis of SPARK 2014 programs relies on the property that expressions are free from side-effects**
- **An expression is side-effect free if its evaluation does not update any object**
- **Side effects can introduce non-determinism in expression evaluation**

```
G : Integer;

function F (X : in out Integer) return Integer;

G := F (G) + F (G); --  ??
```

SPARK 2014 – Limitations – no side-effects in expressions

- **Function calls can appear in expressions, so function calls must be free from side-effects**

- **The objects updated by a function call are any parameters of mode out (or in out), plus global variables updated by the function body**

```
function F (X : in out Integer) return Integer; -- Illegal

function Incr (X : Integer) return Integer;  -- OK?

function Incr_And_Log (X : Integer) return Integer;  -- OK?
```

SPARK 2014 – Limitations – no side-effects in expressions

- **This is OK...**

```
function Incr (X : Integer) return Integer;  -- OK?

function Incr_And_Log (X : Integer) return Integer;  -- OK?
```

- **... provided that the analysis of the bodies doesn't detect any side-effects**

```
function Incr (X : in Integer) return Integer
is (X + 1); -- OK

Call_Count : Natural := 0;

function Incr_And_Log (X : in Integer) return Integer is
begin
  Call_Count := Call_Count + 1; -- Illegal
  return X + 1;
end Incr_And_Log;
```

SPARK 2014 – Limitations – no aliasing of names

- **Aliasing occurs when two names refer to the same object**
- **SPARK forbids aliasing of outputs of subprograms (parameters of mode out or in out and global variables modified by the subprogram)**
- **This can make the subprogram give different results depending on parameter passing mechanism (by reference or by value)**
- **Results can seem unexpected, even for the user**

SPARK 2014 – Limitations – no aliasing of names

```
Total : Natural := 0;

procedure Move_To_Total (Source : in out Natural) is
begin
  Total  := Total + Source;
  Source := 0;
end Move_To_Total;
```

- **The above subprogram is OK**
- **Absence of aliasing will be verified whenever a call is made:**

```
X : Natural := 3;

Move_To_Total (X); -- OK
Move_To_Total (Total); -- Error
```

SPARK 2014 – Identifying SPARK Code

- **SPARK and full Ada code can be mixed**
  - To verify as much as possible of a code base
  - While retaining usage of excluded features when necessary
- **SPARK_Mode (aspect or pragma) specifies whether code is intended to be SPARK**
  - It is then checked by the GNATprove tool
- **Defaults to 'Off', should be added to units to be analyzed**
  - Or can be set globally using a configuration pragma
  - SPARK_Mode on 'withed' units defaults to 'auto' (up to the tool to determine whether or not a given construct is in SPARK)

SPARK 2014 – Identifying SPARK Code

```
package P
  with SPARK_Mode => On
is
   -- package spec is SPARK, so can be used
   -- by SPARK clients
end P;


package body P
  with SPARK_Mode => Off
is
   -- body is NOT SPARK, so assumed to
   -- be full Ada
end P;
```

SPARK 2014 – Identifying SPARK Code

- **SPARK_Mode can be 'On' or 'Off' for**
  - visible part of package specification
  - private part of package specification
  - declarative part of package body
  - statement part of package body
  - subprogram specification
  - subprogram body

- **If SPARK mode is 'Off' for a package or subprogram then it cannot be switched 'On' for a later part of that package or subprogram**

Quiz

Is this correct? 1/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package Stack_Package
  with SPARK_Mode => On
is
  type Element is new Natural;
  type Stack is private;

  function Empty return Stack;
  procedure Push (S : in out Stack; E : Element);
  function Pop (S : in out Stack) return Element;

private
  ...
end Stack_Package;
```

Is this correct? 1/10

NO

```
package Stack_Package
  with SPARK_Mode => On
is
  type Element is new Natural;
  type Stack is private;

  function Empty return Stack;
  procedure Push (S : in out Stack; E : Element);
  function Pop (S : in out Stack) return Element;

private
  ...
end Stack_Package;
```

Function with "in out" parameter is not allowed in SPARK

Is this correct? 2/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package body Global_Stack
  with SPARK_Mode => On
is
  Max : constant Natural := 100;
  type Element_Array is array (1 .. Max) of Element;

  Content : Element_Array;
  Top     : Natural;

  function Pop return Element is
    E : constant Element := Content (Top);
  begin
    Top := Top - 1;
    return E;
  end Pop;

end Global_Stack;
```

Is this correct? 2/10

NO

```
package body Global_Stack
  with SPARK_Mode => On
is
  Max : constant Natural := 100;
  type Element_Array is array (1 .. Max) of Element;

  Content : Element_Array;
  Top     : Natural;

  function Pop return Element is
    E : constant Element := Content (Top);
  begin
    Top := Top - 1;
    return E;
  end Pop;

end Global_Stack;
```

Function updating a global variable is not allowed in SPARK

Is this correct? 3/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package body P
  with SPARK_Mode => On
is
  procedure Permute (X, Y, Z : in out Positive) is
    Tmp : constant Positive := X;
  begin
    X := Y;
    Y := Z;
    Z := Tmp;
  end Permute;

  procedure Swap (X, Y : in out Positive) is
  begin
    Permute (X, Y, Y);
  end Swap;
end P;
```

Is this correct? 3/10

NO

```
package body P
  with SPARK_Mode => On
is
  procedure Permute (X, Y, Z : in out Positive) is
    Tmp : constant Positive := X;
  begin
    X := Y;
    Y := Z;
    Z := Tmp;
  end Permute;

  procedure Swap (X, Y : in out Positive) is
  begin
    Permute (X, Y, Y);
  end Swap;
end P;
```

Aliasing between "in out" parameters is not allowed in SPARK

Is this correct? 4/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package body P
  with SPARK_Mode => On
is
  procedure Swap (X, Y : in out Positive);

  type Rec is record
    F1 : Positive;
    F2 : Positive;
  end record;

  procedure Swap_Fields (R : in out Rec) is
  begin
    Swap (R.F1, R.F2);
  end Swap_Fields;

  ...
end P;
```

Is this correct? 4/10


```
package body P
  with SPARK_Mode => On
is
  procedure Swap (X, Y : in out Positive);

  type Rec is record
    F1 : Positive;
    F2 : Positive;
  end record;

  procedure Swap_Fields (R : in out Rec) is
  begin
    Swap (R.F1, R.F2);
  end Swap_Fields;

  ...
end P;
```

Two different fields of the same record always refer to distinct objects.

Is this correct? 5/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package body P
  with SPARK_Mode => On
is
  procedure Swap (X, Y : in out Positive);

  type P_Array is array (Natural range <>) of Positive;

  procedure Swap_Indexes (A : in out P_Array, I, J : Natural) is
  begin
    Swap (P (I), P (J));
  end Swap_Indexes;

  ...
end P;
```

Is this correct? 5/10

NO

```
package body P
  with SPARK_Mode => On
is
  procedure Swap (X, Y : in out Positive);

  type P_Array is array (Natural range <>) of Positive;

  procedure Swap_Indexes (A : in out P_Array, I, J : Natural) is
  begin
     Swap (A (I), A (J));
  end Swap_Indexes;

  ...
end P;
```

If I = J, the two components of the array can be aliased.

GNATprove will detect this possibility

Is this correct? 6/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P
  with SPARK_Mode => On
is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is access String;
  type Dictionary is array (Letter) of String_Access;

  procedure Store (D : in out Dictionary; W : String);
end P;

package body P
  with SPARK_Mode => On
is
  procedure Store (D : in out Dictionary; W : String) is
    First_Letter : constant Letter := W (W'First);
  begin
    D (First_Letter) := new String'(W);
  end Store;
end P;
```

Is this correct? 6/10

NO

```
package P
  with SPARK_Mode => On
is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is access String;
  type Dictionary is array (Letter) of String_Access;

  procedure Store (D : in out Dictionary; W : String);
end P;

package body P
  with SPARK_Mode => On
is
  procedure Store (D : in out Dictionary; W : String) is
    First_Letter : constant Letter := W (W'First);
  begin
    D (First_Letter) := new String'(W);
  end Store;
end P;
```

Access types are not allowed in SPARK

Is this correct? 7/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P
  with SPARK_Mode => On
is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is private;
  type Dictionary is array (Letter) of String_Access;

  function New_String_Access (W : String) return String_Access;

  procedure Store (D : in out Dictionary; W : String);

private
  pragma SPARK_Mode (Off);

  type String_Access is access String;

  function New_String_Access (W : String) return String_Access is
    (new String'(W));
end P;
```

Is this correct? 7/10


```
package P
  with SPARK_Mode => On
is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is private;
  type Dictionary is array (Letter) of String_Access;

  function New_String_Access (W : String) return String_Access;

  procedure Store (D : in out Dictionary; W : String);

private
  pragma SPARK_Mode (Off);

  type String_Access is access String;

  function New_String_Access (W : String) return String_Access is
    (new String'(W));
end P;
```

SPARK_Mode has been turned off in the private section

The type String_Access is declared in full Ada.

Is this correct? 8/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P with SPARK_Mode => On is
   subtype Letter is Character range 'a' .. 'z';
   type String_Access is private;
   type Dictionary is array (Letter) of String_Access;
   function New_String_Access (W : String) return String_Access;
   procedure Store (D : in out Dictionary; W : String);

private
   pragma SPARK_Mode (Off);
   …
end P;

package body P with SPARK_Mode => On is
   procedure Store (D : in out Dictionary; W : String) is
      First_Letter : constant Letter := W (W'First);
   begin
      D (First_Letter) := New_String_Access (W);
   end Store;
end P;
```

Is this correct? 8/10

NO

```
package P with SPARK_Mode => On is
   subtype Letter is Character range 'a' .. 'z';
   type String_Access is private;
   type Dictionary is array (Letter) of String_Access;
   function New_String_Access (W : String) return String_Access;
   procedure Store (D : in out Dictionary; W : String);

private
   pragma SPARK_Mode (Off);
   …
end P;

package body P with SPARK_Mode => On is
   procedure Store (D : in out Dictionary; W : String) is
      First_Letter : constant Letter := W (W'First);
   begin
      D (First_Letter) := New_String_Access (W);
   end Store;
end P;
```

SPARK mode is 'Off' for P's private part. It cannot be switched back 'On' for P's body.

Is this correct? 9/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package P with SPARK_Mode => On is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is private;
  type Dictionary is array (Letter) of String_Access;
  function New_String_Access (W : String) return String_Access;
private
  pragma SPARK_Mode (Off);
  …
end P;

with P; use P;
package Q with SPARK_Mode => On is
  procedure Store (D : in out Dictionary; W : String);
end Q;

package body Q with SPARK_Mode => On is
  procedure Store (D : in out Dictionary; W : String)  is
    First_Letter : constant Letter := W (W'First);
  begin
    D (First_Letter) := New_String_Access (W);
  end Store;
end Q;
```

Is this correct? 9/10


```
package P with SPARK_Mode => On is
  subtype Letter is Character range 'a' .. 'z';
  type String_Access is private;
  type Dictionary is array (Letter) of String_Access;
  function New_String_Access (W : String) return String_Access;
private
  pragma SPARK_Mode (Off);
  …
end P;

with P; use P;
package Q with SPARK_Mode => On is
  procedure Store (D : in out Dictionary; W : String);
end Q;

package body Q with SPARK_Mode => On is
  procedure Store (D : in out Dictionary; W : String)  is
    First_Letter : constant Letter := W (W'First);
  begin
    D (First_Letter) := New_String_Access (W);
  end Store;
end Q;
```

Is this correct? 10/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
package body P with SPARK_Mode => On is
   type N_Array is array (Positive range <>) of Natural;
   Not_Found : exception;

   function Search_Zero_P (A : N_Array) return Positive is
   begin
      for I in A'Range loop
         if A (I) = 0 then
            return I;
         end if;
      end loop;
      raise Not_Found;
   end Search_Zero_P;

   function Search_Zero_N (A : N_Array) return Natural
      with SPARK_Mode => Off is
   begin
      return Search_Zero_P (A);
   exception
      when Not_Found => return 0;
   end Search_Zero_N;
end P;
```

Is this correct? 10/10


```
package body P with SPARK_Mode => On is
   type N_Array is array (Positive range <>) of Natural;
   Not_Found : exception;

   function Search_Zero_P (A : N_Array) return Positive is
   begin
      for I in A'Range loop
         if A (I) = 0 then
            return I;
         end if;
      end loop;
      raise Not_Found;
   end Search_Zero_P;

   function Search_Zero_N (A : N_Array) return Natural
      with SPARK_Mode => Off is
   begin
      return Search_Zero_P (A);
   exception
      when Not_Found => return 0;
   end Search_Zero_N;
end P;
```

Raising an exception is allowed. GNATprove will try to demonstrate that it will never happen at runtime. Handlers must be in full Ada parts though.

university.adacore.com