Dead Live Longer
A Dramoletto[†]

Christoph Grein
c/o ESG, 81675 München , eMail: Christ-Usch.Grein@T-Online.de

A large and dark room. In the foreground a terminal, at the rear wall a huge screen showing everything that happens at the terminal. At first the screen is off, the room in twilight. Below the stage, but visible, Lady Ada in a coffin. Around the stage on a gallery, nearly invisible in the dark (also when the stage is lit), the chorus of Ada gurus.

Seated on a swan, **Tucker Taft** *hovers down from the ceiling, murmuring some unintelligible quotes from the Ada Reference Manual:*

RM-83 4.5.2(5,6,7) and RM-95 4.5.2(24) ... Equality ...
[rest unintelligible] ... Compound objects ...
[rest unintelligible] ... Tagged types ...
[rest unintelligible] ... Generics ...
[rest unintelligible] ...

The screen brightens, putting the room into an eerie light, and a lake appears on it. Another swan enters the picture, carrying John Goodenough. Tucker's swan penetrates the screen and, with a splash, lands in the lake beside John Goodenough. **Tucker Taft** *startles and shouts:*

Heureka! Liberty, Equality.
Brother, we need equality for all!

John Goodenough *declaims:*

It is already good enough,
Equality for all's a bluff.

Lady Ada suffers convulsive jerks. Tucker Taft soars off, the swan with John Goodenough swims away, and the screen grows dim, leaving the room again in twilight, while the **chorus** *chants:*

Fair is foul and foul is fair
Hover through the filthy air.

Spotlights on, the room becomes as light as day. Enter **Knopp** *(a programmer). He sits down in front of the terminal and starts writing; the screen at the wall displays:*

```
type Container is record
  Length : Natural;
  Content: String (Some_Range);
end record;

function "=" (Left, Right: Container)
  return Boolean is
begin
  return Left.Content (1 .. Left.Length) = Right.Content (1 .. Right.Length);
end "=";

declare
  T: constant String := "Lady Ada";
  CON_1, CON_2: Container;
  B_Con: Boolean;
begin
  CON_1.Length := T'Length;  CON_1.Content (1 .. T'Length) := T;
  CON_2.Length := T'Length;  CON_2.Content (1 .. T'Length) := T;
  B_Con := CON_1 = CON_2;
end;
```

[†] This is the translation of a text published in the German periodical *Softwaretechnik-Trends* Vol. 23.4 (November 2003) http://pi.informatik.uni-siegen.de/stt.

Knopp *starts a program, the result is shown on the screen:*

```
CON_1 = CON_2 returns True! (qed)
```

Chorus *chants:*

Knopp vermeidet dieſen Ort
Und begibt ſich eilig fort.

Swiftly Knopp evades this place
And moves on with speedy pace.

Exit Knopp happily. Enter Knopp's **Successor**. *He writes and the screen displays:*

```
type Container is record
  Length : Natural;
  Content: String (Some_Range);
end record;

function "=" (Left, Right: Container)
  return Boolean is
begin
  return Left.Content (1 .. Left.Length) = Right.Content (1 .. Right.Length);
end "=";

type Super_Container is record
  C: Container;
end record;

declare
  T: constant String := "Lady Ada";
  CON_1, CON_2: Container;
  SUP_1, SUP_2: Super_Container;
  B_Con, B_Sup: Boolean;
begin
  CON_1.Length := T'Length;  CON_1.Content (1 .. T'Length) := T;
  CON_2.Length := T'Length;  CON_2.Content (1 .. T'Length) := T;
  SUP_1 := (C => CON_1);     SUP_2 := (C => CON_2);
  B_Con := CON_1 = CON_2;    B_Sup := SUP_1 = SUP_2;
end;
```

Lady Ada turns in her grave with those activity's meaningfulness. **Chorus** *sings (a tune from "The Bartered Bride" by Bedrich Smetana):*

Oh a lot of sweeties are cajoling pussies,
paws as soft as velvet caress you.

Successor *starts a program, screen displays:*

```
CON_1 = CON_2 returns True!  (qed)
SUP_1 = SUP_2 returns False! (@?!)
```

Lady Ada rises, turns into a tiger and digs her claws into the Successor, while the **chorus** *sings:*

> But how terrifying when they start applying
> tiger claws on you.

Uproar in the audience (Ada programmers), the **Producer** *(Ada Rapporteur Group) dashes on the stage tearing his hair, groans:*

> When does the next swan leave?

On the screen, the lake reappears, clouds gather, the room becomes dimly lit as before, lightnings flash across the sky. **Robert Dewar***'s thundering voice from off-screen:*

> I do not see how you could think otherwise…

While the curtains are falling, you can hear **Reich-Ranitzki** *(a famous German literary critic) holding forth:*

> A very bad play! The mere idea of it is poor, let alone the realization in the Ada Reference Manual 4.5.2(24) – defective. Either something is equal or it is not! Tertium non datur!

Ada 95 has removed some restrictions from the language. Now it is legal to define equality for any type. In Ada 83 this was possible for limited types only (since they haven't a predefined equality), which sometimes led people to make types limited just for this purpose, with the unpleasant consequence of losing assignment – you had to define an "assign" procedure instead.

The new rule is meant to be used on types like `Container` for which the predefined equality incorrectly also considers unused components (those past the range `1 .. Length`). The aforementioned function overrides the predefined equality, remedying this deficiency.

But watch out! The predefined equality is not hidden forever. It for instance re-emerges each time when such a type is used in composite types as a component (i.e. in arrays and records) and objects of those types are compared for equality as a whole. It is however important that the type is not tagged; for tagged types, which did not exist in Ada 83, the overridden equality has really disappeared forever, only the new one will be used at all places.

Therefore, in order to prevent re-emergence of the predefined equality when comparing composite objects whose non-tagged components use a redefined equality, you always have to define the following:

```
type Super_Container is record
  C: Container;
end record;

function "=" (Left, Right: Super_Container) return Boolean is ...;
```

You also have to pay attention to generic instantiations where this effect is present as well. Here you have to take care of transferring the equality as a formal parameter to the generic unit:

```
generic
  type T is private;
  -- Here, predefined equality for
  -- T is implicitly defined; hence
  -- it re-emerges even when
  -- overridden.
package Reemergence is
  ...
end Reemergence;

generic
  type T is private;
  -- This declaration prevents
  -- re-emergence:
  with function "=" (Left, Right: T)
         return Boolean is <>;
package No_Reemergence is
  ...
end No_Reemergence;
```

Which is the origin of this surprising rule in Ada 95 that *equality does not compose except for tagged types*? The rule is defined in RM 4.5.2(24); the annotated reference manual AARM 4.5.2(24.a) only provides a short hint to upward compatibility with Ada 83.

(This whole discussion does however not apply to language-defined types, cf. RM 4.5.2(32.1/1).)

The language creators (Jean Ichbiah et al.) of Ada 83 intended two ways to handle such cases like `Container`. Either with each assignment, also fill the unused components with null-values, so that the predefined equality works correctly, which however for large arrays can become quite inefficient; or make `Container` limited legalizing the definition of "=", at the same time losing assignment, which is equally unfortunate.

There is however a third, albeit hidden, method in Ada 83 to define equality for any type. You have to resort to a trick attributed to John Goodenough. The trick, allegedly well-known and often used, is to derive from a generic formal limited private type and to define equality for this new type, like so:

```
-- Ada 83, after John Goodenough (also valid in Ada 95, but unnecessary -
-- cum mortuis in lingua mortua)

generic
  type Ancestor is limited private;
  with function Equal (Left, Right: Ancestor) return Boolean;
package Define_Equality is
  type Descendant is new Ancestor;
  function "="  -- here legal
   (Left, Right: Descendant) return Boolean;
end Define_Equality;
```

Upon instantiation of `Define_Equality`, the type `Descendant` inherits all properties of the actual for `Ancestor` and additionally has equality defined, i.e. (outside of the package `Define_Equality`) Descendant is not limited if its parent type is not limited.

```
function Equal  -- "=" in Ada 83 here illegal
  (Left, Right: Container) return Boolean is
begin
  return Left.Content (1 .. Left.Length) = Right.Content (1 .. Right.Length);
end Equal;

package Container_Equality is new Define_Equality (Container, Equal);

type New_Container is new Container_Equality.Descendant;
```

This new equality is however valid only for objects of the type `Descendant` and further derived types like `New_Container`, but not for components of this type in composite objects. Voilà – Ada 95 having to be upward compatible with Ada 83, it obeys this darn rule.