Often we have to type init procedures when the body of the init procedure is just a set of assignments of parameters into unit attributes. We have assigners (setters) for particular attributes and a similar scheme can be applied for init procedures. The trick is to use anchored types “as <name\_of\_the\_attribute>”. If init is marked with “:=” sign then the compiler will automatically generate the body of the init procedure saving the time and efforts for more important coding. See example below

**unit** A

// Unit has 4 attributes

a1: T1

a2: T2

a3: T3

a4: T4

**init** (**:=** a: **as** a2; b: **as** a1) /\* Init procedure parameters has ‘:=’ prefix \*/

[**require**

a < b

// The compiler will automatically generate the body

do

*a2 := a*

*a1 := b*

end // init] // We can optionally add precondition for init

**init** := (**as** a1; **as** a4) /\* Even shorter form can be used when there is no require clause and no need to name parameters\*/

The compiler will automatically generate the body

*do*

*a1 := \_first\_parameter*

*a4 := \_forth\_parameter*

*end*

**init** (**:=** a: **as** a3; b: T4; **as** a4)

/\* The most advanced case when we have just a few parameters in the form of as and := postfix \*/

The compiler will automatically generate necessary assignments right before the actual body starts.

*a3 := a*

*a4 := \_third\_parameter*

are done before ‘do’ in left to right order of parameters declaration. Two ‘as’ parameters referring to the same attribute is prohibited. (**as** a1; **as** a1) – is an error

**do**

a1 := b.getA1

a2 := b.getA2

**end** // init

**end** // unit

a1 **is new** A (**new** T2, **new** T1)

a2 **is new** A (**new** T3, **new** T4)

The same approach with compiler-generated asignemnts can be applied to any unit rotuine

foo (**:=** a: **as** a3; b: T4; **as** a4) **do**

**end**