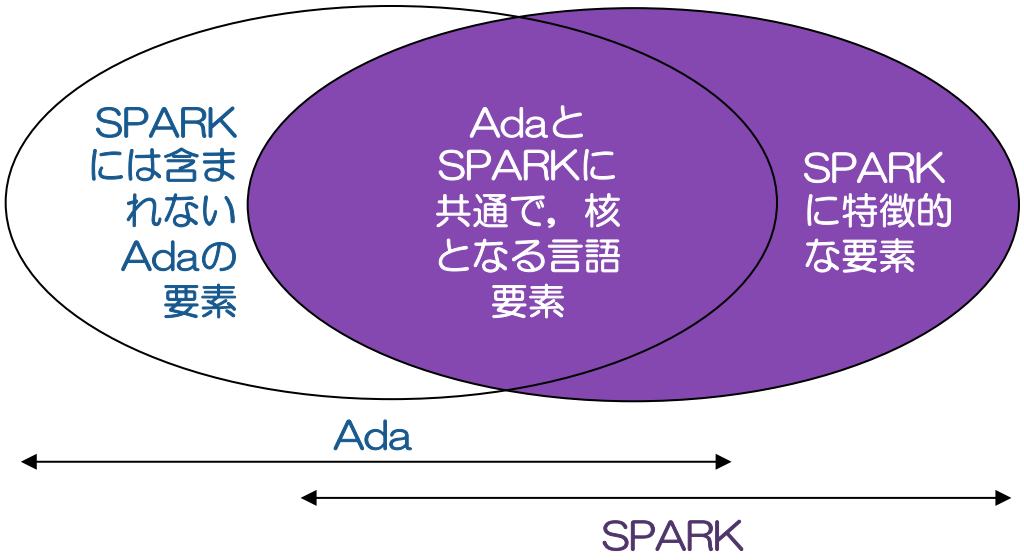
SPARK 2014: Overview

Claire Dross and Martyn Pike

University.adacore.com

SPARK 2014 – それは何か？

- プログラム言語である
  - Ada言語のサブセットである．静的検証を目的としている
  - プログラム仕様を向上させるための特徴を持つ



- プログラム検証ツール群である

SPARK 2014 – ツールは何ができるか？

- Adaソースコードに対して静的検証を実行する
  - プログラムは，「斉形式（well-formed）」であるか
  - 意図したとおりの機能を実行するか

- 静的検証には幾つかの形式がある：
  - 静的意味検査（強い型付け，可視性など）
  - フロー分析（変数の初期化，データ依存性など）
  - 完全性と機能検証（実行時エラーがないこと，機能の正しさなど）

SPARK 2014 – 主要なツール

- GNAT compiler
  - プログラムが，Ada 2012 とSPARK 2014に適合しているかを検査する
  - プログラムをコンパイルし，実行イメージを作る
- GNATprove
  - SPARK 2014の検査を強化する
  - フロー分析を実行する
  - 形式的証明を用いて完全性と機能検証を実行する
- GNAT Programming Studio (GPS)
- GNATbench plugin of Eclipse

SPARK 2014 – 小さな例題

```
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;
```

データ依存 ✓

フロー依存 ✓

機能 ✓

実行時エラーがないこと ✓

SPARK 2014 – プログラム言語

- SPARK 設計原則:
  - 正しく記述することや検証することが困難な言語としての特徴を排除した
  - 曖昧さを生む可能性のある箇所を取り除いた

- SPARKから取り除かれた言語要素:
  - アクセス型と割当て演算子
  - 副作用のある式（関数呼び出しを含む）
  - 別名化
  - goto 文
  - 被制御型（controlled types）
  - 例外処理（例外送出文を用いることはできる．しかし，実行しないことを証明しようとするに過ぎない）

SPARK 2014 – 制限– 副作用のない式

- プログラムは副作用を持たないというプロパティを前提として，SPARK 2014 は分析を行う
- 式の評価によって，オブジェクトが更新されないならば，その式は副作用がないといえる
- 副作用があると，式の評価を行うときに，非決定性（non-determinism）の考慮が必要になる

```
G : Integer;

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

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

SPARK 2014 –制限– 副作用のない式

- 式中で，関数呼び出しを行うことができる．関数呼び出しで副作用を生じてはいけない

- 関数呼び出しによって更新されるオブジェクトは，アウトモード（out）ないしは，インアウト（in out）モードの引数である．或いは，関数の本体部によって更新される広域変数である

```
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 –制限– 副作用のない式

- 以下は問題ないが…

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

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

- 実際には，本体部の分析によって，副作用があるかないかを確認しなくてはならない

```
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 – 制限– 別名化の禁止

- 別名化というのは，二つの名前が，同一のオブジェクトを指すことである
- SPARKは，サブプログラム出力の別名化を禁止している（out モードないしは in out モードの引数，およびサブプログラムによって更新された広域変数）
- 引数の受け渡し機構（参照渡し，値渡し）に依存して，結果が異なるためである
- 結果は，ユーザにとってさえ期待通りにならないことがあり得る

SPARK 2014 – 制限– 別名化の禁止

```
Total : Natural := 0;

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

- 上記の手続きは問題ない

- 呼び出しの都度，別名化がないことを検査する

```
X : Natural := 3;

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

SPARK 2014 – SPARKコードを特定する

- SPARKと純粋なAdaコードを一緒に記述できる
  - プログラム全体（code base）を可能な限り検証する
  - 要すれば，（SPARKで）除外した特性を維持する
- SPARK_Mode(アスペクトないしは pragma）によって，SPARKコードであることを指示できる
  - これにより，GNATprove ツールによって検査できる
- 解析対象コードに対して，適切に‘Off’を設定すべきである
  - 或いは，構成 pragma を用いて，広域的に設定する
  - With節によって呼ばれたユニットの SPARK_Mode は，いわば「自動」モードとなる（記述が SPARK であるかどうかはツールの判断に依存する）

SPARK 2014 - SPARKコードを特定する

```
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 – SPARKコードを特定する

- 以下に関して，SPARK_Mode は ‘On’ か ‘Off’ である
  - パッケージ仕様部の公開部分
  - パッケージ仕様部の非公開部分
  - パッケージ本体部の宣言部
  - パッケージ本体部の実行部
  - サブプログラム仕様部
  - サブプログラム本体部

- パッケージないしはサブプログラムで，SPARKモードが ’Off’ であれば，そのパッケージないしはサブプログラムの残りの部分で，’ON’にすることはできない

Quiz

正しいですか　1/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか 1/10

いいえ

```
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;
```

SPARKでは，in out モードの引数を持つ関数を利用できません

正しいですか　2/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか 2/10

いいえ

```
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;
```

SPARKでは，広域変数を更新する関数は，記述できない

正しいですか　3/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか 3/10

いいえ

```
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;
```

in out パラメータ間の別名化は，
SPARKでは許可されない

正しいですか　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;
```

正しいですか　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;
```

同じレコードの異なる要素は，常に異なるオブジェクトを参照しています

正しいですか　5/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか　5/10

いいえ

```
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;
```

もし，I が J と等しいと，2つの配列要素は別名化していることになります．

GNATprove は，この可能性を検出します．

正しいですか 6/10


```
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;
```

正しいですか 6/10

いいえ

```
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;
```

SPARKでは，アクセス型を使用できない

正しいですか　7/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか　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;
```

非公開部分で，SPARK_Modeは，オフになっています．

String_Access 型を，Ada言語部分で宣言しています．

正しいですか　8/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか　8/10

いいえ

```
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;
```

Pの非公開部分では，SPARKモードはオフです．Pの本体部でオンに戻すことはできません．

正しいですか　9/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか　9/10

はい

```
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;
```

正しいですか　10/10

はい
（チェックアイコンをクリックする）

いいえ
（エラーの場所をクリックする）

```
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;
```

正しいですか　10/10

はい

```
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;
```

例外を発行することができます．実行時にその例外が発行されないことを
GNATproveは示そうと試みます．
例外発行後に，例外を扱う部分は，もちろんAda言語で書く必要があります

university.adacore.com