

HI-LITE



HI-LITE Progress on Praxis Work Packages

- An experimental version of Victor bundled with Alt-Ergo has been released available to customers with the Linux version of the SPARK Toolset since November 2010 and on Windows since March 2011. It is being used extensively by us on proving parts of the SPARK Toolset (SPARK) code.
- Generics are the main topic of this presentation.
- SPARK container library is discussed at the end of the presentation.
- Package invariants are scheduled for a future release.

- SPARK Generics LRM and User View Documents written.
- SPARK generics are Included in the new version of “High Integrity Software: The SPARK Approach to Safety and Security” by John Barnes to be published in Q2 2012.
- SPARK generics excludes those features of Ada which are not allowed in SPARK but leaves a powerful subset.
- Supports the concept of *prove once use many times*.

- Library–level generic subprograms and packages may be declared.
- A generic declaration may also appear in the visible part of a library-level generic package declaration.
- A generic subprogram instantiation may appear anywhere a subprogram could be legally be declared in SPARK and similarly for a generic package instantiation.
- Additionally a generic package instantiation may appear in the private part of a package.
- A generic declaration may have an instantiation check .
- A generic subprogram declaration and its instantiation may have pre and post/return annotations.

- The following generic formal type parameters are supported:
 - Formal private and limited private types
 - Formal discrete type
 - Formal signed integer type
 - Formal modular type
 - Formal floating and fixed point types
 - Formal constrained and unconstrained array types

- Formal generic object parameters are supported but they must be of mode **in** and instantiated with a SPARK constant expression.
- Formal generic subprogram parameters are supported but they cannot refer to global variables.

HI-LITE Generic subprogram declaration - example

generic

type T1 is range <>;

type T2 is range <>;

--# check T1'Last * T1'Last <= T2'Last and

--# T1'First * T1'First <= T2'Last;

function Square (X : T1) return T2;

--# return T2 (X * X);

HI-LITE Generic subprogram instantiation - example

```
type Actual_T1 is range 0 .. 10
```

```
type Actual_T2 is range 0 .. Actual_T1'Last ** 2;
```

```
function My_Square
```

```
--# pre X > 1;
```

```
--# return R => R = T2 (X * X) and R >= 4;
```

```
is new Square (T1 => Actual_T1, T2 => Actual_T2);
```

HI-LITE Generic package declaration - 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;

- Instantiation check is declared on the generic but checked at the point of instantiation by generating appropriate VCs.
- If the instantiation does not have its own pre or post conditions then the corresponding pre or post condition from the generic declaration applies.
- If the instantiation has its own pre or post conditions, then VCs are generated as appropriate to prove:
 - ***pre_instantiation*** \Rightarrow ***pre_generic***
 - ***pre_instantiation*** and ***post_generic*** \Rightarrow ***post_instantiation***
- Properties proven for a body of a generic hold for all instantiations of a generic provided the instantiation check is satisfied.

HI-LITE Unchecked conversion - a special case

generic

type Source is private;

type Target is private;

function Unchecked_Conversion (S : Source) **return** Target;

-- pre and return annotations assumed to be True

type Byte **is mod** 256;

type Status **is** (S0, S1, S2, S3, S4);

subtype Running **is** Status **range** S1 .. S4;

function Byte_To_Status

--# **pre** S > 0 **and** S <= 4;

--# **return** R => R **in** Running; -- Generates a warning

is new Unchecked_Conversion

(Source => Byte, Target => Status);

HI-LITE SPARK Generics: current status

- The inclusion of generics has required a major update of the Examiner with much refactoring but with many more self-consistency checks and proofs.
- Generic subprogram declarations and instantiations now supported in next release of SPARK Pro Release 10.1 (due December 2011).
- Instantiation checks not yet implemented nor proof of consistency between instantiation and generic pre and post conditions – these require VC generation from declarations.
- The implementation of generic bodies is not complete but can be used as ‘beta’ release. Currently, a warning is generated.
- Generic bodies and packages will be available during 2012.

- SPARK.Ada.Strings.Unbounded, SPARK.Ada.Command_Line, SPARK.Ada.Text_IO and Interfaces library components completed and shipped.
- Libraries for secure applications including SPARK_Skein and SPARKLib_Crypto are available.
- A draft specification of the Container Library has been written but it needs updating following analysis by AdaCore.
- The SPARK subset of generics has been designed to facilitate SPARK interfaces to the Ada container library, in particular allowing the declaration of nested generic packages.

Simple SPARK interface to Ada.Containers.Vectors (1)

```
with Ada.Containers, Ada.Containers.Vectors,  
      SPARK.Ada.Containers;  
--# inherit SPARK.Ada.Containers;  
generic  
  type Item is private;  
package Unbounded_Vector is  
  type Index is new SPARK.Ada.Containers.Count_Type;  
  type Vector is private;  
  function Last_Index (V : Vector) return Index;  
  procedure Initialize (V : out Vector);  
--# post Last_Index (V) = 0;
```

Simple SPARK interface to Ada.Containers.Vectors (2)

```
function Get_Element (V : Vector; I : Index) return Item;
```

```
--# pre I <= Last_Index (V);
```

```
procedure Set_Element (V : in out Vector;
```

```
                  I : in Index;
```

```
                  Value : in Item);
```

```
--# pre I <= Last_Index (V);
```

```
--# post Get_Element (V, I) = Value;
```

```
-- Appends Value to V, extending the capacity of V
```

```
procedure Append (V : in out Vector;
```

```
                  Value : in Item);
```

```
--# post Last_Index (V) = Last_Index (V~) + 1 and
```

```
--#       Get_Element (V, Last_Index (V)) = Value;
```

Simple SPARK interface to Ada.Containers.Vectors (3)

private

--# hide Unbounded_Vector

package Vectors **is new** Ada.Containers.Vectors

(Index_Type => Index,
Element_Type => Item);

type Vector **is new** Vectors.Vector;

end Unbounded_Vector;

HI-LITE Simple SPARK bounded set type (1)

```
with SPARK.Ada.Containers;
```

```
--# inherit SPARK.Ada.Containers;
```

```
generic
```

```
  type Item is private;
```

```
  Max_Elements : in Spark.Ada.Containers.Count_Type;
```

```
--# check Max_Elements >= 1;
```

```
package Indexed_Set is
```

```
  type Set is limited private;
```

```
  type El_Ref is limited private;
```

```
  function Is_Element_Of (S : Set; E : El_Ref) return Boolean;
```

```
  procedure Initialize (S : out Set);
```

HI-LITE Simple SPARK bounded set type (2)

```
function Get_Element (S : Set; E: El_Ref) return Item;  
--# pre Is_Element_Of (S, E);
```

```
procedure Add_Element (S      : in out Set;  
                      Value : in      Item;  
                      E      :      out El_Ref);  
--# post Is_Element_Of (S, E) and  
--#      Get_Element (S, E) = Value;
```

HI-LITE Simple SPARK bounded set type (3)

private

subtype Index **is** SPARK.Ada.Containers.Count_Type
range 1 .. Max_Elements;

type Mapping **is array** (Index) **of** Index;

type Set **is array** (Index) **of** Item;

type El_ref **is record**

 I : Index;

end record;

end Indexed_Set;

HI-LITE

Thank you.

trevor.jennings@altran-praxis.com

stuart.matthews@altran-praxis.com

HI-LITE People

