Ada Magica

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

Tote leben länger
Ein Dramolett

Ein großer, dunkler Raum. Im Vordergrund ein Terminal, hinten an der Wand ein großer Bildschirm, auf dem alles erscheint, was am Terminal geschieht. Zu Anfang ist der Schirm dunkel, der Raum im Halbdunkel. Unter der Bühne, aber sichtbar, Lady Ada in einem Sarg. Rund um den Raum auf einer Empore, fast unsichtbar im Dunkel (auch wenn die Bühne erleuchtet ist), der Chor der Ada-Gurus.

Tucker Taft *schwebt auf einem Schwan von der Decke herab. Währenddessen murmelt er einige unverständliche Zitate aus dem Ada Reference Manual:*

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

Der Schirm leuchtet auf, wodurch er den Raum in ein unwirkliches Licht taucht, und ein See erscheint darauf. Ein weiterer Schwan kommt ins Bild, der John Goodenough trägt. Tuckers Schwan durchdringt den Schirm und landet im See mit einem Platsch neben John Goodenough.
Tucker Taft *schreckt hoch und ruft:*

> Heureka! Freiheit, Gleichheit.
> Brüderchen – wir brauchen Gleichheit für alle!

John Goodenough *deklamiert:*

> Es ist schon alles gut genug,
> Gleichheit für alle ist ein Trug.

Lady Ada erleidet konvulsivische Zuckungen, Tucker Taft entschwebt, der Schwan mit John Goodenough schwimmt davon, und der Bildschirm erlischt, wodurch der Raum wieder im Halbdunkel versinkt, während der **Sprechchor** *intoniert:*

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

Scheinwerfer leuchten auf, der Raum wird taghell.
Knopp *(ein Programmierer) betritt die Bühne, setzt sich ans Terminal und schreibt; es erscheint auf dem Bildschirm an der Wand:*

```
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;
```

Knopp *startet ein Programm, die Ausgabe erscheint auf dem Bildschirm:*

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

Chor *im Sprechgesang:*

Knopp fröhlich ab. Knopps **Nachfolger** *tritt auf, schreibt, und der Bildschirm zeigt:*

```
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 dreht sich im Grabe um ob der Sinnhaftigkeit solchen Tuns. **Chor** *singt:*

> Ach so manches Schätzchen ist ein Schmeichelkätzchen,
> das mit Sammetpfötchen dich umgibt.

Nachfolger *startet ein Programm, Bildschirm zeigt:*

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

Lady Ada erhebt sich, verwandelt sich in einen Tiger und krallt den Nachfolger. **Chor** *singt dazu:*

> Aber wie entsetzlich, wenn man einmal plötzlich Tigerkrallen spürt.

Tumult im Publikum (Ada-Programmierer), **Regisseur** *(Ada Rapporteur Group) stürzt auf die Bühne, rauft sich die Haare, stöhnt:*

> Wann geht der nächste Schwan?

Auf dem Schirm erscheint wieder der See, Wolken ziehen auf, es wird dämmrig wie zu Anfang, Blitze zucken über den Himmel. **Robert Dewar***s Stimme donnernd aus dem Off:*

> I do not see how you could think otherwise…

Während der Vorhang fällt, hört man **Reich-Ranitzki** *tönen:*

> Ein sehr schlechtes Stück! Schon die Idee ist dürftig, und die Ausführung im Ada Reference Manual 4.5.2(24) ist mangelhaft. Entweder ist etwas gleich, oder es ist es nicht! Tertium non datur!

Ada 95 hat einige Einschränkungen aus der Sprache entfernt. So ist es jetzt erlaubt, die Gleichheit für beliebige Typen zu definieren. In Ada 83 war das nur für limitierte Typen erlaubt (für die ja von Haus aus keine Gleichheit definiert ist), was manchmal dazu führte, dass Typen nur aus diesem Grund limitiert wurden, mit der unangenehmen Folgerung, dass dann keine Zuweisung mehr möglich war. Statt dessen musste zusätzlich eine Zuweisungsprozedur definiert werden.

Die neue Regel ist hauptsächlich gedacht für Typen der Gestalt `Container`, für die die vordefinierte Gleichheit fälschlicherweise auch unbenutzte Komponenten in `Content` (die jenseits des Bereichs `1 .. Length`) berücksichtigt. Die obige Funktion überschreibt die vordefinierte Gleichheit und kann diese Unzulänglichkeit leicht beheben.

Aber Vorsicht! Die vordefinierte Gleichheit ist nicht für alle Zeiten überschrieben. Sie taucht zum Beispiel immer dann wieder auf, wenn ein solcher Typ in zusammengesetzten Typen als Komponente verwendet wird (also in Feldern und Verbunden) und solche zusammengesetzten Objekte insgesamt auf Gleichheit untersucht werden. Dabei ist wesentlich, dass der Typ nicht etikettiert (*tagged*) ist – für etikettierte Typen, die es in Ada 83 ja noch nicht gab, wird stets die neue Gleichheit angewendet, die vordefinierte ist tatsächlich für alle Zeit verschwunden.

Wenn wir also zusammengesetzte Typen, deren nicht-etikettierte Komponenten eine umdefinierte Gleichheit verwenden, in ihrer Gänze auf Gleichheit überprüfen wollen, müssen wir für sie stets zusätzlich folgendes definieren, um das Wiederauftauchen der vordefinierten Gleichheit zu verhindern:

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

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

Auch bei generischen Ausprägungen gilt es aufzupassen, denn hier tritt dieser Effekt ebenfalls auf. Bei ihnen müssen wir darauf achten, dass der generischen Einheit die Gleichheit als formaler Parameter übergeben wird:

```
generic
  type T is private;
  -- Hier ist die vordefinierte
  -- Gleichheit für T implizit
  -- definiert; sie taucht also
  -- wieder auf, auch wenn sie
  -- überschrieben wurde.
package Reemergence is
  ...
end Reemergence;

generic
  type T is private;
  -- Diese Deklaration verhindert
  -- das Wiederauftauchen:
  with function "=" (Left, Right: T)
         return Boolean is <>;
package No_Reemergence is
  ...
end No_Reemergence;
```

Woher kommt nun diese überraschende Regel in Ada 95, dass die vordefinierte Gleichheit unter Umständen wieder auftaucht? Sie steht im RM 4.5.2(24), das annotierte Referenzmanual liefert in AARM 4.5.2 (24.a) mit dem Hinweis auf Kompatibilität zu Ada 83 eine überaus knappe Begründung.

(Diese ganze Problematik existiert allerdings nicht für sprachdefinierte Typen, siehe RM 4.5.2(32.1/1).)

In Ada 83 sind von den Sprachschöpfern (Jean Ichbiah et al.) zwei Möglichkeiten vorgesehen worden, mit Fällen wie Container umzugehen. Zum Einen werden jedesmal die unbenutzten Komponenten mit Null-Werten aufgefüllt, so dass die vordefinierte Gleichheit richtig wird, was jedoch für große Felder ungebührlich aufwendig sein kann. Oder man macht Container limitiert, wodurch die Definition von "=" legal wird, gleichzeitig jedoch die Zuweisung illegal, was äußerst unpraktisch ist.

Allerdings gibt es in Ada 83 auch noch eine dritte, versteckte Möglichkeit, die Gleichheit für beliebige Typen neu zu definieren. Man musste dazu einen Trick verwenden, der John Goodenough zugeschrieben wird. Der Trick, angeblicherweise wohl bekannt und oft verwendet, besteht darin, von einem generischen formalen limitiert-privaten Typ abzuleiten und für diesen neuen Typen die Gleichheit zu definieren. Das sieht folgendermaßen aus:

```
-- Ada 83, nach John Goodenough (auch
-- in Ada 95 gültig, aber unnötig -
-- 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 "="  -- hier erlaubt
   (Left, Right: Descendant)
   return Boolean;
end Define_Equality;
```

Wenn nun Define_Equality ausgeprägt wird, erbt der Typ Descendant alle Eigenschaften vom aktuellen Typ für Ancestor und erhält zusätzlich noch den Gleichheitsoperator; das heißt, Descendant ist (außerhalb des Pakets Define_Equality) nicht limitiert, wenn sein Elterntyp nicht limitiert ist.

```
function Equal  -- "=" in Ada 83 hier
                -- nicht erlaubt
  (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;
```

Diese neue Gleichheit gilt allerdings nur für Objekte des Typs Descendant und davon abgeleiteten Typen wie New_Container und nicht für Komponenten dieses Typs in zusammengesetzten Objekten. Voilà – da Ada 95 aufwärts kompatibel zu Ada 83 sein musste, haben wir nun diese unselige Regel.