

## Minutes of ARG Meeting 63G

9 January 2025

### Attendees:

Steve Baird, John Barnes, Randy Brukardt, Jeff Cousins, Gary Dismukes, Bob Duff, Christoph Grein, Niklas Holsti, Brad Moore, Alejandro Mosteo, Jean-Pierre Rosen, Justin Squirek, Tucker Taft, Tullio Vardanega (joined 10:37), Richard Wai.

### Observers:

None.

## Meeting Summary

The meeting convened on Thursday, 9 January 2025 at 10:33 hours EST and adjourned at 13:33 hours EST. The meeting was held using Zoom. The meeting covered all of the new and updated AIs on the agenda.

### AI Summary

The following AI was approved:

AI22-0121-1/02 Internal calls in contracts of protected types (15-0-0)

The following AIs were approved with editorial changes:

AI22-0117-1/03 More Terms and Definitions (15-0-0)

AI22-0122-1/01 Order of finalization of components (15-0-0)

AI22-0125-1/01 Atomic\_Operations and checks (13-0-2)

The intention of the following AI was approved but it requires a rewrite:

AI22-0124-1/02 Aggregates and capacity of bounded containers (14-0-1)

The following AIs were discussed and assigned to an editor:

AI22-0022-1/04 Difficult example issues from WG 9 review

AI22-0075-1/02 Explicitly Aliased Results (13-0-2)

AI22-0076-1/02 Restricting Dynamic Accessibility Checks

AI22-0123-1/03 Type of aggregates and raise expressions (15-0-0)

The following Github Issues were discussed and assigned to an editor to create an AI:

#12 Iterator over vectors should preserve unconstrained state of elements of discriminated type

#100 Allow Add\_Named and Add\_Unnamed together

The following Github Issues were discussed without reaching a conclusion:

#83 Full relational operators for ordered containers

## Detailed Minutes

### Welcome

Steve welcomes everyone to this meeting.

### Apologies

Arnaud Charlet reported that he had a conflict.

### Previous Meeting Minutes

There were no comments on the minutes: Approve minutes: 15-0-0.

### ***Date and Venue of the Next Meeting***

Our next electronic meeting is proposed for Thursday, March 20. Tucker has a conflict for that day. He notes that the following week is worse for him. So we decide to have the meeting on Wednesday, March 19 at the usual time (10:30-13:30 EDT [-6 UTC]) and method (Zoom). [Editor's note: It has come to my attention that while Daylight Savings Time will have started in the US before the meeting date, Summer Time will not yet have started in Europe. So the "usual time" in the US is not the "usual time" for Europeans, it will be an hour later.]

### ***Ada User Society***

The Ada User Society is up and running. Please join to support the continuation of Ada.

We congratulate Tullio for the hard work it has taken to get this far.

### ***ARG Voting rights***

Steve explains that WG 9 has asked us to adopt a policy for voting rights for ARG members. Our understanding is that there is an SC 22 requirement to do so.

Jeff notes that BSI requires attendance at 1/3rd of the meeting. Tucker says a group he is involved in has a rule about missing 3 meetings in a row.

The management is asked to have a discussion on this topic on the ARG mailing list. Randy takes an action item to do this.

### ***Progress report on the Corrigendum***

Randy gives a status report. He says that he has gotten all but 9 of the approved AIs into a Corrigendum document, but he still needs to reformat it, remove inappropriate AIs (and put them into an Amendment document for checking purposes), and update some of the introductory text in the RM.

He issued draft 2 of the RM with those approved AIs in it. Hopefully, that will cut down on repeated questions and make our current state clearer.

He has started working on the "justification" document for the approved AIs. He notes that many of the "qualifiers" explaining the purpose of Binding Interpretations are wrong. Most of the existing AIs are either fixing errors or adding clarifications, as it doesn't make a ton of sense to change a bunch of existing text and call the reason an omission. He will circulate this document well before our next meeting.

Randy asks whether we should include Usage and Terms & Defs in the Corrigendum. Doing so would add risk, and these aren't critical (they're better considered presentation improvements). That's especially true for the Terms and Definitions improvements, as those have lots of rules that are easy to get wrong. Tucker agrees that they should be left out for that reason; he doesn't think we want to add unnecessary risk here. No one objects to that plan.

Randy intends that all ARG approved AIs will be included in the draft RM, with AIs included in the Corrigendum labeled as /6, and AIs that are not included (waiting for a future revision) labeled as /7. That way, we'll get the benefit of all approved improvements whether or not they have been adopted at the ISO level.

### ***Unfinished Action Items***

There has not been much progress on AI22-0022-1. Jeff did not hear back on any of his suggestions for this AI. We reassign AI22-0022-1 to Jeff.

It is noted that John's updated book has finally been published.

### ***Current Action Items***

The combined unfinished old action items and new action items from the meeting are shown below.

Steve Baird:

- AI22-0076-1 (with help from Tucker Taft)
- Open a Github issue on the term normal (see discussion of AI22-0115-1 from meeting #63F)
- Create AI to describe the evaluation of aspect Relative\_Deadline (see discussion of AI22-0121-1).

Randy Brukardt:

- AI22-0101-1
- AI22-0123-1
- AI22-0124-1
- Open a discussion on ARG voting rights on the ARG mailing list.

Editorial changes only:

- AI22-0117-1
- AI22-0122-1
- AI22-0125-1

Jeff Cousins:

- AI22-0022-1

Gary Dismukes:

- Github Issue #12

Christoph Grein:

- Create an AI to make the example given in AI22-0121-1 into a Usage item for subclause 9.5.

Justin Squirek:

- ACATS C-Test(s) for filters

Tucker Taft:

- AI22-0063-1 (Split work with Richard Wai)
- AI22-0071-2
- AI22-0076-1 (assist Steve Baird)
- Review AI22-0075-1
- Github Issue #100

Richard Wai:

- AI22-0063-1 (Split work with Tucker Taft)

## ***Detailed Review***

The minutes cover detailed review of Ada 2022 AIs (AI22s). The AI22s are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the working Ada 202y AARM, the number refers to the text in draft 2 of the Ada 202y AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2022 AARM; again the paragraph numbers in the many drafts may vary.

## **Detailed Review of Ada 2022 AIs**

### **AI22-0022-1/04 Difficult example issues from WG 9 review**

We have assigned this to Jeff to rewrite, so we should let him do that before reviewing it.

### **AI22-0075-1/02 Explicitly Aliased Results**

Tucker explains the basic idea. He attempted to address the various issues noted in the minutes.

The issue becomes how to determine the accessibility level of the function result. It can return a global, or a part of a parameter. He has proposed rules for doing that.

Tucker includes some thoughts about tampering. Randy notes that such a thing interacts with Global (as these are often passed to a subprogram call, and the only way to check what is used in the call is via Global).

Tucker should review the AI as it has been 16 months since it was last worked on seriously. A careful, fresh reading might turn up some issues.

Keep Alive: 13-0-2. Christoph, Jean-Pierre abstain.

### **AI22-0076-1/02 Restricting Dynamic Accessibility Checks**

Tucker had cleaned up some errors in examples, but no work has been done. Steve is supposed to try to write wording. It is noted that the priority to do that should be raised.

### **AI22-0101-1/06 Valid\_Scalars attribute is defined**

Tucker explains the basic understanding of the attribute. The primary use case is to check the conversion of an externally sourced “bag of bits” into an Ada object. The “bag of bits” could come from IO like Sequential\_IO or Direct\_IO, it could come from a stream, it could come from an imported subprogram parameter or object, or from an instance of Unchecked\_Conversion. It could also be used to check objects after various other problems like writes through dangling pointers or interrupted assignments.

The basic idea is to define a “safe harbor” of objects where it is always (portably) safe to test with Valid\_Scalars. Evaluation of the Valid\_Scalars attribute will not by itself cause erroneous execution on such types, so analysis tools like SPARK can count on the attribute check to catch all of the possible problems. We call the safe harbor types “all-scalar composite types”, having (surprise) all scalar subcomponents ultimately, and various other restrictions such as no discriminant-dependent components and no arrays that aren’t statically constrained. The goal is to have a type that no implementation would ever try to implement with indirect or helper components.

Randy takes over and explains how he had to redo the way that abnormal is handled. For the most part, existing Ada says that when objects become abnormal is unspecified. But this helps no one, as abnormal is not a testable property, and thus all readers and analysis tools have to assume-the-worst, which is that all such objects are abnormal. Thus one of the fixes is to remove that uncertainty and just define exactly when objects are abnormal. After all, an object being abnormal does not require it to do something bad, it just means that it might cause an issue. An implementation is not required to erase the hard disk just because it evaluated an abnormal object!

As part of this redefinition, we define that objects of all-scalar composite types cannot become abnormal, since they can’t (or shouldn’t) contain anything that could cause a problem. To this end, Randy disallowed implementation-defined components in all-scalar composite types. Such types mostly would be used with extensive representation clauses anyway in order to match some external data structures; added junk would interfere with the intended use so no serious Ada compiler would include it. He did not want to simply say that an all-scalar composite type was one that does not have an implementation-defined component, as that would make the definition implementation-dependent and reduce the portability of the “safe harbor” to almost none (any implementation could simply say it had an implementation-defined component in any type it did not want to support in Valid\_Scalars).

Tucker would like to allow implementation-defined components to be allowed in tagged types. He thinks that a tag might be in several parts because of interfaces or nesting.

Randy asks the group if we want to include tagged types in the safe harbor. Tucker had suggested that we do not. Jean-Pierre concurs with Tucker's suggestion that we do not include tagged types. Randy notes that that will slightly simplify the wording.

We agree that allowing class-wide types is asking for trouble. It is too expensive to do it right (at least in many compilers that do not do bind-time elimination of unused tagged operations), and doing it wrong is inconsistent with the rest of Ada and very likely to cause mistakes. Users could build their own `Is_Valid_Scalars` dispatching operation that invoked the attribute on a parameter of a specific type to work around the missing operation.

Randy notes that `Unchecked_Unions` need to raise `Program_Error`. That is the usual response for a `Unchecked_Union` when an operation is applied that would need to read the non-existent discriminants. We can't in general, have a Legality Rule to prevent them, as they could be hidden behind a private type or a generic formal type. Making all of those things illegal would be a non-starter. He notes we could have a Legality Rule to make the obvious case illegal (as in GNAT), but Tucker objects, saying that we generally define operations on `Unchecked_Union` to raise `Program_Error`. There is no reason to be different here.

Jean-Pierre suggests that we just call this `Valid`, and only allow it on all-scalar composite types. Besides a vague concern about adding possible ambiguity (pretty unlikely), that would be different than the GNAT attribute we are using as a model. More importantly, we want to define this so implementations can do better. Outside of the safe harbor, execution is technically erroneous on bad values, but that doesn't prevent an implementation from doing the right thing. That could include checking tags for sanity, checking any gaps or implementation-defined components, and so on. And just correctly checking the scalar components can often catch problems. We agree to leave this a separate attribute.

Bob asks if this proposed definition matches what GNAT already does, with the exception of `Unchecked_Union` raising `Program_Error` and some legality changes. To the best of our knowledge, the answer is Yes (although it is hard to say what GNAT does in some corner cases; however, cases where differences might be detectable are very likely to be erroneous execution – at which point whatever happens is OK).

Randy summarizes the changes from the AI as written:

- No tagged types in all-scalar composites.
- Class-wide types and generic formals that might have a class-wide actual are not allowed as the nominal subtype of the prefix of the attribute.
- `Unchecked_Unions` raise `Program_Error` if given to the attribute.

This goes back to Randy for an update with these changes (some of the discussion will need updating as well).

Approve intent: 13-0-2. Christoph, John abstain.

### **AI22-0117-1/03 More Terms and Definitions**

We look at all proposed definitions and wordsmith several.

Approve AI with changes: 15-0-0.

### **AI22-0121-1/02 Internal calls in contracts of protected types**

Steve explains the problem and the solution. Essentially, we cannot allow internal calls in any context where they would be made when a protected action does not already exist for the object. We had previously noted that problem in preconditions and in default expressions, and Steve had uncovered two more places where the problem could occur.

While writing this up, Randy had noticed that the sentence describing when the `Relative_Deadline` aspect expression is evaluated is missing. He presumes it should read something like the similar sentence for `Priority` or `CPU`. We give Steve an action item to create an AI to add this missing rule.

Christoph would like the example of an alternative way to write such a predicate made into a Usage item for subclass 9.5. He is given an action item to create a simple AI to do that (we want it separate since we're not going to put Usage items into the Corrigendum, but this AI will go there).

Approve AI: 15-0-0.

### **AI22-0122-1/01 Order of finalization of components**

We discuss the wording. Randy notes that not all objects are “stand-alone”; that requires a specific form of declaration. Temporaries, for example are not.

We want the order of finalization of temporaries to be unspecified. That's true as they are not themselves declarations. But we should be explicit about that. Add (roughly) the “order is unspecified” to 7.6.1(13.1/6) as part of this AI.

Approve AI with changes: 15-0-0.

### **AI22-0123-1/03 Type of aggregates and raise expressions**

Randy suggests that we consider casting this fix as a Static Semantics (or perhaps Legality Rule). That is when we need to know the exact type, and not some earlier point. Tucker agrees.

Randy will revise the AI.

Keep alive: 15-0-0.

### **AI22-0124-1/02 Aggregates and capacity of bounded containers**

Randy explains the idea. Bounded containers have a capacity discriminant. For assignment operations (which include initialization of constrained objects), the discriminants have to match. But aggregate capacity is determined by the expected number of elements, not whatever subtype is appropriate for the target. If a user defines a constrained subtype to attempt to ensure that all objects are using the same capacity, the problem gets worse, as it happens in more contexts. Moreover, the usual tools to get the correct answer (qualifying the aggregate) just moves the exception earlier.

Tucker also reported a problem getting the correct capacity for an iterator. This is implementation-defined, but even if we defined it properly, it wouldn't help in most cases as noted previously.

What we need is to get the capacity we want to use from the context. To do that, we need a form of applicable discriminant constraint, which would work similarly to that for arrays (the famous applicable index constraint).

Randy proposes to generalize the idea of “applicable index constraint” into simply “applicable constraint”, and then use that term in 4.3.3 and 4.3.5 appropriately. That way, array aggregates and container aggregates would work similarly.

Jeff and Richard agree that this is good idea.

It is suggested that the name of the choice be `Capacity_Discriminant`, it's not necessary to repeat `Aggregate`. Randy notes that it was originally proposed to stand alone in the Github issue, but he realized that it made more sense as part of the `Aggregate` aspect, and moved it without reconsidering the name.

Tucker now suggests that the name of the choice be `Aggregate_Constraint`, or maybe `Discriminant_Constraint`. We want to allow the value to be used for other purposes by clever programmers, so we don't want to talk about capacities in the actual wording.

Steve asks if this interacts with `<>` or **others**. Those aren't allowed in container aggregates. Steve thinks that this could work with a record aggregate by giving the value for a `<>` discriminant. Perhaps, but that is a separate question and there is no intent to cover that. We're not looking for arbitrary cool features to add, especially if they don't have an obvious use.

Randy will take this back and develop wording.

Approve Intent: 14-0-1. Christoph abstains.

### **AI22-0125-1/01 Atomic\_Operations and checks**

Randy explains the idea. First, that checking the overflow bit can be done after the item is modified. This would allow using a hardware in-place atomic add with the check directly following (but still atomic). Second, that any predicate check is not atomic with the Add. We're not expecting any predicates on the associated type, but we have to say what it does if needed. Implementations may be able to do better, but generally only if they cannot use a hardware operations to guarantee atomicity (one can always implement it with a compare-and-swap loop, which allows evaluating an expression of fairly arbitrary complexity).

Tucker fixes some typos.

Approve AI with changes: 13-0-2. Gary, Jeff abstain.

### ***Detailed Review of Github Issues***

#### **#12 Iterator over vectors should preserve unconstrained state of elements of discriminated type**

The problem is that vectors and arrays act differently in this case.

Tucker notes that the aliased returns would provide a fix for this issue.

Gary suggests that we could add an aspect to an access type to fix this issue.

Gary will take an action item to look at that the idea of aspect to "behave as though the designated type has a partial view".

#### **#83 Full relational operators for ordered containers**

Randy argues that these operators don't necessarily make sense in this context, because equivalence is not the same as equality.

Richard says that these are based on the order of elements in the container (we're only talking about ordered containers), so there has to be a useful combination.

It is noted that "`<`" on cursors are based on key (for maps, or element for sets) comparison. And equivalence is not the same as "`=`" for keys.

We briefly discuss whether it would make more sense to define cursor ordering based on the position in the container. Perhaps, but then the operations that compare a cursor to a key would be different (and still could not have an "`=`").

[Editor's note: Also, these operations as defined do not require the cursors to come from the same container, only the same instance. That allows equivalent but unequal keys or elements to get into the mix.]

We're out of time, so we decide not to decide this one now.

**#100 Allow Add\_Named and Add\_Unnamed together**

Tucker explains he had a heterogeneous structure, and it made sense for the type to have both kinds of aggregates.

There doesn't seem to be much downside. Only a handful of kinds of iterators could happen for both kinds of aggregates, and it isn't obvious if there is an intuition as to how those work.

Tucker is given an action to write an AI on this.