# Minutes of the 38th ARG Meeting

12-14 June 2009

Brest, France

**Attendees**: Steve Baird, John Barnes, Randy Brukardt, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Tulio Vardanega (Friday only).

**Observers**: Greg Gicca, Matt Heaney (Friday only), Steve Michell (Friday only).

## Meeting Summary

The meeting convened on 12 June 2009 at 14:25 hours and adjourned at 12:00 hours on 14 June 2009. The meeting was held in a room on the campus of Telecom Bretagne on 12 June and in the meeting room of the Belvedere Hotel on 13-14 June. As expected, the meeting covered only about half of the entire agenda.

### AI Summary

The following AIs were approved with editorial changes:

AI05-0001-1/05 Bounded Containers and other container issues (6-0-2)
AI05-0102-1/02 Some implicit conversions ought to be illegal (7-0-0)
AI05-0107-1/02 A failed allocator need not leak memory (7-0-0)
AI05-0123-1/05 Composability of equality (7-0-0)
AI05-0130-1/02 Order of initialization/finalization of record extension (7-0-0)
AI05-0137-1/02 New conversion package (7-0-0)
AI05-0143-1/02 In Out parameters for functions (6-0-1) [conditional on future approval of AI05-0144-1]
AI05-0148-1/04 Accessibility of anonymous stand-alone objects (7-0-0)
AI05-0152-1/01 Restriction No_Anonymous_Allocators (6-0-0)
AI05-0156-1/01 Elaborate_All applies to bodies imported with limited with (7-0-0)

The intention of the following AIs was approved but they require a rewrite:

AI05-0136-1/04 Multiway tree container (6-0-1)
AI05-0142-4/03 Explicitly aliased parameters and accessors for Ada.Containers (6-0-0)
AI05-0144-1/03 Detecting dangerous order dependencies (6-1-0)
AI05-0146-1/02 Type invariants (6-0-1)
AI05-0147-1/02 Conditional expressions (6-0-0)
AI05-0149-1/04 Access types conversion and membership (6-0-0)
AI05-0155-1/01 'Size clause on type with non-static bounds (7-0-0)
AI05-0157-1/00 Unchecked_Deallocation should be illegal for zero-sized pools (7-0-0)
AI05-0159-1/01 Queue containers (7-0-0)

The following AIs were discussed and assigned to an editor:

AI05-0111-1/04 Specifying a pool on an allocator
AI05-0145-1/02 Pre- and Postconditions
AI05-0151-1/03 Allow incomplete types as parameter and result types
AI05-0153-1/01 User-defined constraints
AI05-0158-1/01 Generalized membership tests

The following SIs were approved with editorial changes:

SI99-0047-1/03 Editorial Review Corrections (7-0-0)
SI99-0048-1/01 Summary of changes (7-0-0)

## Detailed Minutes

### *Meeting Minutes*

John sent a bunch of typo corrections, they've been applied (but not posted). Do we want to look at them? Not in detail. Minutes were approved unanimously with changes.

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

The next meeting will be at SIGAda, Tampa, FL, USA, November 6-8, 2009. The hours were changed to make room for WG 23 annex work, so we will meet from Friday **afternoon** to Sunday **evening** (6 pm).

What about the planned February meeting? Steve Baird will look to host in Asilomar (Monterey area), he will check for possible dates. Tucker says that he doesn't have any obligations for February at this point (nor does anyone else).

### *WG 23 Ada Annex*

The HRG asks ARG and WG9 members that can spare the time to help prepare this annex, as well as the WG 23 workshop participants. There will be a workshop at the SIGAda meeting to finalize the work; this will precede the ARG meeting.

### *ASIS*

WG 9 approved forwarding the ASIS CD. Noting the problems that comments on WG 23's document had, it is important that comments from user bodies go through national bodies.

### *Thanks*

Thanks to Ada Europe for Friday's accommodations and lunch.

### *Old Action Items*

Brad did not do an update to AI05-0127-1. Randy did not do an update to AI05-0049-1, he got stuck creating AI05-0153-1 (user-defined constraints), and he ran out of time to revise AI05-0147-1 based on the voluminous e-mail. Bill Thomas sent an ASIS example after the deadline. Pascal did not propose new wording.

The accessibility subcommittee met by phone 4 times (2-3 hours a call) and clarified the issues surrounding a number of Amendment proposals, refining several AIs on a variety of topics. We agree that we should continue this work in between meetings, broadening the mandate to include most Amendment proposals. Randy suggests that we take the summer off, both to give his highly stressed budget a rest and also to allow for vacations and decompression time. We agree to restart the calls in September.

### *New Action Items*

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

Steve Baird:

- AI05-0135-1
- AI05-0153-1 (with Randy Brukardt)
- Investigate and possibly create an AI to require that streaming work between bounded and unbounded definite forms of the same container kind (see discussion of AI05-0001-1).
- Participate in the amendment subcommittee.

Randy Brukardt:

- AI05-0049-1
- AI05-0136-1

- AI05-0142-4
- AI05-0144-1 (with Tucker Taft)
- AI05-0147-1
- AI05-0153-1 (with Steve Baird)
- AI05-0157-1
- Create an AI to add missing rule about cursors after a Splice in List containers (see discussion of AI05-0136-1) [assigned AI05-0160-1 after the meeting].
- Participate in the amendment subcommittee.

Editorial changes only:

- AI05-0001-1
- AI05-0102-1
- AI05-0107-1
- AI05-0123-1
- AI05-0130-1
- AI05-0137-1
- AI05-0143-1
- AI05-0148-1
- AI05-0152-1
- AI05-0156-1
- SI99-0047-1
- SI99-0048-1
- Create SI out of the approved changes to SI99-0047-1 and SI99-0048-1 (assigned SI99-0055-1 after the meeting).

Bob Duff

- Participate in the amendment subcommittee.

Matthew Heaney:

- AI05-0159-1

Pascal Leroy:

- Create AI to combine wording in AI05-0006-1 with other parts of the language (from his editorial review).

Brad Moore:

- AI05-0127-1

Jean-Pierre Rosen:

- Create an AI for restriction No_Default_Stream_Attributes (see discussion of AI05-0152-1). [Assigned AI05-0161-1 after the meeting.]

Ed Schonberg:

- AI05-0158-1
- Participate in the amendment subcommittee.

Tucker Taft:

- AI05-0111-1

- AI05-0144-1 (with Randy Brukardt)
- AI05-0145-1
- AI05-0146-1
- AI05-0149-1
- AI05-0151-1
- AI05-0155-1
- Create an AI for the syntax of aspect specification (see discussion of AI05-0145-1).
- Create an AI for restriction No_Allocators_After_Elaboration (see discussion of AI05-0152-1).
- Create an AI for restriction No_Allocation_From_Default_Pools (see discussion of AI05-0152-1).
- Participate in the amendment subcommittee.

## *Detailed Review*

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2005 AIs. The AIs and SIs 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 Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

### *Detailed Review of ASIS Issues*

#### SI99-0047-1/03 Editorial Review Corrections

On Friday, we assign portions of this SI to each ARG member in order to make a quick review, two members per portion. On Saturday, we look at the resulting reviews.

Randy explains that we will have to make any corrections in a separate SI, so that we keep changes made to the ASIS CD after its ballot separate.

Erhard looked at the actual standard, not the SI. Groans all around. We look at Erhard's comments:

1.1.4 (new) "suppressed" should say "detected" (two places).

"Implementation overloaded" -> "Capacity exceeded"

1.2 "standard" -> "standards" throughout. (There is more than one standard.)

2.2.3.4: The description of Hash is junk, but so is 13.56. 13.56 is an open comment recorded in SI99-00053-1; so add this one there.

Steve Baird's comments:

1.1.2: OK. 1.1.3.3: OK (ASIS assumes an exception is raised for unimplemented operations).

1.1.5 (now 1.1.4): "ASIS applications are encouraged to similarly follow the convention of recording a Status and Diagnosis prior to explicitly raising any ASIS exception.

2.2.1: What the heck is "reverse bottom-up"?Re-word to "Structural queries are those ASIS queries which provide information about the syntactic structure of the compilation unit."

Change the glossary entry for ASIS queries similarly.

2.2.3: OK as is (no change).

2.2.3.1: "hierarchical tree" -> "abstract syntax tree".

John's comments:

2.3: Need a square bracket closing the deletion. (Fix SI99-0047-1, as this is a typo in the SI.)

"packages units" -> "packages".

2.4.1: "Object_Views" wraps in the PDF, fix the formatting.

2.4.3: In the paragraph starting with "The ASIS application is almost complete...", add "to" in front of the other verbs (they all should start with "to" like the new one).

3.9.19: Should be "{The} type Attribute_Kinds". Same thing in 3.9.4.

Jean-Pierre's comments:

3.10.4: A library_unit_body has as a Supporter[,] its corresponding library_unit_declaration, if any.

8.6: "physical {effect}[affect]"

6.8: "Parameters specifies" (because the identifier of the singular parameter is Parameters).

Ed's comments:

10.19 and 10.20 "is given" bothers him. Replace with: "returns a null string if Compliation_Unit is ..." Check rest of standard for similar issues.

11.2: "the time {at which}[that] this ..."

13.39 => 15.39 (typo in SI99-0047-1).

17.2: Value_Image {includes}[returns]

17.4, 17.5: "the limits for the query which is limited to the element..." better to say "the limits for the query which is restricted to the element..."

18.28: Corresponding_Called_Entity (typo in SI99-0047-1)

Tucker's comments

13.1:Returns a Nil_Element_List if {the Unit_Kind of} the unit is A_Nonexistent_Declaration, A_Nonexistent_Body, or An_Unknown_Unit.

13.3: " unit [has]{is} A_Nonexistent_Declaration…"

13.31, 13.32, 22.11, 22.12: drop "variable". (Check standard for this change.)

17.5: drop "only" from "only if".

18.28: move "the" from the lead-in to start each of the bullets: "The prefix", etc.

20.24: (now 20.22): "may" should be "might".

21.1: "A Id" should be "An Id".

21.6: "{If True, } the two Ids...

Annex A: Fix Needed Units: drop "to make up or".

C.3.3 (now C.2.3): Start with "To use ASIS in a CORBA environment, ..."

Approve this SI and approve the SI (assigned SI99-0055-1 after the meeting) with the previously noted changes: 7-0-0.

### SI99-0048-1/01 Summary of changes

Drop "to ASIS" from the bullet starting "Added support".

"...an element {that is}[representing] a name represents..."

As noted for SI99-0047-1, we'll need to make these changes in a separate SI as they will be changes to the ASIS CD; they might as well be included with the SI created for SI99-0047-1.

Approve this SI and approve the SI (assigned SI99-0055-1 after the meeting) with the previously noted changes: 7-0-0.

## *Detailed Review of Ada 2005 AIs*

### AI05-0001-1/05 Bounded Containers and other container issues

"task-safe queues" should be dropped from the proposal section (it is now in AI05-0159-1).

Does Copy for an unbounded hashed map have a capacity parameter? Matt says that capacity means different things for the unbounded and bounded forms: it means the size of the hash table for Hashed_Map (unbounded), while for bounded forms it means number of elements. In that case, it might be dangerous to have the parameter. Tucker disagrees with the basic premise; he says that by the formal definition in A.18.5 capacity means the number of nodes in the container for the unbounded form, which is the same as the bounded form.

So for the unbounded hashed map and sets, there should be a Capacity parameter for Copy (just as there is a Reserve_Capacity for them).

[Editor's note: after the meeting, a couple of minor issues with the wording were noted and corrected:

- The new wording for Move in A.18.4 and A.18.7 depends on Assign, but Assign is not defined there. Since the wording is the same for all of the cases anyway, Assign should be defined in A.18.4 and A.18.7 and the definition was moved there.

- Quite a bit of the wording said "Capacity_Error is raised" rather than the correct "Capacity_Error is propagated" for cases where exceptions are raised by subprograms. (OTOH, "raised" is used when talking about checks.)

Should the unbounded hash forms have Default_Modulus? No, only in the bounded forms. (So move that routine to the bounded forms for hashed maps and hashed sets.)

In the last bullet of A.18.18, drop the word "only". Add an "AARM Implementation Note" that says that "naturally" you would stream the entire capacity, which we don't want.

Tucker would like to move the two bullets about Implementation Advice into their own IA section (for each bounded form). "Bounded vector objects should be implemented without implicit pointers or dynamic allocation. The implementation advice for procedure Move to minimize copying does not apply."

A.18.16: "sorted in the ordering determined by the generic formal Before function".

"{The} generic formal Before and {the} generic formal Swap ..."

[Editor's note: The definition of "strict weak ordering" needs to be broadened in order to cover formal functions that are not operators. The current text appears to only apply to operators.]

Does this subclause ordering make sense? The array sorting is in the middle. Should we change some of the indefinite container clause numbers? No, but we will move Array sorting to the very end (which will change its clause number), and then put the bounded containers after the indefinite ones.

Drop the extra blank line from the generic Generic_Sort.

"is called with {index values that identify} a particular pair of element values."

Add "The actual function for the generic formal Swap should exchange the values of the indicated elements. If the actual for either Before or Swap behaves in some other manner, the behavior of Generic_Sort is unspecified. ..."

Tucker wonders if the definition of 'Write and 'Read shouldn't be mentioned for all containers. Yes (the definition of what to stream really applies to all kinds of containers), so move the wording for 'Write and 'Read to the unbounded forms. That should be added immediately after the definition of Empty_Vector (and so on) to be consistent with the discussion of streaming for Cursor.

Should we require that you can stream the bounded form out and read it in a similar unbounded form? (Or vice versa.)

There might be a compatibility issue with such a requirement. Matt is pretty sure that the existing GNAT implementation would have a problem with that requirement. Someone says that such incompatibilities are exactly why we need a rule to prevent implementations from being lazy.

But Matt says that the requirements of the language make that problematical. For instance, the indefinite forms require an extra bit because streaming of empty elements is required to work. He wants to except them from any requirements. Randy worries that we would be preventing an indirect element implementation for unbounded definite forms (we have lots of wording to allow such an implementation to date); it would have similar problems with empty elements. This is a problem for all forms of all containers that have empty elements.

We discuss empty elements (they are mainly an issue for the vector containers); A.18.2(238.b) points out that you have to be able to stream empty elements. We don't want to require streaming their contents as they don't necessarily exist, we surely don't want to make an implementation requirement that they exist. Even if they do exist, they could be filled with invalid garbage; streaming that out probably would work, but streaming it back in would result in an abnormal value, and accessing that would be erroneous. We surely don't want that. Steve Baird will take a new AI to see if some sort of requirement for streaming of bounded vs. unbounded definite can be done.

Jean-Pierre notices a typo in third bullet in A.18.19 "...if and only {if}...". Get rid of the two does nots from that sentence. "The type List needs finalization if and only type Element_Type needs finalization." All of these clauses have this same issue and should be corrected the same way.

In A.18.19, "In {the} three-parameter procedure Splice whose Source has type List, if the sum of [the] Target length and Source length is greater than [the] Target capacity, then Splice raises Capacity_Error." A similar change should be made for the four parameter version.

Approve AI with changes: 6-0-2.

### AI05-0102-1/02 Some implicit conversions ought to be illegal

Put this after 8.6(27), it is more important than some of the others at the end.

Should 3.7(16), 3.7.1(9), 6.4.1(6) be removed? These are the same rule, just scattered about, so delete all of these. These were the only cases for Ada 95, but Ada 2005 has many more cases. The discussion should say that.

Approve AI with changes: 7-0-0.

### AI05-0107-1/02 A failed allocator need not leak memory

Move the second paragraph of the editor's note into the Appendix so there is a record for not doing the change (but there is no problem with not making this change). "has" -> "have".

In first sentence of Implementation Requirements, move "only" to after the word "implementation" (Tucker channeling Bob Duff).

Steve worries that this wording does not allow calling Deallocate to kill off the objects allocated by a local access type. That seems necessary if the pool outlives the access type. Add "as part of the finalization of the collection for T".

"The Deallocate procedure of a user-defined storage pool object P may be called by the implementation to deallocate storage for a type T whose pool is P only at the places when an Allocate call is allowed for P, during the execution of an instance of Unchecked_Deallocation for T, or as part of the finalization of the collection of T."

Erhard wonders why we are not mandating cleanup. The current rules don't require it, and besides it can be very expensive to implement [depending on how an implementation implements exception handling] for a very rare case (really a bug).

"For such a call of Deallocate, P (T'Storage_Pool) is passed as the Pool parameter. The value of the Storage_Address parameter for a call to Deallocate is the value returned in the Storage_Address parameter of the corresponding {successful call to} Allocate [call]."

Approve AI with changes: 7-0-0.

### AI05-0111-1/04 Specifying a pool on an allocator

Tucker gives a presentation on his current proposal; this was the same talk that he gave at the conference.

His proposal has static and dynamic handles, which point between regions (subpools). Handles are immutable; they only go away when the scope or region goes away. Regions are reference-counted, and are only recovered when all of the handles disappear.

Erhard wonders about the cost of pointer checks.

A goal is that walking a data structure is safe and there is no extra overhead (that is, there is no overhead when reading access values). Overhead occurs when you store access values.

Tucker’s proposal also has strong and weak links. These are managed access types that can be converted to a real access value and an appropriate dynamic handle. Management is generally in the hands of the user.

This sounds expensive, and there is concern about whether cycles can be detected reasonably.

At this point, there is neither wording changes (mostly in terms of new operations in language-defined packages) nor any implementation model. We can't go much further without either of those things.

Tucker will work on an implementation model for this approach, and will suggest the form of handles and links (in terms of the operations that manipulate them).

Keep AI alive: 7-0-0.

### AI05-0123-1/05 Composability of equality

3rd paragraph of proposal:

"...same answer [aa]{as} a corresponding call to the [the] predefined..."

Why doesn't this work on arrays? Array (of elementary) have a number of additional operations which would not compose/reemerge. This would be nasty, especially giving inconsistent results for ">=" and "=" inside of generics. Arrays of records will compose (as the components will compose properly); only user-defined array "=" would not compose.

Erhard suggests getting rid of "(tagged or untagged)" (found in several places in the wording), because it doesn't add anything and just points out that we fixed this.

The first wording should go after 4.5.2(9.7/2), in the existing legality rules for equality.

Ed notes that this was fairly rare in GNAT test programs, and several of the occurrences assumed that "=" would compose.

Summary has an extra space before the "."

Drop the first paragraph of the discussion. Move most of the !proposal to the !discussion, and get rid of at least part of the existing discussion.

!problem "User-defined" should be "user-defined".

John "Op" should usually be spelled out as "operator".

Approve AI with changes: 7-0-0.

### AI05-0130-1/02 Order of initialization/finalization of record extension

"choice one set" -> "choose one set".

We recommend not writing an ACATS test for this issue. It is likely to be a significant cost on compilers.

Approve AI with changes: 7-0-0.

### AI05-0136-1/04 Multiway tree container

Randy explains what he did since the last meeting. He added the idea of a designated root node; all other top-level nodes are called orphans.

We turn to the open issues noted in the AI:

Should we have a maximum depth function? No, no big advantage over writing it yourself using one of the iterators.

Should we have the sibling routines Matt had suggested? No, they are not sufficiently useful, and would make the interface even bigger.

We discuss options for the No_Parent constant, and the idea of orphans. Tucker wonders if we could name things top-level rather than orphan. The discussion continues for quite a while, getting rather contentious, without much progress. Then Tucker suggests that container contains a root node that always exists and has no value. We'd use that instead of No_Parent.

So the No_Parent constant becomes a Root function. You cannot put a value into the Root – an exception is propagated if you try. With this change, we no longer need orphans, the primary node, or any of the operations related to that. In particular, we can get rid of the xxx/Overall_xxx bifurcation. Randy notes that is essentially what he was suggesting as an alternative, with different terminology. He is told that sometimes, terminology makes all the difference.

Randy asks (as an aside) whether the doubly linked list Splice from one list to another, invalidates any cursors that point into the part that is moved. It seems that it must be true, since cursors (implicitly) contain a reference to the container. Matt agrees that must be the case. Randy notes that we're missing wording to that effect, he takes an action item to create an AI to fix that.

Approve intent of AI: 6-0-1.

### AI05-0137-1/02 New conversion package

The subject needs to be more descriptive, something like "String encoding package". The clause number A.4.10 was already taken by AI05-0001-1.

Line for BOM_8 needs to be wrapped.

Tucker would like a user note to explain how the BOM ought to be used.

Tucker mentions again that "shall" is requirements on the user (legality rules); we just use "is" for requirements on the implementation. So:

The Encode functions [shall]{do} not put BOM sequences in the result.

For UTF_8 no overlong encoding {is}[shall be] returned. The lower bound [shall be]{is} 1.

The implementation permissions should not use "shall". The second one should be IA. (but they're removed later, see below).

Second paragraph: "8 bits encoding" -> "8-bit encoding". Same for 16 bits.

Tucker suggests add "No_Encoding_Scheme" to the enumeration, and two functions to query the encoding scheme of text. Erhard suggests naming that new literal UTF_None; it should be placed at the start of the enumeration.

```
function Encoding (Item : in String) return Encoding_Scheme;

function Encoding (Item : in Wide_String) return Encoding_Scheme;
```

For each of the Encoding functions, if the initial characters of Item match a BOM, the corresponding encoding is returned; otherwise, UTF_None is returned.

Remove the Implementation Permission (both paragraphs). Add an Implementation Advice "If an implementation supports other encoding schemes, another similar child of Ada.Strings should be defined."

Rename package to UTF_Encoding.

Approve AI with changes: 7-0-0.

### AI05-0142-4/03 Explicitly aliased parameters and accessors for Ada.Containers

Randy tries to explain this and fails miserably. Tucker takes over and succeeds in explaining it more clearly.

The wording 3.10.2(6-16) seems to require a time machine. Randy says that the wording it depends on is long so he was trying to reuse it; maybe we need to define a term and reuse it that way. Randy and Tucker will take this off-line. There should be "(see 6.1)" in this wording for "explicitly aliased".

Steve says that we need to say something to exclude these from the by-copy rule "A non explicitly aliased parameter of a by-copy type..."

There is a stray + in the Language Design Principles.

Tucker wants to verify that the accessibility rules are correct. He wonders if there needs to be a requirement to change the lifetime of the temporary for a function call used directly as an aliased parameter. Randy says that there already is such a rule (for functions), it doesn't seem necessary for procedures.

The name of Accessor_Type and its discriminants seem OK. Randy specifically asks about Bob's suggestion to use 'E' as the name of the discriminant; the group prefers to use "Element".

Tucker says that read-only accessors are important. We should have them. What should the name of the type be? Read_Only_Accessor_Type, Constant_Accessor_Type, Constant_Accessor?

Possibly call this "Reference" instead. That seems good. "Reference" and "Constant_Reference", "Reference_Type" and "Constant_Reference_Type".

Should this be called Query and Update (to be consistent with the existing routines)? No, because this just grants the access to do that.

"The default initialization of an object of type Reference_Type propagates Program_Error."

Approve intent of AI: 6-0-0.

### AI05-0143-1/02 In Out parameters for functions

Tucker says that he wants AI-144 to feel comfortable. We agree to this proviso.

Re-word 6.6(3):

The subprogram_specification of a unary or binary operator shall have one or two parameters, respectively. The parameters shall be of mode in. A generic function instantiation whose designator is an operator_symbol is only allowed if the specification of the generic function has the corresponding number of parameters of mode in and no other parameters.

Approve AI with changes: 6-0-1 (conditional on approval of AI05-0144-1).

### AI05-0144-1/03 Detecting dangerous order dependencies

Tucker had sent Randy "improved" wording. Randy wasn't able to determine if that wording was in fact correct because of lack of time. We discuss Tucker's wording.

Tucker added the terminology "known to refer to the same object", these are generally components of a larger object.

We don’t want false positives in these rules, that would make people really aggravated. So we are defining objects that are the "same", and then checking for misuse of those cases.

There are two separate issues, one is a single call with two **in out** parameters, and the other is with function calls with **in out** parameters used as actual parameters of enclosing calls (or as expressions in another entity with arbitrary ordering, such as an aggregate).

Tucker would prefer to only make these checks for **in out**, as the access parameters case would be incompatible and people know that access types are dangerous.

Erhard would like the first rule to apply to all types as well (it is proposed to apply only to by-copy types). He is worried about the user view. 6.2(12) defines cases where the semantics is weakly defined; for by-reference parameters the language has well defined semantics.

Randy notes that the point of these rules is to detect non-portable code, by-reference types are well-defined and *are* portable.

Someone asks Tucker why he feels this is so important. He wants this sort of check because he doesn't want to make the language less safe. Ed points out that making the language less safe is impossible since the language already allows anything. Tucker insists that this is a new case, more insidious than the existing ones of hidden side-effects. He argues that a routine that uses a side-effect internally is (or ought to be) written so that the side-effect doesn't damage the correctness of the function. People rarely write functions that can only be called once. On the other hand, a side effect occurring in a call could not be known to the author of the function and may not be known to the author of the call either. This is code that is clearly non-portable, and is likely to break on a different compiler (or different compiler version, or even different optimization settings). There is little value to allowing it.

Steve Baird wonders whether there needs to be rules for **and then**; no because the order of evaluation is specified for that.

Jean-Pierre Rosen suggests changing "elementary" to "not known to be by-reference" in the first rule. That would check everything unless it is known not to be a problem. But "by-reference" breaks privacy. Possibly "untagged and (not immutably limited)". Or define "known to be by-reference" to avoid privacy breaking.

Is there a generic contract problem with this change? No, the generic would be illegal in interesting cases so whether the instance should be legal is not a problem. If the formal is tagged or immutably limited, then the actual also has to be tagged or immutably limited. (This is the rare case where the rules get weaker [more is allowed], not stronger, as more information is available.)

Steve Baird notes that <> and default expressions need to be covered by these rules (as if they are included literally in the call/aggregate).

Approve intent of AI: (without access parameters, copy-back is checked for everything except tagged/immutably limited) 6-1-0 (Erhard continues to object to allowing any kinds of parameters in the copy-back check.)

### AI05-0145-1/02 Pre- and Postconditions

Should there be a separate AI to define the syntax for aspect specification? Yes, assign that to Tucker. Remove that from this AI and AI05-0146-1.

When referring to both, say "Pre/postconditions". Change the whole AI (including the subject) to reflect this.

Tucker suggests Pre'Class and Post'Class instead of Inherited_Pre.

Steve Baird asks what are you allowed to put on abstract subprograms? Just Pre'Class and Post'Class, the others don't make sense since you can never have a call on an abstract subprogram. Does the AI say that anywhere? No; fix that.

It should be illegal to write Pre and Post for abstract types; otherwise a programmer could forget the 'Class and the precondition would not be inherited.

If multiple interfaces have Pre'Class, those are **or**ed together, Post'Class is the reverse (**and**ed together).

Non-primitive subprograms should not allow Pre'Class. What about inherited non-dispatching operations? (For instance, derived integers?) The *inherited* routines get the same Pre and Post, *overridden* ones override the Pre and Post.

Pre'Class and Post'Class are only allowed on primitive operations of tagged types.

Pre and Post belong to a particular body.

Names are resolved at the end of declaration list where the subprograms declared (or at a freeze everybody point, if earlier).

These expressions cause freezing like a default expression (at the point of use).

These expressions allow references to things declared later in the same declaration list. If we make premature calls, we could be freezing items that aren't declared yet. Yuck. Such calls pretty much have to raise Program_Error, we might as well make that illegal. This sounds complicated (we don't try to make calls to unelaborated routines illegal now, for good reason).

Tucker will tackle this problem and update the AI accordingly.

These are not allowed on instantiations or renames.

We don't require a place of evaluation, but we want to always assume that it can be evaluated in the body. The exception cannot be handled within the subprogram.

The invariant is that the postcondition is true or an exception is propagated. That would not allow local handling – otherwise it might be possible to return normally with the postcondition being false.

Keep AI alive: 6-0-1.

### AI05-0146-1/02 Type invariants

Steve Baird notes that stream attributes should be covered by type invariants, because they are visible outside of the package.

Steve asks if the "current instance" is constant or variable. Constant of course is the answer.

John notes that this needs rewriting to support **in out** parameters for functions, which we previously decided to allow.

Make the function result/type conversions cases more visible; make those cases into sub-bullets along with the existing procedure call case.

Should these be inherited by an untagged derived type? It would seem to be annoying for them not to, especially if these are generated in the body. Tucker says that the existing rules imply a conversion to the original type, which would trigger the invariant.

But should it be inherited for new subprograms and for overridden routines? No, that seems weird. So, it probably should disappear upon inheritance.

Invariants pretty much have to be a function call (no components are visible, as these allowed only on private types – the only visible thing is any discriminants). That means it is OK for them to disappear; there would be a convinient name to call to create a new invariant for the derived type.

Drop "with the specified message if any," text, since we got rid of the messages long ago.

Approve intent of AI: 6-0-1.

### AI05-0147-1/02 Conditional expressions

Randy notes that the AI is not updated with the results of a lot of the mail.

Tucker says that he has given up on the optional parenthesis. They cause readability and parsing issues.

Steve says that we want this to work more like parentheses and not like a function call. In particular, there should be no temporary created here for build-in-place types.

Tucker comments that one use for this is to be able to choose which object is being renamed (especially valuable if the object is limited). That can't be done currently, but this would allow it. Add that as a justification to the AI.

Erhard does not like the optional **else** part. He is worried about confusion about the value of the else. Jean-Pierre says that the natural choice is "**else** False". But that is *not* what we want. So it seems confusing. No conclusion here.

The third example in the problem section has an extra close parenthesis.

The call of Q in the discussion section is missing an open parenthesis at the beginning of the call.

"float" should be capitalized in the parameter in the definition of Q.

4.9(33) has extra words, but it needs to be rewritten anyway (it needs to be broader).

Move condition from 5.3 to 4.5.7 to avoid a forward reference.

Erhard wonders what the problem is with omitting the parentheses. Randy answers that there are problems with error detection if you don't have the parentheses; if you leave out the ); at the end of a call, a following if statement could turn into a conditional expression and the error would only be detected at the end of that statement, far away from the actual error. It also harms readability.

Steve worries about class-wide issues. If one result is dynamically tagged and the other result is statically tagged, there is an issue that needs to be resolved. We generally don't allow dynamically tagged items in non-dispatching calls of specific types.

Randy reminds everyone that this is not really done, we should defer detailed work until the AI has been fully updated.

Approve intent of AI: 6-0-0.

### AI05-0148-1/04 Accessibility of anonymous stand-alone objects

Tucker justifies this AI. He notes that you can use the same implementation model as done for access parameters (you have to "normalize" the static level). In particular, a single numeric level works; things that are not statically visible should appear to be local.

Typos: "folowing" should be "following". Right before !wording, "libary" should be "library". " volumnous" should be "voluminous" at the start of the !appendix.

Ed worries about the "flow analysis" mentioned in the Implementation Issues part. Tucker says that this is very simple, as these are just integer values. The run-time check would be essentially equivalent to that of an access parameter.

Take the "Implementation Issues" and turn it into a large AARM implementation note, as the model should be explained. (The normative wording doesn't explain much for access parameters, either.)

[Editor's note: The last sentence in the new text for 4.6(48) can be read as applying all the time, not just in the special case. It was connected to the previous sentence with "; if the check succeeds".]

Approve AI with changes: 7-0-0.

### AI05-0149-1/04 Access types conversion and membership

This AI contains three related proposals; they all change overlapping wording.

Steve worries that two different named access types designating the same class-wide type could get implicitly converted. He thinks that's a problem because if someone declares two different identical types, they probably really want conversions, there is little reason to do so otherwise.

We consider disallowing the implicit conversion if the designated types are the same. Randy says that is ugly, and it interferes with the limited with usages. If you import a limited view of a type, we might be forced to declare a new named type that we really don't want to be treated separately.

We definitely need to add this to the discussion.

Most of us are not really excited by this idea, but the O-O people (especially coming from other languages) really insist that they need it. Erhard chimes in to say that he has wanted these access-to-class-wide conversions for the last 5 years.

We study the wording for the membership. The new text is the third inner bullet.

Add words to the fourth and fifth bullet of 8.6: "{whose} designated type {is} a specific type".

The paragraph added after 8.6(27) is completely new.

Steve writes an example:

```
type T1 is access all D'Class;
type T2 is access all D'Class;

A, B : T1 := ...;

A_eq_B : Boolean := A = B; -- Ambigious (T1."=", T2."=", Standard."=")
```

Randy notes that this seems to happen for anonymous access as well (if there is an appropriate named type around):

```
X, Y : access D;
type T3 is access all D;

X_eq_Y : Boolean := X = Y; -- Ambigious (T3."=", Standard."=")
```

We can have a preference rule to use Standard."=" in the latter case (both anonymous access), but the named to named comparison is unfixable without introducing a Beaujolais effect (we can only have preference rules for things that are always visible (that is, entities in Standard)). (A preference for Standard."=" over the user-defined "=" for a specific type would work semantically, but would be very surprising and incompatible – effectively, user-defined "=" would be ignored.) So you can't call "=" at all if there is more than one type. That's not going to work. We quickly lose interest in proposal 3.

A preference rule to avoid ambiguity with "=" will have to be crafted for proposal 2.

Approve (new) intent of the AI: 6-0-0.

### AI05-0151-1/03 Allow incomplete types as parameter and result types

This proposal allows more usage of types imported from a **limited with** package. It means that there will be less need to introduce access types just because **limited with** is involved.

Having parameters of incomplete types is required for access-to-subprogram types, so compilers have to be able to deal with the possibility already. Of course, this possibility will be far more common, meaning that getting the implementation right is very important (while it is not very important now).

This basic idea seems appealing, as it is removing restrictions. But the proposal still needs detailed wording. Since the devil is always in the details, we can't say much more about this now.

Keep AI alive: 6-0-0.

### AI05-0152-1/01 Restriction No_Anonymous_Allocators

This should be placed after H.4(8/1).

Jean-Pierre will submit a restriction No_Default_Elementary_Stream_Attributes. The Swiss asked for a restriction No_Allocators_After_Elaboration. Tucker wonders if we should have No_Allocation_From_Default_Pools.

All of these should be separate AIs. Tucker will handle the last two.

Approve AI with change: 6-0-0.

### AI05-0153-1/01 User-defined constraints

Randy tries to explain how he got stuck. He was concerned that we would have "constraints" that could easily become untrue. He originally noticed the problem for constrained access types.

Someone wonders why we would want to bother with this at all. Tucker says that one of the uses for this is part of a precondition that occurs on many subprograms. You would like to be able give that portion of the precondition a name.

Tucker suggests that we don't allow these on access types. Randy says the problem still happens even for simple stand-alone objects.

```
type Rec is
   A, B : Natural;
end record;

subtype Equal is Rec when (A = B);

Obj : Equal := (A => 1, B => 1);

Obj.A := 2; -- Constraint no longer holds.
```

Trying to check whenever any part is changed seems like a nightmare. And it doesn't work sensibly when a part is passed by-reference to a subprogram – we surely don't want to have to pass a checking thunk with all by-reference parameters. Checking after the call is too late: the constraint is already violated (and a handler could see that).

Steve Baird suggests limiting the use of these to constant views and in a membership (constant views include in formal parameters, function results, and qualified expression).

These should be usable for scalar subtypes; they should just be **and**ed together with any ranges.

The constant view idea seems better, because it is too weird to allow on variables: we want the checks to stay in the normal places.

Syntax: `with Constraint => *Boolean*_expression;`

Tucker thinks this isn't really a constraint.

Randy will try again on this line (of constraints). Steve will try taking a stab at defining a "named predicate" this way.

### AI05-0155-1/01 'Size clause on type with non-static bounds

For non-static bounds, you have to support confirming clauses, but nothing else. 13.3(55) only applies to static subtypes. There is no requirement to support very large sizes.

The example in the question is not declaring a static subtype, so there is no major requirement here.

Pak3 is illegal as written (the bounds need to be static); a derived type needs to be used and then it is essentially the same as Pak2. Randy is dubious about this; in Pak2, we know that the parent subtype uses 32 bits. But we know nothing at all about the parent subtype of Pak3 – do we just assume the worst?

"Must represent all values" is always required, so the first example (Pak1) is always illegal. The second example (Pak2) is an implementation choice ("must represent all values" is satisfied), but whether the size clause is accepted is up to the implementation.

Tucker will write this up as a ramification.

For generics, type-related representation items aren't allowed, but that does allow Size and Alignment. But again, they aren't static and no requirements are made other than confirming representation clauses.

Randy asks about similar examples involving record components in a record representation clause, but no one hears him.

Approve intent of AI: 7-0-0.

### AI05-0156-1/01 Elaborate_All applies to bodies imported with limited with

"A string reading" -> "A strict reading".

"...which is {a} tough constraint…"

"...likelyhood..." -> "...likelihood..."

No one seems interested in repairing the wording to fix what is essentially a pathology.

Approve AI with changes: 7-0-0.

### AI05-0157-1/00 Unchecked_Deallocation should be illegal for zero-sized pools

You can't call an allocator for such a type, so why can you instantiate unchecked_deallocation for it?

Tucker is worried about Unchecked_Deallocation in a generic spec that is not actually called anywhere in the generic. The recheck of legality rules in an instance would catch this (if the actual type had Storage_Size = 0). He claims that an allocator in the same spec is unlikely, so such a generic would currently be legal.

He says that he would rather raise Program_Error on any call.

Many do not like turning what is otherwise a compile-time error into a run-time one.

We decide to continue discussing this on Sunday (it's time to catch the bus to town for dinner).

Tucker suggests that we make the *call* on Unchecked_Deallocation illegal if Storage_Size is statically 0. This avoids the unused generic instance problem noted yesterday.

We also could have Implementation Advice that Program_Error should be raised if it is called with a pool where Storage_Size is 0 (not statically known). (Such an allocator would raise Storage_Error, so it seems strange to allow the deallocator.)

Randy will write up the AI that way (it is not written up now).

Approve intent of AI: 7-0-0.

### AI05-0158-1/01 Generalized membership tests

Ed suggests allowing this syntax on all types – the contents could be a bunch of variables.

What equality is used in that case? Right now, we use the predefined operations (which is appropriate for scalar types). For record types, we would want to use primitive "=" (see our earlier vote). We would need wording to make that clear.

Several people want to parenthesize the choice list to avoid ambiguity. `A in (1..3 | 7)`, and the contents requires at least one bar. Tucker doesn't like this idea.

Ed says we could use this syntax in loops as well.

The idea is that this is a new kind of thing called a "Set". Tucker still finds that is weird.

```
V in (A | 1 .. 10)

for I in (1..100|-1) loop
```

These aren't sets, they have order and allow duplicates. So these are sequences not sets. Tucker would still prefer using a choice_list here:

```
if X in 1 .. 10 | 33

if X in A .. B | C .. D
```

These are not exclusive, not static, not only discrete. So they definitely are not a choice_list (which are all of those things). It should just be called a sequence.

The equals is the one you get in a generic (predefined for scalar, primitive for record). Definitely will need wording to that effect.

Ed says that these just expands to a list of "or else" with X evaluated once.

Is there an ambiguity here (without the parentheses)?

```
X in A or B or C  ==>   (X in A) or B or C

X in A | B or C   ==>   (X in A | B) or C
```

Tucker points out that we already have this issue in the language:

```
X in A .. B or C  ==>   (X in A .. B) or C
```

We get into a pointless discussion of precedence rules.

Case statements are ambiguous, Tucker again suggests limiting choice lists to simple_expression, but that breaks "**and**" and "**or**" in modular operations (such as the flags for Windows). Randy shows an example:

```
case Flags is
   when Bordered or Menued => ...
   when Bordered => ...
   when Menued => ...
   when others => ...
end case;
```

There is no problem if we require the parentheses. Tucker still hates adding parentheses in memberships. We're not making any progress on this issue.

We will allow any type, equals as noted above.

Straw poll: "sequence syntax" (parenthesizing the sequence in a membership): 5-0-3.

A sequence must have one vertical bar. But Randy objects to that, you would like to be consistent, and a range is a valid sequence. A single item would be a weird way to write an equality check. But a range makes sense as a sequence, but it would not allow parentheses, which is obnoxious. It would be very annoying to be required or allowed always except one case. Most are not convinced. Someone recalls an old bug in a now defunct compiler where such parentheses were allowed. Randy notes that RR Software got several "bug" reports about not accepting that incorrect syntax that the other compiler allowed.

No firm decision is made on this point.

### AI05-0159-1/01 Queue containers

Steve Baird asks why we are not doing other protected containers. That's discussed in the !discussion of this AI.

#### *Queue interface*

Enqueue: "[If]{While} Is_Full is True, [then] Enqueue waits [on the entry] for storage to become available. {As soon as Is_Full is False, }copies New_Item onto the tail of the queue."

Dequeue: "[Is]{While} Is_Empty is True, [then] Dequeue waits [on the entry] for an item to become available. {As soon as Is_Empty is False, } removes the element at the head of the queue, and returns a copy of the element."

"Is_Empty returns True if the container contains no items."

"Is_Full returns True if the container is unable to contain additional items."

### *Unbounded_Queue*

Is_Full always returns False. We don't need to talk about Storage_Error. We do need to say that this does not leak storage (an Implementation Requirement like A.18.2(253/2) - No storage associated with an unbounded queue object shall be lost upon scope exit.)

### *Bounded_Queue*

Upon reflection, we think that we don't really want Is_Full and Is_Empty because they would always be wrong (assuming multiple tasks are using a queue) - they essentially would be a race condition waiting to happen.

Do we want a Count operation? You also might want to know the maximum number of elements. Max_Count? "Max_Enqueued" and "Num_Enqueued"? "Max_Used_Capacity"?

Steve Baird suggests "Max_Headroom", generating smiles all around.

"Num_in_Use", "Max_in_Use" are chosen.

### *Priority_Queues interface*

This interface doesn't seem to be needed. We want to put the order relation on the instance of the concrete queue, not on the interface. So drop the interface package, and use the basic interface for the priority queues.

### *Unbounded_Priority_Queue*

Should we use "<" to represent the order relationship here? No, it is not necessarily the typical ordering relationship (it might only work on part of the element, or be related to something like time). If not, what should the name of this formal function be? We could use Before. Other suggestions include Higher. But we already used Before for the ordering relationship for array sorting, it makes sense to be consistent. So we'll use Before.

### *Bounded_Priority_Queue*

Nothing additional to discuss now.

Approve intent of AI: 7-0-0.