****

J3/02-330

Date: November 2002
To: J3
From: Aleksandar Donev
Subject: The value of something
Reference: J3-007R2, J3/02-302, J3/02-294r1

****

Summary

There is a big gap in the draft as to what the typical words "something gets the value of something" really mean for objects of derived types with defined assignment. The wording might also be missing for all objects of derived-types, but the meaning in that case is clear: Do recursive ``bitwise'' copy for each component of intrinsic type, deep copy for allocatables, shallow for pointers. For things that have defined type-bound assignment, even the meaning is not clear, because defined assignment may need to be envoked, which is dangerous. Further issues are brought on if there is a type-bound finalization procedure there and when and if that should be called.

I also point to another issue with defined type-bound assignment which should be considered in conjunction with the above one.

J3 DATA subgroup needs to consider this issue carefully and make specific edits to correct the problem before the March meeting.

Problems with "value of"

Take the following example:

```
TYPE :: my_type
   ...
CONTAINS
   GENERIC :: ASSIGNMENT(=) => my_assignment
END TYPE

INTERFACE
   SUBROUTINE my_assignment(lhs,rhs)
      TYPE(my_type), INTENT(INOUT) :: lhs
      TYPE(my_type), INTENT(IN) :: rhs
   END SUBROUTINE
END INTERFACE

TYPE(my_type) :: R
```

The problem occurs in many places:

1) Intrinsics (see J3/02-302)

In some intrinsic functions (see J3/02-302) we say the result is obtained from the argument, so some kind of a copy is made. How? The natural thing seems to be to that the copy is made as if intrinsic assignment were performed:
copy_of_object=object
where the declared type of the lhs and rhs is the dynamic type of object.

There are big issues with this. Intrinsic assignment for objects of derived type that contain components of a derived type with an accessible defined assignment invokes this defined assignment (see 141:13). This was apparently decided during an interp request by J3/WG5. So now take the example:

```
TYPE(my_type), PARAMETER :: P
```

```
TYPE(my_type) :: Q=MERGE(P,P,.TRUE.)
```

Should the initialization of Q invoke the user-defined assignment of P? It better not unless compilers are much smarter then I know them to be.

---

2) Expression evaluation

How is the expression (R) evaluated? Is my_assignment invoked?

---

3) VALUE arguments

Assume we pass R as an actual argument for a dummy argument:
```
TYPE(my_type), VALUE :: S
```
How is the temporary copy (see 265: 35-37) that is associated with the dummy obtained, exactly?

---

4) source-variable in ALLOCATE
When cloning an object via SOURCE in ALLOCATE we use intrinsic assignment (see 110:9-10). Is this really what is intended.

Access to assignment inside defined assignment

When writing subroutines to perform defined assignment, it is often most useful to have access to some kind of ``default'' assignment, and then add some extra statements in there (like usage counting statistics). But as soon as defined assignment is accessible, one cannot access the intrinsic assignment (it would really be a recursive call to the defined one). This is rather unfortunate.

I am bringing this up here because my basic idea to the problem of making copies of objects of defined type is to invent a new term, say "default assignment", which will be like intrinsic assignment, but will not invoke defined assignments. Then we will say that for something to get the value of something really means to use this default assignment. I would eventually want to make this kind of assignment directly accessible to the users too.