In the case of constrained genericity, we must extend the rule CTUG given earlier for unconstrained genericity, includes an extra condition for constrained genericity: not only must the number of actual parameters match the number of formal parameters; the conformance requirements must also be met. So if *C* is a generic class *C* [*G*, ...], the Unconstrained Genericity rule required that any generically derived type be of the form *C* [*T*, ...] with the same number of actual and formal generic parameters. If a formal generic parameter is given as *G* –> *D* with a constraint *D*, the corresponding actual *T* must conform to *D*.

Here is the precise rule (whose first part, for completeness, repeats the two clauses of the Unconstrained Genericity rule):

Constrained Genericity rule *CTCG*

Let *C* be a constrained generic class. A Class_type *CT* having *C* as base class is valid if and only if *CT* satisfies the two conditions of the Unconstrained Genericity rule (CTUG):

1 • *C* is a generic class.

2 • The number of Type components in *CT*’s Actual_generics list is the same as the number of Formal_generic parameters in the Formal_generic_list of *C*’s declaration.

and, in addition:

3 • For any Formal_generic parameter in the declaration of *C* having a constraint of the form –> *D*, the corresponding Type in the Actual_generics list of *CT* conforms to the type obtained from *D* by replacing every ocurrence of a formal generic parameter of *C* by the corresponding actual generic parameter in *CT*.

Clauses 1 and 2 are a repetition of the clauses of the Unconstrained Genericity rule, CTUG, page 360.

Again, the existence of the base class in the surrounding universe will be ensured by the Class Type rule. In clause 1, we don’t need the clause “*except if C class TUPLE*”, since *TUPLE* as defined in the Kernel Library is not constrained.

At first the phrasing of clause 3 seems more complicated than necessary: why must the actual generic parameter conform not just to *D* but to “the type obtained from *D* by ...”? This is to permit *recursive generic constraints*, as explained next.

Also note that there is no specific validity rule applying to the generic constraint itself, *D*. It simply needs to be a valid type. In fact it can involve a generic parameter, or even *be* a generic parameter; this is the case of “recursive generic constraints”, the topic of the next section.

12.9 RECURSIVE GENERIC CONSTRAINTS

(Although important, the case described in this section does not arise in elementary uses, and may be skipped in a first reading.)

To understand the last part of clause 3 of the Constrained Genericity rule, assume you want to define a class as

```
class C [G, H –> ARRAY [G]] ...
```

This makes perfect sense and the intent is clear: you want to allow any type of the form *C* [*T*, *U*] where *T* is an arbitrary type and *U* is *ARRAY* [*T*] or a type conforming to *ARRAY* [*T*]. So the following will be valid

The Class Type rule appeared on page 356.

```
 C [INTEGER, ARRAY [INTEGER]]
C [POLYGON, FIXED_LIST [POLYGON]]
        -- Where FIXED_LIST is a descendant of ARRAY
```

But for example *C* [*INTEGER*, *REAL*] is not valid. Similarly, you should be able to define

```
class C [G –>H, H –> G] ...
```

meaning: the first actual generic parameter must conform to the first, and conversely. Only derivations of the form *C* [*T*, *T*], using the same type as actual generic parameter, will be valid. Unlike the first example, this scheme seems useless, but for consistency it is permitted.

This is the reason for the phrasing of clause 3 of the Constrained Genericity rule. If we just required that

"*the corresponding* Type *in the* Actual_generics *list of CT conform to D*"

then in our first example *ARRAY* [*INTEGER*] does not conform to *ARRAY* [*G*]; actually this conformance question is meaningless since there usually won't even be a type *G* in the class that wants to use *C* [*INTEGER*, *ARRAY* [*INTEGER*]. Similarly, in the *C* [*G* –>*H*, *H* –> *G*] example, if we want to use *C* [*T*, *T*] in a certain class other than *C*, the questions "does *T* conform to *G*?" and "does *T* conform to *H*?" are meaningless in that class.

For such conformance questions to become meaningful, we must first replace, in the constraint, any occurrence of a formal parameter by the corresponding actual parameter. Hence the rephrased clause:

"*the corresponding* Type *in the Actual_generics list of CT* [must] *conform to the type obtained from D by replacing every ocurrence of a formal generic parameter of C by the corresponding actual generic parameter in CT.*"