Confomarnce and conversion rotuines

It is essential to define well when the assignment is valid. The general rule is rather straightforward – the type of the expression on the right side of the assignment should either conform to the type of the writable on the left side or have a proper conversion routine be in place.

Let’s start with conformance. The simplest case of conformance that every type conforms to itself.

a: A **is new** A

The next case is when there is a path in the inheritance graph between the current unit type and another one. And this path consists only of conformant inheritance edges.

**unit** A **end**

**unit** B **extend** A **end** // That is a conformant inheritance

**unit** C **extend** ~A **end** // That is a non-conformant inheritance

a: A **is new** **B** // OK! As B conforms to A

a: A **is new** C // Compile-time error as C does not conform to A

Next is when the type is generic instantiation then in addition to unit type conformance it is necessary to take into account type by type conformance of all elements of the instantiation.

**unit** A[U, V] **end**

**unit** B[X, Y] **extend** A [X, Y] **end**

**unit** T1 **end**

**unit** T2 **end**

**unit S1 extend** T1 **end**

**unit** A[A, B, C] **end**

a: A[T1, T2] **is new** A[T1, T2]// OK!

a: A[T1, T2] **is new** A[S1, T2]// OK!

a: A[T1, T2] **is new** A[T1, S1]// Compile time error as S1 does not conform to T2

a: A[T1, T2] **is new** B[T1, T2]// OK!

a: A[T1, T2] **is new** B[S1, T2]// OK!

a: A[T1, T2] **is new** B[T1, S1]// Compile time error as S1 does not conform to T2

a: A[T1, T2] **is new** A[T1, T2, S1]/\* Compile time error as A with 3 generic parameters does not conform to A with 2 generic parameters \*/

And last but not least is tuple conformance. All tuples are of the same type – tuple type and it means that we need to consider (similar to generic instantiations) by-element conformance of element types.

a: (T1, T2) **is** (**new** T1, **new** T2)// OK!

a: (T1, T2) **is** (**new** S1, **new** T2)// OK!

a: (T1, T2) **is** (**new** T1, **new** S1)// Compile time error as S1 does not conform to T2

a: (T1, T2) **is** (**new** S1, **new** T2, **new** S1)/\* OK! as all elements of the longer tuple, which has corresponding elements in the shorter tuple, conform to them \*/

And now let’s consider conversion routines as they also play important roles in assignments. There are two types of converison rotuines: from-conversion and to-conversion. The first one is a procedure with one parameter and the second one is a function with no arguments. Let’s examine the following example

**unit** A

:= (other: T) **do end**

/\* That is a from-conversion procedure, which has some algorithm how to perform conversion from objects of type T into the objects of current type A \*/

:= (): T **do end**

/\* That is a to-conversion function which creates a proper object of type T and

works well for assignments too \*/

foo (arg: T) **do end**

**end**

**unit** T **end**

**var** a **is new** A /\* We need to create a valid object of type A first. To ensure A invariant is held \*/

a := **new** T /\* And then we can assign to an object of type A the object of type T using the from-convertor procedure \*/

a: A **is new** T /\* That is incorrect (compile-time error) as object a of type A was not created yet \*/

A.foo (**new** T) // That is Ok as T conforms to T

A.foo (**new** A) /\* That is OK as unit A has to-conversion function to type T, the semantics of any rotuien call is that arguments passing is an assignment of arguments to parameters so conformance and conversion functions will work and that is why conversion functions are marked with the := sign \*/