Type Contracts

University.adacore.com

Type Contracts in Ada 2012 and SPARK 2014

Natural evolution in Ada from previous type constraints

- Scalar range specifies lower and upper bounds
- Record discriminant specifies variants of the same type

Executable type invariants by Meyer in Eiffel (1988)

- Part of Design by Contract ™
- Type invariant is checked dynamically when an object is created, and when an exported routine of the class returns

Ada 2012 / SPARK 2014 support *strong* and *weak* invariants

- A strong invariant must hold all the time
- A weak invariant must hold outside of the scope of the type

Static Predicate

Original use case for type predicates in Ada 2012

- Supporting non-contiguous subtypes of enumerations
- Removes the constraint to define enumeration values in an order that allows defining interesting subtypes

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Day with
  Static_Predicate => Day_Off in Wednesday | Weekend;
```

Typical use case on scalar types for holes in range

- e.g. floats without 0.0

Types with static predicate are restricted

- Cannot be used for the index of a loop (but OK for array index / value in case statement)

Dynamic Predicate

Extension of static predicate for any property

- Property for static predicate must compare value to static expressions
- Property for dynamic predicate can be anything

```
subtype Day_Off is Day with
  Dynamic_Predicate => Check_Is_Off_In_Calendar (Day_Off);
```

Various typical use cases on scalar and composite types

- Strings that start at index 1 (My_String'First = 1)
- Upper bound on record component that depends on the discriminant value (Length <= Capacity)
- Ordering property on array values (Is_Sorted (My_Array))

Restrictions on Types With Dynamic Predicate

Types with dynamic predicate are restricted

- Cannot be used for the index of a loop (same as static predicate)
- Cannot be used as array index
- Cannot be used for the value in a case statement

No restriction on the property in Ada

- Property can read the value of global variable (e.g. Check_Is_Off_In_Calendar) → what if global variable is updated?
- Property can even have side-effects!

Stronger restrictions on the property in SPARK

- Property cannot read global variables or have side-effects
- These restrictions make it possible to prove predicates

Dynamic Checking of Predicates

Similar to other type constraints

- Checked everywhere a range/discriminant check would be issued: assignment, parameter passing, type conversion, type qualification
- Exception Constraint_Error is raised in case of violation
- Static predicate checked by default, dynamic one only with -gnata

Static predicate does not mean verification at compile time!

```
My_Day_Off : Day_Off := This_Day;
```

Property should not contain calls to functions of the type

- These functions will check the predicate on entry, leading to an infinite loop
- GNAT compiler warns about such cases

Temporary Violations of the Dynamic Predicate

Sometimes convenient to locally violate the property

- Inside subprogram, to assign components of a record without an aggregate assignment

Idiom is to define two types

- First type does not have a predicate
- Second type is a subtype of the first with the predicate
- Conversions between these types at subprogram boundary

```
type Raw_Week_Schedule is record
  Day_Off, Day_On_Duty : Day;
end record;

type Week_Schedule is Raw_Week_Schedule with
  Dynamic_Predicate => Day_Off /= Day_On_Duty;
```

Type Invariant

Corresponds to the *weak* version of invariants

- Predicates should hold always (only enforced with SPARK proof)
- Type invariants should only hold outside of their defining package

Type invariant can only be used on private types

- Either on the private declaration
- Or on the completion of the type in the private part of the package (makes more sense in general, only option in SPARK)

```
   type Week_Schedule is private;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record with
      Type_Invariant => Day_Off /= Day_On_Duty;
```

Dynamic Checking of Type Invariants

Checked on outputs of public subprograms of the package

- Checked on results of public functions
- Checked on (in) out parameters of public subprograms
- Checked on variables of the type, or having a part of the type
- Exception Constraint_Error is raised in case of violation
- Not checked by default, activated with -gnata

No checking on internal subprograms!

```
procedure Internal_Adjust (WS : Week_Schedule) is
begin
   WS.Day_Off := WS.Day_On_Duty;
end Internal_Adjust;
```

- Choice between predicate and type invariants depends on the need for such internal subprograms without checking

Inheritance of Predicates and Type Invariants

Derived types inherit the predicates of their parent type

- Similar to other type constraints like bounds
- Allows to structure a hierarchy of subtypes, from least to most constrained

```
subtype String_Start_At_1 is String with
  Dynamic_Predicate => String_Start_At_1'First = 1;
subtype String_Normalized is String_Start_At_1 with
  Dynamic_Predicate => String_Normalized'Last >= 0;
subtype String_Not_Empty is String_Normalized with
  Dynamic_Predicate => String_Not_Empty'Length >= 1;
```

Type invariants are typically not inherited

- A private type cannot be derived unless it is tagged
- Special aspect Type_Invariant'Class preferred for tagged types

Other Useful Gotchas on Predicates and Type Invariants

GNAT defines its own aspects Predicate and Invariant

- Predicate is the same as Static_Predicate if property allows it
- Otherwise Predicate is the same as Dynamic_Predicate
- Invariant is the same as Type_Invariant

Referring to the "current object" in the property

- The name of the type acts as the "current object" of that type
- Components of records can be mentioned directly

Type invariants on protected objects

- Ada/SPARK do not define type invariants on protected objects
- Idiom is to use a record type as unique component of the PO, and use a predicate for that record type

Default Initial Condition

Aspect defined in GNAT to state a property on default initial values of a private type

- Introduced for proof in SPARK
- GNAT introduces a dynamic check when -gnata is used
- Used in the formal containers library to state that containers are initially empty

```
type List (Capacity : Count_Type) is private with
  Default_Initial_Condition => Is_Empty (List);
```

Can also be used without a property for SPARK analysis

- No argument specifies that the value is fully default initialized
- Argument **null** specifies that there is no default initialization

Quiz

Is this correct? 1/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Day range Wednesday | Weekend;
```

Is this correct? 1/10

NO

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Day range Wednesday | Weekend;
```

The syntax of range constraints does not allow sets of values. A predicate should be used instead.

Is this correct? 2/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Weekend with
  Static_Predicate => Day_Off in Wednesday | Weekend;
```

Is this correct? 2/10

NO

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Weekend with
  Static_Predicate => Day_Off in Wednesday | Weekend;
```

This is accepted by GNAT, but result is not the one expected by the user. Day_Off has the same constraint as Weekend.

Is this correct? 3/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Day with
  Dynamic_Predicate => Day_Off in Wednesday | Weekend;
```

Is this correct? 3/10


```
type Day is (Monday, Tuesday, … Sunday);
subtype Weekend is Day range Saturday .. Sunday;
subtype Day_Off is Day with
  Dynamic_Predicate => Day_Off in Wednesday | Weekend;
```

It is valid to use a Dynamic_Predicate where a Static_Predicate would be allowed.

Is this correct? 4/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
function Next_Day_Off (D : Day_Off) return Day_Off is
begin
   case D is
      when Wednesday => return Saturday;
      when Saturday  => return Sunday;
      when Sunday    => return Wednesday;
   end case;
end Next_Day_Off;
```

Is this correct? 4/10


```
function Next_Day_Off (D : Day_Off) return Day_Off is
begin
   case D is
      when Wednesday => return Saturday;
      when Saturday  => return Sunday;
      when Sunday    => return Wednesday;
   end case;
end Next_Day_Off;
```

It is valid to use a type with Static_Predicate for the value tested in a case statement. This is not true for Dynamic_Predicate.

Is this correct? 5/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type Week_Schedule is private with
      Type_Invariant => Valid (Week_Schedule);
   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record;
   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

Is this correct? 5/10


```
   type Week_Schedule is private with
      Type_Invariant => Valid (Week_Schedule);
   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record;
   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

It is valid in Ada because the type invariant is not checked on entry or return from Valid. Also, function Valid is visible from the type invariant (special visibility in contracts).

But it is invalid in SPARK, where private declaration cannot hold a type invariant. The reason is that the type invariant is assumed in the precondition of public functions for proof. That would lead to circular reasoning if Valid could be public.

Is this correct? 6/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type Week_Schedule is private;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record with
      Type_Invariant => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

Is this correct? 6/10


```
   type Week_Schedule is private;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record with
      Type_Invariant => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

This version is valid in both Ada and SPARK.

Is this correct? 7/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
subtype Sorted_String is String with
  Dynamic_Predicate =>
    (for all Pos in Sorted_String'Range =>
       Sorted_String (Pos) <= Sorted_String (Pos + 1));

subtype Unique_String is String with
  Dynamic_Predicate =>
    (for all Pos1, Pos2 in Unique_String'Range =>
       Unique_String (Pos1) /= Unique_String (Pos2));

subtype Unique_Sorted_String is String with
  Dynamic_Predicate =>
    Unique_Sorted_String in Sorted_String and then
    Unique_Sorted_String in Unique_String;
```

Is this correct? 7/10

NO

```
subtype Sorted_String is String with
  Dynamic_Predicate =>
    (for all Pos in Sorted_String'Range =>
       Sorted_String (Pos) <= Sorted_String (Pos + 1));

subtype Unique_String is String with
  Dynamic_Predicate =>
    (for all Pos1, Pos2 in Unique_String'Range =>
       Unique_String (Pos1) /= Unique_String (Pos2));

subtype Unique_Sorted_String is String with
  Dynamic_Predicate =>
    Unique_Sorted_String in Sorted_String and then
    Unique_Sorted_String in Unique_String;
```

There are 3 problems in this code:

- **there is a run-time error on the array access in Sorted_String**
- **quantified expression defines only one variable**
- **the property in Unique_String is true only for the empty string**

Is this correct? 8/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
subtype Sorted_String is String with
  Dynamic_Predicate =>
    (for all Pos in Sorted_String'First ..
                    Sorted_String'Last - 1 =>
       Sorted_String (Pos) <= Sorted_String (Pos + 1));

subtype Unique_String is String with
  Dynamic_Predicate =>
    (for all Pos1 in Unique_String'Range =>
      (for all Pos2 in Unique_String'Range =>
         (if Pos1 /= Pos2 then
            Unique_String (Pos1) /= Unique_String (Pos2))));

subtype Unique_Sorted_String is String with
  Dynamic_Predicate =>
    Unique_Sorted_String in Sorted_String and then
    Unique_Sorted_String in Unique_String;
```

Is this correct? 8/10


```
subtype Sorted_String is String with
  Dynamic_Predicate =>
    (for all Pos in Sorted_String'First ..
                    Sorted_String'Last - 1 =>
       Sorted_String (Pos) <= Sorted_String (Pos + 1));

subtype Unique_String is String with
  Dynamic_Predicate =>
    (for all Pos1 in Unique_String'Range =>
      (for all Pos2 in Unique_String'Range =>
         (if Pos1 /= Pos2 then
            Unique_String (Pos1) /= Unique_String (Pos2))));

subtype Unique_Sorted_String is String with
  Dynamic_Predicate =>
    Unique_Sorted_String in Sorted_String and then
    Unique_Sorted_String in Unique_String;
```

This is a correct version in Ada. For proving AoRTE in SPARK, one will need to change slightly the property of Sorted_String.

Is this correct? 9/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type Week_Schedule is private with
      Default_Initial_Condition => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record;

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

Is this correct? 9/10


```
   type Week_Schedule is private with
      Default_Initial_Condition => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off, Day_On_Duty : Day;
   end record;

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

The default initial condition is not satisfied.

Is this correct? 10/10

YES
(click on the check icon)

NO
(click on the error location(s))

```
   type Week_Schedule is private with
      Default_Initial_Condition => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off     : Day := Wednesday;
      Day_On_Duty : Day := Friday;
   end record;

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

Is this correct? 10/10


```
   type Week_Schedule is private with
      Default_Initial_Condition => Valid (Week_Schedule);

   function Valid (WS : Week_Schedule) return Boolean;
private
   type Week_Schedule is record
      Day_Off     : Day := Wednesday;
      Day_On_Duty : Day := Friday;
   end record;

   function Valid (WS : Week_Schedule) return Boolean is
      (WS.Day_Off /= WS.Day_On_Duty);
```

This is a correct version, which can be proved with SPARK.

university.adacore.com