Minutes of the 34th ARG Meeting

8-10 February 2008

St. Pete Beach, Florida, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Gary Dismukes, Bob Duff, Brad Moore, Ed Schonberg, Tucker Taft, Bill Thomas (except Saturday).

Observers: Greg Gicca (except Saturday), Stephen Michell (Friday morning only), Sergey Rybin (Friday only).

Meeting Summary

The meeting convened on 8 February 2008 at 09:25 hours and adjourned at 13:30 hours on 10 February 2008. The meeting was held in a conference room at the Sirata Beach Resort in St. Pete Beach, Florida, USA. The meeting covered the entire agenda.

AI Summary

The following AIs were approved:

AI05-0086-1/01 Statically compatible needs to take null exclusions in account (8-0-1)
AI05-0088-1/01 Only the value of "**" is equivalent to repeated "*"s (8-0-1)

The following AIs were approved with editorial changes:

AI05-0004-1/10 Presentation issues in the Standard. (9-0-0)
AI05-0013-1/10 No_Nested_Finalization is difficult to enforce (7-1-1)
AI05-0023-1/04 'Read on records with variant parts (7-0-2)
AI05-0029-1/02 Meaning of 12.5(8) (8-0-1)
AI05-0030-2/02 Requeue on synchronized interfaces (6-0-2)
AI05-0041-1/05 Can a derived type be a partial view? (5-0-3)
AI05-0047-1/04 Annoyances in the array packages (7-0-2)
AI05-0052-1/04 Coextension and distributed overhead (7-0-1)
AI05-0053-1/03 Aliased views of unaliased objects (8-0-0)
AI05-0060-1/04 The definition of remote access types is too limiting (7-0-1)
AI05-0063-1/02 Access discriminants on derived formal types (6-0-2)
AI05-0066-1/03 Temporary objects are required to live too long (8-0-1)
AI05-0076-1/01 Meaning of "function with a controlling result" (8-0-0)
AI05-0077-1/01 The scope of a declaration does not include any context clause (8-0-0)
AI05-0078-1/01 Alignment need not match for Unchecked_Conversion (7-0-1)
AI05-0079-1/01 Other_Format characters should be allowed wherever separators are allowed (7-0-1)
AI05-0080-1/01 "view of" is not needed when it is clear from context (7-0-1)
AI05-0082-1/01 Accessibility level of generic formal types (6-0-3)
AI05-0084-1/01 Pragma Remote_Types for Container library units (8-0-0)
AI05-0087-1/01 Formal non-limited derived types should not have limited actual types (8-0-1)

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

AI05-0051-1/05 Accessibility of dispatching function calls (7-0-1)
AI05-0067-1/03 More build-in-place issues (6-0-2)
AI05-0083-1/01 Representation values of formal parameters (6-0-3)
AI05-0090-1/01 Ambiguities with prefixed views of synchronized primitives (8-0-1)

The following AIs were discussed and assigned to an editor:

AI05-0059-1/02 Limited derived types and build-in-place
AI05-0075-1/02 More access discriminants checks needed
AI05-0081-1/01 4.8(5.1) should be checked in the private part of an instance

The following AI was tabled due to a lack of agreement as how to proceed:

AI05-0057-1/03 The class attribute of a constrained subtype

The following AIs were discussed and voted No Action:

AI05-0030-1/02 Requeue on synchronized interfaces (8-0-0)
AI05-0085-1/01 Changing Assertion_Policy in a smaller scope (7-0-2)
AI05-0089-1/00 Renames of components of Unchecked_Union types (7-0-2)

The following SI was approved:

SI99-0023-1/02 Usages of subtypes Name and Name_List in the ASIS specifications (9-0-0)

The following SIs were approved with editorial changes:

SI99-0022-1/06 Add Boolean queries to ease use of Trait_Kinds (6-0-3)
SI99-0028-1/05 What does "appropriate kinds" mean? (8-1-0)
SI99-0029-1/02 Inconsistent inconsistent list (9-0-0)
SI99-0031-1/01 Actual parameters for normalized associations (6-0-3)
SI99-0032-1/02 Add Universal_Access to Root_Type_Kinds (7-0-2)
SI99-0034-1/01 Subprogram_Default_Kinds needs a A_Null_Default (8-0-1)

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

SI99-0030-1/01 Add definition section to the standard (9-0-0)
SI99-0035-1/01 Undefined capabilities of the Data_Decomposition package (8-0-1)

The following SI was discussed and assigned to an editor for further work:

SI99-0024-1/08 Provide a semantic model in addition to existing syntactic model

The following SI was discussed and voted No Action:

SI99-0033-1/01 "=" for ASIS defined private types (7-0-2)

Detailed Minutes

Meeting Minutes

John notes that in AI05-0047-1 the title heading is incorrect. In the paragraph of AI05-0054-2 that starts "Ed suggests " fix "...[is] via..." Steve Baird notes that AI05-0057-1 is missing from his action item list. Add "(later assigned number AI05-0090-1)" to Gary Dismukes' item. Similarly, note that the last Steve Baird item is SI99-0035-1; the last Bob Duff item is AI05-0084-1; the last Randy Brukardt item is SI99-0033-1; the Greg Gicca definitions item is SI99-0030-1. AI05-0061-1's title is "discriminanted" - fix that everywhere.

The minutes were approved unanimously with these changes.

Date and Venue of the Next Meeting

Our next meeting is after Ada Europe in Venice, Italy. June 20-22, 2008. WG 9 is in the morning of June 20th, we'd start in the afternoon. We are reminded to make our reservations soon (if we already haven't).

The following meeting will be associated with the SIGAda conference in Portland, OR. Our dates will be October 31-November 2, 2008. WG 9 is in the afternoon on October 30th; we'll start on Friday morning.

Discussion of ASIS Standard draft

We look at the changes in general. It is decided to look at the SI99-0028-1 changes as part of that SI.

Looking at 15.9, a question is raised about the SI99-0004-1 changes: What does "after is" mean? Randy notes that we already have an SI (SI99-0032-1) for corrections to SI99-0004-1. We'll revisit this question as part of that SI.

It is difficult to find added subclauses, as we've tried to avoid changing the clause numbering for now. Tucker suggests somehow making them show up in the TOC. Randy suggests that perhaps we could make them sub-sub-clauses temporarily. It is thought that would be better than not having them in the TOC.

Thanks

Unanimously, we thank Greg Gicca (and his employer, AdaCore) for the superb local arrangements for the meeting that have made the meeting work well. We also add appreciation for the excellent weather that he provided. (The editor notes that it is 72F and sunny here, and -1F and snowing at his home at the moment of this motion.)

Old Action Items

Randy Brukardt did not describe a massive extension to Ada.Directories (AI05-0049-1). Greg Gicca did not do the renumbering (which is not planned for the immediate future). Bibb Latting did not do an update to AI05-0009-1. Reassigned that to Randy Brukardt. Bob Duff did not do AI05-0050-1, preferring to wait until the direction of AI05-0067-1 was determined. Tucker Taft did not do SI99-0007-1, SI99-0019-1, SI99-0021-1, AI05-0071-1, and AI05-0074-2. Tucker did AI05-0061-1 very late; it was folded into AI05-0041-1 that is on the agenda (but a new version was produced and distributed on Tuesday).

Erhard did not come up with a new draft for AI05-0054-1; he would like AI05-0054-2 taken off the agenda until he can be present for the discussion. The group agrees.

New Action Items

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

Steve Baird:

- SI99-0035-1
- Check if there are other rules [excluding the ones modified by AI05-0052-1 and AI05-0063-1] in the standard that could be reworded to use "immutably limited types" (See discussion of AI05-0063-1).

Randy Brukardt:

- AI05-0009-1
- AI05-0049-1
- AI05-0083-1
- Split question 2 of AI05-0079-1 into a separate AI, and determine the changes needed to implement the change (assigned AI05-0091-1 after the meeting).

Editorial changes only:

- AI05-0004-1
- AI05-0013-1
- AI05-0023-1
- AI05-0029-1
- AI05-0030-2

- AI05-0041-1
- AI05-0047-1
- AI05-0052-1
- AI05-0053-1
- AI05-0060-1
- AI05-0063-1
- AI05-0066-1
- AI05-0076-1
- AI05-0077-1
- AI05-0078-1
- AI05-0079-1
- AI05-0080-1
- AI05-0081-1
- AI05-0082-1
- AI05-0084-1
- AI05-0085-1
- AI05-0087-1
- AI05-0089-1
- SI99-0022-1
- SI99-0028-1
- SI99-0029-1
- SI99-0031-1
- SI99-0032-1
- SI99-0033-1
- SI99-0034-1

Gary Dismukes:

- AI05-0090-1

Bob Duff

- AI05-0050-1
- AI05-0059-1 (merge into AI05-0067-1)
- AI05-0067-1
- Create an ASIS example tool to find return-by-reference incompatibilities (preferably both using and avoiding the new semantic subsystem).

Greg Gicca:

- Add newly ARG approved SIs to the draft ASIS standard.
- SI99-0030-1
- Check the entire standard for predicates where both a "Raises ASIS_Inappropriate_Element for an unexpected element" and "Returns False for an unexpected element" are given (see discussion of SI99-0028-1).
- Create a new SI to highlight hard to understand wording for "expected" kinds for later ARG discussion (see discussion of SI99-0028-1).

- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format in the Paris minutes)

Tucker Taft:

- AI05-0051-1
- AI05-0071-1
- AI05-0074-2
- AI05-0075-1 (merge into AI05-0051-1)
- Check if 3.7(10/2) [as modified by AI05-0063-1] needs to be reworded to allow some partial views (see discussion of AI05-0063-1).
- SI99-0007-1 (possibly add to SI99-0024-1)
- SI99-0019-1
- SI99-0021-1 (possibly add to SI99-0024-1)
- SI99-0024-1

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs), Ada 2005 non-amendment AIs, and Ada 2005 amendment 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 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

SI99-0022-1/06 Add Boolean queries to ease use of Trait_Kinds

Change the question: "it needs to be possible" should be "it must be possible".

Has_Abstract should not include An_Interface_Type_Definition; it doesn't include **abstract**.

Approve SI with changes: 6-0-3.

SI99-0023-1/02 Usages of subtypes Name and Name_List in the ASIS specifications

Approve SI: 9-0-0.

SI99-0024-1/08 Provide a semantic model in addition to existing syntactic model

Randy thinks that suggested changes from the e-mail of October 15th need to be done. Quoting from that note: "Perhaps they should all be changed to a form like "Is_Pure_Unit," "Is_Remote_Call_Interface_Unit," "Is_Remote_Types_Unit," etc. That would address both the grammatical and consistency concerns." It is agreed that this change should be made.

Tucker notes that he called this a "subsystem" rather than an "interface" because ASIS is the Ada Semantic Interface Standard and one wonders what the rest of it is if this part is the "semantic interface". (But we have a hard time informally calling this anything other than the "semantic interface".)

Sergey wonders if we should use "program library"; ASIS had previously decided to use "environment" to follow the Ada standard. Tucker is worried about using "environment" by itself - it sounds like Al Gore should be involved. He suggests using "compilation environment". Sergey is also worried about use of "context", as that is an ASIS technical term. But it is used only in the "proposal" section; it is not in the formal wording.

Sergey is asked about his feeling on the interface so far (he has looked at it in detail). He is concerned that there isn't enough detail at this point, but he is worried that there is not enough interconnection between the old syntactic subsystem and this semantic subsystem.

What sort of expression is intended for Corresponding_View in Asis.Expressions? The Introduction implies that it is just "Name" (Name is not separate from Expression in ASIS). What is the result of a dynamic call (a dispatching call, or a dereference of an access type)? Tucker thinks that it should be a callable view (all expressions involving operators are really function calls). Clearly, we need the detail of this function defined.

Sergey is worried that you can get to the body, but then you have to go back to the structural subsystem. Tucker says that there isn't much semantic that you can do with the body per-se, items like the contents of it (a sequence-of-statements) belongs to the structural interface.

Sergey notes that old ASIS can only get the information about an object if it has a definition; this interface will work in all cases. That is good.

Ed notes that the ordinal of declaration is weird; generally, we only need to know the relative order of two declarations (and that's all compilers tend to have). Randy agrees. Both agree that it wouldn't be that hard to add an ASIS specific ordinal if needed. Tucker thinks that changing to a pure relationship between two declarations could be done.

Sergey says that he can add a few of these to ASIS himself, so that he doesn't need anything this complex. It is noted that he has access to the guts of an implementation that an ordinary user doesn't have. Sergey responds that they sometimes write such add-ons for customers. So he would like to see useful examples of how this will help create ASIS applications.

Tucker points out that processing names is very complex. Trying to find whether something is aliased would be fairly difficult; it's likely that a purely structural code will not correctly handle complex cases. But a compiler (of course) has already handled all of those cases.

Steve Baird wonders about the compilation unit (which is not covered here). That's fairly well covered by the structural subsystem, and we're not trying to duplicate everything.

Sergey notes that several capabilities are missing from existing ASIS: getting information from any sort of Name, no matter how complicated. And that is provided here. More powerful syntactical capabilities are also needed, like getting a list of subprograms from a given scope. That also would be covered by this set of packages.

This latter seems to be a good example to try: is it easier to write that iterator with the new interface or only with the original functions? Tucker notes that this interface tries to unify how to handle all kinds of declarations (for instance packages and subprograms work the same way).

Perhaps there are ways to simplify the presentation. Tucker wonders if multiple routines can be documented together (Is_Scalar, Is_Elementary, etc. are identified). Randy says that is OK (the Ada Standard does it in some cases, most notably Ada.Strings.Bounded and Ada.Strings.Unbounded).

View_Identifier takes a declaration; this returns the Defining_Identifier. This works on implicit declarations as well as explicit ones.

Tucker thinks that you would need to be able to find the declaration of a component from a component reference.

Object_View includes expressions (it really is Object_or_Value_View, but that name is too long). Of course, the Callable_View of an expression is probably the operator function call.

Sergey would like to put the Corresponding_xxx routines into child packages so it is clear when the semantic subsystem is used.

We clearly need examples. The possible call tree of dispatching calls is suggested. Another suggestion is to rewrite the task safety tool that Sergey has been using for an example. The task safety tool tries to find task interactions in the (static) call graph. This is 3300 lines of ASIS 99 code. Tucker will try to do that. Another choice is a tool to find possible incompatibilities between Ada 95 and Ada 2005. Bob is working on a return-by-reference tool for AdaCore anyway, so he should do that (and is given an action item to do so).

The best value is from unifying the treatment of names and possibly regions.

Tucker will document every function he will use.

Sergey would prefer a flatter untagged type structure, Tucker says that we previously debated that. (And the rest of the group groans.)

Keep it alive: 9-0-0

SI99-0028-1/05 What does "appropriate kinds" mean?

John would prefer that the subject would say "The meaning of "appropriate kinds"".

In the discussion, correct "this {is} quite tricky".

17.25 should not say "raise ASIS_Inappropriate_Element...", as it already has "Returns False for an unexpected element. This is also true of clauses 17.26, 16.38, and probably more. Greg will check the standard for this and fix them when encountered.

We turn to looking at the wording for 16.19, which Randy notes is confused no matter how he and Greg tried to write it.

Tucker suggests adding "either":

Type_Definition expects an element that has either:

one of the following Type_Kinds:
    An_Access_Type_Definition
    A_Formal_Access_Type_Definition

or that has one of the following Access_Definition_Kinds:
    An_Anonymous_Access_To_Procedure
    An_Anonymous_Access_To_Protected_Procedure
    An_Anonymous_Access_To_Function
    An_Anonymous_Access_To_Protected_Function

Also if the kind is An_Access_Type_Definition or A_Formal_Access_Type_Definition then it also has one of the following Access_Type_Kinds:
    An_Access_To_Procedure
    An_Access_To_Protected_Procedure
    An_Access_To_Function
    An_Access_To_Protected_Function

Similar wording is needed for 16.20.

Tucker thinks that the editors need to find cases like this and pull them out for future wording discussion. Greg will create such an SI.

Approve SI with changes: 8-1-0. Bob Duff thinks that the change is not the best solution.

SI99-0029-1/02 Inconsistent inconsistent list

ASIS '83 should be "ASIS 1.1.1"; ASIS 1995 should be ASIS 1999.

Approve SI with changes: 9-0-0.

SI99-0030-1/01 Add definition section to the standard

This SI proposes two options. We take a straw poll on which option is preferred.

Option 1: 10; Option 2: 1; Don't care: 1

The wording needs to refer to the Amendment 1 as well as the Ada 95 Standard.

"ASIS" and "Asis" need to be defined separately; the upper-and-lower case form is used to refer to the package - see the existing 1.3.

Approve intent of AI: 9-0-0.

SI99-0031-1/01 Actual parameters for normalized associations

Change the summary to read:

"The enclosing element of a normalized actual parameter is the unnormalized association when the unnormalized form includes an explicit association."

Remove the comment marks from the front of the wording.

Add a closing parenthesis just before the ":" for the end of the second paragraph of the question.

Approve SI with changes: 6-0-3.

SI99-0032-1/02 Add Universal_Access to Root_Type_Kinds

Change subject to "Corrections for SI99-0004-1". [After the meeting, it was noted that one of these corrections is to SI99-0006-1, so the title ought to be "Corrections for SI99-0004-1 and SI99-0006-1".]

Correct question 1: "...defines {a} new universal type...", "...so {a} new ..."

Add a new question 3: '15.9 says: "For a single_task_declaration or single_protected_declaration, returns the task_definition or protected_definition following the reserved word **is**; use Is_Task_Definition_Present to determine if the entire definition is omitted." This makes no sense in the case where there is no definition present - there is nothing to return. What should this say?'

Tucker explains that this returns the subtype definition associated with the task. So get rid of the syntactic stuff here. Sergey wonders what it returns in this case?

"For a single_task_declaration or single_protected_declaration, returns the task_definition or protected_definition; use Is_Task_Definition_Present to determine whether the reserved word **is** is present."

Sergey wants to know the details of the returned subtype definition if there is no **is**. Specifically, what is the line number, position, and span of the returned entity? He thinks this is too complicated. Tucker notes that the distinction is irrelevant unless you are reconstructing the source. Getting nil some of the time is not helpful, for instance, you cannot find out the containing declaration of nil.

Tucker gives a use case for this change: I want a tool that prints out the line number of the type declaration of all objects in the program. You have no way to do that if this returns nil.

Returning to the wording.

Progenitor_List (15.49) ought to say that an empty list is returned if there are no progenitors.

"For a single_task_declaration or single_protected_declaration, returns the task_definition or protected_definition. If no task_definition is given explicitly (the reserved word "is" is not written), an empty task_definition is returned for which Is_Task_Definition_Present returns False."

For Is_Task_Definition_Present, change the text to say: "Returns true if the task_definition is given explicitly."

The empty task_definition has Is_Part_of_Implicit is False (else we could not get source information). Span is empty, the image text is empty, Line is that of the semicolon. Add that to the wording.

Approve SI with changes: 7-0-2.

SI99-0033-1/01 "=" for ASIS defined private types

Correct the first paragraph of the discussion by adding the missing word: "declaring "=" {abstract}...".

Randy tries to explain the difference between the two equality-like operations: Is_Equal is same for different context; Is_Identical is same for the same context.

It's not obvious which should be used for "=". Bob suggests having "=" raise Program_Error, but then it still wouldn't compose (the original operation would still re-emerge). Requiring composition would fix that. But the fixes seem worse than the problem.

Tucker motions No Action.

No Action: 7-0-2.

SI99-0034-1/01 Subprogram_Default_Kinds needs a A_Null_Default

"accomodate" should be "accommodate".

Approve AI with changes: 8-0-1.

SI99-0035-1/01 Undefined capabilities of the Data_Decomposition package

Sergey says that a customer created a sort of data dictionary for their program using this package. He goes on to say that AdaCore does not have Portable_Transfer implemented, so no one could possibly be using that.

Steve thinks that IBM doesn't support Portable_Transfer, either.

So only the capabilities of finding out stuff about the types is interesting. Randy wonders if the semantic subsystem will cover this. That information might be found late in the back-end, so it is not clear that it should be part of the semantic subsystem. And there is little value in forcing users to change their programs.

The stream operations all use types Portable_Data and Portable_Value. We could make all of those operations and types obsolescent; no one could be using them.

What about handling class-wide here? This package is mainly about the definition of types that meet the "simple static model" (see 22.1). Getting components of class-wide types (as opposed to those of the associated specific type) would not be a "simple static model" (for one thing, the size is unknown), and that is all this guarantees to handle.

So, make simple fixes (like the discriminants fixes), and make the stream part obsolescent. Then we'll see if we want to do more.

Approve intent of SI: 8-0-1.

Detailed Review of Ada 2005 regular AIs

AI05-0004-1/10 Presentation issues in the Standard.

Fix various typos:

In the discussion for (1): "reasomable" should be "reasonable".

In the discussion for (4): "parenthisized" should "parenthesized".

In the wording for (8): "generic_assocation" should be "generic_association".

In the discussion for (16): "objvious" should be "obvious".

In the question for (5): "is called a{n} attribute...".

In the question for (14): "shouldn't this [should] be".

A long discussion erupts on (18). Eventually, we decide that it is indeed obvious: the elaboration of an access_definition does nothing, and in fact the deferred constant and its full definition have two different types. Perhaps add an AARM note to note the ramification: "These elaborations cannot have any effect, because the subtypes have to statically match, meaning that either they are subtype_marks or static constraints."

Move question (20) to AI05-0067-1: seems like it is related to that and it needs discussion.

Approve AI with changes: 9-0-0.

AI05-0013-1/10 No_Nested_Finalization is difficult to enforce

Steve Baird explains that restrictions never note when they are intended to be enforced dynamically.

Tucker suggests: "* A restriction may impose requirements on the runtime behavior of the program, as indicated by the specification of runtime behavior associated with a violation of the requirement." AARM Ramification: In this case, there is no post-compilation check required."

For Simple_Barriers, Tucker suggests: "The Boolean expression in each entry barrier is either a static expression or a name that statically denotes a component of the enclosing protected object."

No_Nested_Finalization, 2nd sentence: "If an access type does not have library-level accessibility, then there are no allocators of the type where the type determined by the subtype_mark of the subtype_indication or qualified_expression needs finalization."

Tucker and Bob would prefer a shorter version:

"If an access type does not have library-level accessibility, then there are no allocators of the type where the type determined by the subtype_mark needs finalization."

But this does not indicate clearly that the subtype_mark in question is the one that is in the subtype_indication or qualified_expression. It is rejected, and we decide to use the previous suggestion.

Steve Baird has questions about Immediate_Reclamation. What does this mean? We don't know, and can't answer. Let's not open a can of worms trying to define this.

Steve then asks about No_Dynamic_Attachment. It says that "no calls" are allowed. But what about 'Access? What about a call of derived inherited operation? We surely don't want to allow either of those. It is suggested that "...call to..." should be "...use of...". Add an AARM note mentioning that this covers "'Access and 'Address, and derived inherited operations."

Another of Steve's questions involves No_Task_Hierarchy. He wonders how this should be checked for functions that return types that might have a task part. Tucker says that this restriction can be checked at the call site. But the task will briefly depend on a nested master before it is activated, so the language rules appear to disallow such functions when No_Task_Hierarchy is in effect. It would seem OK to allow that.

Steve points out that this is very hard to implement. This restriction is part of Ravenscar; it's not clear whether this added complexity would be worthwhile there. So return statements with a task part are not allowed. The language already says that; nothing needs to be done.

Add an AARM Ramification: "This disallows any function returning an object with a task part or coextension, as such a task would temporarily depend on a nested master, which is disallowed by this restriction."

Gary would like to drop "another" from this wording: "No task depends on a task other than the environment task of the partition."

Steve is worried about anonymous access types. It seems mainly to be an interaction with AI05-0051-1, let's discuss it then.

Approve AI with changes: 7-1-1. Bob disagrees; he wants the call of a function with a task part to be illegal (for No_Task_Hierarchy). Bob does not believe that you can catch this at compile-time. Tucker says that you can reject this as soon as you can prove that it is violated (that is, that is there is a function with a task part - it has to have a return statement, and thus violate the restriction). Remember that restrictions always ignore privacy. Ed agrees with Tucker. Bob still doesn't buy it.

AI05-0023-1/04 'Read on records with variant parts

13.13.2(27) has parens inside of parens. Tucker would like to remove the outer parens, and replace the opening one with a comma. Five commas in one sentence has to be near a record.

In the new text after 13.13.2(56), change the first paragraph to:

"If T is a discriminated type and its discriminants have defaults then in two cases an execution of the default implementation of S'Read is not required to create an anonymous object of type T: If the discriminants values that are read in are equal to the corresponding discriminant values of Item, then no object of type T need be created and Item may be used instead. If they are not equal and Item is a constrained variable, then Constraint_Error may be raised at that point, before any further values are read from the stream and before the object of type T is created."

Change the start of the second paragraph: "{The}[A] default implementation of S'Input..."

Fix a small typo in the question: "discriminant records" should be "discriminated records".

Approve AI with changes: 7-0-2.

AI05-0029-1/02 Meaning of 12.5(8)

This is the final Pascal AI, we forgot to put it on the agenda for the Fairfax meeting.

In the Summary, change "may not" to "might not".

Approve AI with changes (and another commendation for the author): 8-0-1.

AI05-0030-1/02 Requeue on synchronized interfaces

The original alternative is now obsolete because of the approval of the second alternative (AI05-0030-2).

No action: 8-0-0.

AI05-0030-2/02 Requeue on synchronized interfaces

Replace the Summary with: "Requeue is permitted to a synchronized, task, or protected interface if it is known that the operation is an entry. A new pragma is provided to accomplish this."

In the second paragraph of the Discussion, change "syntactic" to "static".

We consider the proposed Legality rules from pragma Implemented. Replace them with the following:

The *procedure_*local_name of a pragma Implemented shall denote a primitive procedure of a synchronized interface.

AARM Note: The *procedure_*local_name can denote more than one such operation if there are several overloaded routines.

A pragma Implemented with implementation_kind By_Protected shall not be applied to a primitive procedure of a task interface.

A procedure for which the implementation_kind is specified as By_Entry shall be implemented by an entry. A procedure for which the implementation_kind is specified as By_Protected_Procedure shall be implemented by a protected procedure.

If a primitive procedure overrides an inherited procedure to which a pragma Implemented applies then any pragma Implemented applied to the overriding operation shall be have the same implementation_kind or the inherited implementation_kind shall be By_Any.

In the Note at the end of pragma Implemented, change "declared" to "implies".

"Change 9.5.4(2-3)" should be "Replace 9.5.4(2-3)". Similarly, "Change 9.5.4(5)" should be Replace 9.5.4(5).

The resolution rule (9.5.4(3)) is awfully complex: do we want it to be this complex?? No, generally we make the complex parts a legality rule, we want ambiguity to be an error unless it is quite clear what is intended. So most of this should be a legality rule.

The Name resolution rule should be: "The *procedure_or_entry_*name of a requeue_statement shall resolve to denote a procedure or an entry (the requeue target) that either has no parameters, or that has a profile that is type conformant (see 6.3.1) with the profile of the innermost enclosing entry_body or accept_statement."

The rest becomes a Legality Rule added to 9.5.4(5):

"If the requeue target has parameters, then its profile shall be subtype conformant with the profile of the innermost enclosing callable construct."

"If the target is a procedure, the name shall denote a rename of an entry, or shall denote a prefixed view of a primitive subprogram of a synchronized interface, where the first parameter of the unprefixed view of the primitive subprogram shall be a controlling parameter, and a pragma Implemented with implementation_kind By_Entry shall apply to the primitive subprogram."

In 9.5.4(12), replace "new entry" by "requeue target" (drop the existing change).

Tucker wonders whether 9.5.4(6) is affected by that. What is the entry_declaration determined by the second sentence? Randy wonders about nested tasks that implement a synchronized interface. Tucker will investigate if there any problems with this, and get back to the group.

We stop for the day. The next morning, Tucker reports that the requeue accessibility is fine. The entry_declaration in question is the one that belongs to the entry_body mentioned earlier in this sentence. This is the entry_body that the requeue is completing, not the one that it being queued on. Randy notes that he had thought about this, too, and came to the same conclusion.

So, modify 9.5.4(6) as follows "...entry_declaration {for the entry_body}."

Ed says that dispatching Requeue was not a lot of (additional) implementation effort in GNAT.

Approve AI with changes: 6-0-2.

During the discussion of AI05-0060-1, Tucker wonders if pragma Implemented should be allowed on a synchronized private. Yes, it should be allowed. So this AI should say "synchronized tagged type".

AI05-0041-1/05 Can a derived type be a partial view?

We go through the wording carefully.

Randy notes that in 3.7.1(7), we still need the generic boilerplate (that applies to a legality rule, not to the definition of "known to be constrained"). He also worries that this seems backwards (even though it is right), because "known to be constrained" in this context really means "known to be indefinite".

Drop extra "is" from 3.10.2(27.2/2) (was "is is").

Before 3.10.2(29), add "In addition to the places where Legality Rules normally apply (see 12.3), the above requirements apply also in the private part of an instance of a generic unit." then drop that from the other bullets, including the new one.

There is a typo in the question: "...contains a[n] subtest".

[After the meeting, it was noted that the wording change for 3.3(23.10/3) does not cover untagged generic formal private types, as used in the example of question (2). The wording was adjusted to talk about an "untagged generic formal private or derived type", same as the wording change in 3.10.2(27.2/2).]

Approve AI with changes: 5-0-3.

AI05-0047-1/04 Annoyances in the array packages

John explains these items. Most of us barely understand.

John notes that the two AARM notes about the citations from Wilkinson should be the same (the longer one).

Approve AI with changes: 7-0-2.

AI05-0051-1/05 Accessibility of dispatching function calls

Tucker says that we had assumed that the caller and callee have the same idea of the accessibility of a function. But that isn't true if there is dispatching to nested extensions.

Consider a dispatching function returning a class-wide object of type T. If it is called for a nested type T2, it will have verified that the result will live as long as type T2. But the type of the return object is actually of type T'Class - which lives longer.

Tucker explains that in these cases the caller has to have provided additional information about the source. He claims that this already occurs for build-in-place, so the implementation effort is not new. Steve counters that these objects aren't necessarily build-in-place. Tucker suggests the coextensions definitely have this model, limited or not - they have to have a passed-in storage pool. There certainly are cases where passing stuff in is required, so adding some more isn't a terrible hardship.

Tucker is not changing the static accessibility checks. Steve says that they will then miss things, Tucker notes that is OK, as there always will be a dynamic accessibility check.

We look at the example in the Discussion. The comment should note that X.F4 is called. There are two types called Extension, which is confusing. Call them Extension1 and Extension2.

It's hard to imagine what is going on here, without the calls being written.

Consider the following declaration inside of Call_Functions:

```
Z : Tagged_Type'Class := X.F2; -- alternatively written as: F2(X);
```

We want this to succeed. The dynamic check is passed in from the caller.

Now imagine the following type declared after Tagged_Type (at the outer-level):

```
type TT_Ptr is access Tagged_Type'Class;
```

And then put the following declaration inside of Call_Functions:

```
Q : TT_Ptr := new Tagged_Type'Class(X.F2);
```

We want this to fail (the allocator has [almost] library-level accessibility).

Steve notes that this is required for *all* calls; it has nothing to do with dispatching. It only depends on whether the return type needs an accessibility check (class-wide, discriminated with access discriminants, and access returns).

But in the Q case, the allocator will make the tag accessibility check, so there is no problem in this case. The problem occurs only for access discriminants. If the extension in Other_Locals had an access discriminant, then we'd have the problem.

That's the case for F3. Imagine another type declared after Tagged_Type:

```
type D_Ptr is access Discriminated;
```

then consider the following declarations in Call_Functions:

```
Z3 : Discriminated := X.F3; -- Should succeed, because this will not live as long
                            -- as the local discriminant.

Q3 : D_Ptr := new X.F3; -- Should fail, lives longer than the discriminant.
```

Tucker claims that the same thing can be accomplished with a direct call to a nested function.

Steve wonders why the allocator can't check. That's because access discriminants don't carry accessibility levels.

Tucker says that accessibility check could be implemented as a "known to work" flag or a particular static level. If you were to pass in a flag (he suggests a large value) that it is known to work, then you have a check that will always succeed. If you do that, you don't have to implement generalized accessibility checks (that is, depending solely on dynamic nesting). He goes on the explain that the "current" level is always represented as "known to work"; other levels are passed normally. He says that increased levels of nesting don't have any effect, because you can't get to them.

He says that you have to do that to properly handle nested extensions. This comes as a surprise to the other implementers in the group.

Gary asked about whether anonymous access-to-subprogram causes trouble. Tucker does not think so, since this depends only on the return type, and that is a known part of the profile.

The AI needs to describe the full scope of the problem (any type that *could* have an access discriminant). There doesn't seem to be a problem with tagged types, because there are checks on allocators and other places. So it doesn't seem that there is a problem there, and the AI needs to explain that and remove the changes related to that.

Steve also would like an implementation model. (For instance, the one that Tucker has suggested.) He thinks that identifying a common ancestor could be hard (especially for anonymous access-to-subprogram). Tucker doesn't think that's true.

Randy asks why a solution similar to the tagged one can't be used (that would mean having accessibility stored with access discriminants). Tucker says that such a check would have to be complex (he used the word "nightmare"). That would introduce incomparable accessibility checks much more generally than currently.

This problem is new because in Ada 95 non-limited types did not have access discriminants, so there was no possibility of this problem. Also, in Ada 95, limited returns worked completely differently.

These dynamic checks are much more accurate than the original one, which often is too conservative or (worse) liberal. It is likely to fail only if there is a problem.

Access results would need to pass a storage pool along with the level. Tucker thinks it would be similar to build-in-place. This is a new feature, so the runtime cost is less of an issue (there is no compatibility issue to worry about).

Tucker also relaxed the finalization order for anonymous access types, so that anywhere during the correct finalization is OK.

Tucker is asked why he changed 3.10.2(15). He is not sure if this change is needed. He thinks it may have had to do with an existing check. He'll need to figure that out, one way or the other.

The example needs to be fleshed out with real calls, showing which succeed and fail. (The previous discussion provides a starting point).

Bob says that the GNAT implementation would use some of the build-in-place mechanisms, so it wouldn't be too bad.

Revise the subject to match the problem scope.

Approve intent: 7-0-1.

AI05-0052-1/04 Coextension and distributed overhead

The first "might be limited" in the recommendation should just be "limited".

Fix a typo in the recommendation: "a coextension[s]".

Use the recommendation to replace the first paragraph of the summary, and replace the recommendation with "(See Summary.)".

The 4.8(5.3/2) rule needs to include discriminants defined as part of subtypes. (That is, in "`subtype S is new T(new Lim);`")

"If the designated type of the type of the allocator is limited, then the allocator shall not be used to define the value of a discriminant, unless the discriminated type is inherently limited (see 7.5)."

Add at the end of the list of bullets:

"Within a generic body, a descendant of a formal private type is not inherently limited."

This is assume-the-best in the generic spec, so we'll need the generic boilerplate on all legality rules that depend on inherently limited.

There is a typo in the derived type bullet: "of a{n} inherently limited type".

Bob would prefer to name this differently because he thinks that "inherently limited" is a term that implies that it breaks privacy. (This does not break privacy.) We start brainstorming names: "permanently limited", "immutably limited", "intrinsically limited", "instinctively limited", "natively limited", "indelibly limited", "perpetually limited".

Bob suggests "explicitly limited", but that is already used. "statically limited". "really limited". "truly limited", "unavoidably limited", "definitely limited", and "clearly limited". There were literally dozens more suggestions; Tucker emptied out his thesaurus.

"Immutably limited" has the most support. There was concern that this term implies that limitedness change over time, and that isn't appropriate. However, someone notes that the wording "becomes non-limited" is part of the standard. So surely a term involving time is OK. Thus, "immutably limited" is chosen.

Fix another bullet in the definition: "an explicitly limited record {type}"

[After the meeting, it was noted that this definition can't go after 7.5(6.1/2), since that would split the "Otherwise..." part of 7.5(7) from its associated bullets. It makes more sense for it to go after 7.5(8), since otherwise that paragraph would be hanging by itself (immutably limited types are a subset of limited types).]

The 4.8(5.3/2) rules should now be written as:

"If the designated type of the type of the allocator is limited, then the allocator shall not be used to define the value of an access discriminant, unless the discriminated type is immutably limited (see 7.5)."

Factor the generic boilerplate out of 4.8(5.3/2), and have it apply to all of the legality rules here. Use "These rules apply" for that purpose. Paragraphs 5.1 and 5.2 almost certainly need the boilerplate, and it is best that the all (or none) of the legality rules for a particular entity apply in the private part of a generic.

An unrelated question about the last sentence of 4.8(5.3/2) is raised. A formal access type couldn't have a static Storage_Size, so how could this matter?

We can't decide, and this isn't important, so we just leave it.

To be consistent, we need to drop "view of" here; it is implied. We'll discuss that as part of AI05-0080-1.

Approve AI with changes: 7-0-1.

Later, during the discussion of AI05-0063-1, we note that this definition is not quite right. Tucker thinks we should always include formal limited private types in "immutably limited", and then recheck in the instance. In addition, we need assume-the-worst for the body.

We reword the definition to:

A type is *immutably limited* if it is one of the following:

- An explicitly limited record type;
- A non-formal tagged limited private type;
- A formal limited private type;
- A task type, a protected type, or a synchronized interface;
- A type derived from an immutably limited type;
- A type with a part that is of an immutably limited type.

A type T is not immutably limited in the body of a generic unit or within a body declared within the declarative region of a generic unit, if the type T is a descendant of a formal private type declared within the formal part of the generic unit.

There is a conflict between parts and derived types in this wording. Eliminate the derived type bullet, and merge the generic body rule, changing the definition to:

A type is *immutably limited* if it is one of the following:

- A descendant of an explicitly limited record type;
- A descendant of a non-formal tagged limited private type;

- A descendant of a task type, a protected type, or a synchronized interface;
- A descendant of a formal limited private type, except within the body of a generic unit or a body declared within the declarative region of a generic unit, if the type is declared within the formal part of the generic unit;
- A type with a part that is of an immutably limited type.

  Add an AARM note: "A limited interface is not immutably limited; a type derived from it can be nonlimited."

This rewording was approved along with the AI05-0063-1 changes.

AI05-0053-1/03 Aliased views of unaliased objects

Humm, looks like another place for "immutably limited". Change the wording of 3.10(9/2) to:

"In the case of an immutably limited type, the current instance of the type is defined to be aliased, as is any return object of such a type that is declared via an extended_return_statement (see 6.5)."

Tucker would prefer bullets, he fears that this seems to say that the return statement depends on the real type, not the view. Instead, we try further rewordings:

"The current instance of an immutably limited type is defined to be aliased. A return object of an immutably limited type that is declared via an extended_return_statement (see 6.5) is also defined to be aliased."

or:

"The current instance of an immutably limited type is defined to be aliased, as is any return object of an immutably limited type that is declared via an extended_return_statement (see 6.5)."

and finally:

"The current instance of an immutably limited type is defined to be aliased, as is the return object of an extended_return_statement (see 6.5) that is of an immutably limited type."

Change the summary to "Remove the aliased reserved word from the syntax of extended_return_statement; make immutably limited return objects implicitly aliased."

Approve AI with changes: 8-0-0.

AI05-0057-1/02 The class attribute of a constrained subtype

There is a typo in the question: "satifies" in the last sentence should be "satisfies".

Steve is trying to replace the dynamic rules of 6.5(5.9/3) (from AI05-0032-1) by requiring "static compatibility".

Does that cause problems with existing rules that use "static compatibility"? Those are 3.7(15) and 12.5.1(8). 3.7(15) only applies to elementary types; and the other is only about unconstrained ancestors which could not be class-wide. So it appears to be OK.

There seems to be other problems (unfortunately, not recorded) with trying to cram this into static compatibility. Moreover, doing so isn't necessary - there is a run-time check which would prevent any problems.

Ed motions to vote this no action. Randy suggests a class of pathology. But there is concern that this is not a pathology (even though there is no evidence that anyone has ever used it). There doesn't seem to be a consensus either way.

We'll let Steve try to fix this up and we'll take it up again tomorrow.

AI05-0057-1/03 The class attribute of a constrained subtype

[Editor's note: this version was distributed on the third day of the meeting.]

Steve removed all of the "statically compatible" wording.

6.5(5.2/3): why "constrained discriminants" instead of "non-null constraint"? Randy notes that "non-null constraint" is also informal wording. It should be the same.

Steve got this wording from 7.3(13). Copying existing wording is OK, so we won't make a check.

If you do this static check, then you don't need the dynamic check 6.5(5.9/3), it should be dropped.

Drop the parenthetical remark in 6.5(5.2/3), it is redundant and not very helpful.

Randy wonders whether there is a problem with anonymous access to class-wide in deferred constants; that requires static matching and doesn't need rewording here.

We agree that this AI is now technically correct; we now turn to the political question of what to do with the AI.

Is this a pathology? Bob disagrees that this is a pathology. He thinks that blames the user for writing something that might be reasonable in this case.

Ed notes that he would be annoyed if ACATS tests appeared on this item. Randy notes that would be a likely result of approving this AI; we try to test things where it is known that compilers get it wrong (once the ARG has decided what "right" is).

We take a straw poll on the preferred action for this AI:

No Action: 2; Move forward: 2; Abstention: 3. This shows that we have nothing close to a consensus on this issue.

Randy suggests that we table this one. If we get new information (that is, users try to use this), we could have a different vote in the future. We'll have to decide something before the next language revision, but that is likely to be a long ways off. Everyone seems to accept this resolution.

AI05-0059-1/02 Limited derived types and build-in-place

This one should be folded into Bob's AI05-0067-1; it should use the same wording. Probably the subject of AI05-0067-1 should be changed to "Build-in-place issues" (because this AI was the original one for which AI05-0067-1 is "more").

AI05-0060-1/04 The definition of remote access types is too limiting

We start by discussing the wording change of E.2(14):

Bob wonders if it is already OK to ignore annex pragmas that aren't implemented (if so, we don't need this wording). Tucker thinks that is true, but then reads 1.1.3(17). It doesn't seem to allow ignoring of annex-defined pragmas (either compile-time rejection or an exception is allowed). So we do need this wording (even for implementations that implement none of Annex E).

Tucker would like to put this in the introduction to Annex A. We check if any other pragmas should be covered. We don't find any.

Add after A(4):

"An implementation may omit pragma Remote_Types (see E.2.2) in language-defined library units if the implementation does not conform to Annex E."

The new paragraph E.2.2(14) should be in terms of pragma Implemented. It is OK to implement entries or protected procedures as ordinary procedures (in which case they could be blocking); they just can't promise non-blocking.

"The corresponding specific type shall not have a primitive procedure to which a pragma Implemented applies unless the implementation_kind of the pragma is By_Any."

Note 6 is confusing. We briefly try to reword this to make it both useful and correct, but give up. Drop the note 6, we don't think users are going to care much.

"7 The value of a remote access-to-class-wide limited interface can designate an object of a nonlimited type derived from the interface."

Drop the note 8, we're not going to make recommendations on use.

"9 A remote access type may designate a class-wide synchronized, protected, or task interface type."

Remove question (4). Question (1) should have an answer (Yes.) Question (3) should be answered (No.).

Drop the last paragraph of the summary. The second last paragraph of the summary should say "can designate". Get rid of everything but the first sentence of the second paragraph.

The editor asks Brad to update the discussion to reflect the changes to the other parts of the AI before Feb 14th.

Approve AI with changes: 7-0-1.

AI05-0063-1/02 Access discriminants on derived formal types

This appears to be an incompatibility with Ada 95 as we're now assuming the worst everywhere. This was not an intended incompatibility.

Tucker thinks we should always include formal limited private types in the definition for "immutably limited", and then recheck in the instance. We also have to assume-the-worst in the generic body. See the discussion for AI05-0052-1 for the rewording for the definition of "immutably limited" that resulted.

Steve Baird has volunteered himself to investigate what other rules in the Standard could (and perhaps should) use "immutably limited".

Move the wording for 3.7(10/2) from AI05-0059-1/02 to this AI and change it to:

"A discriminant_specification for an access discriminant may have a default_expression only in the declaration for an immutably limited type. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit."

Approve AI with changes: 6-0-2.

Tucker is concerned that private types are not allowed by 3.7(10/2). But this is not new; the Ada 2005 wording does not allow that. But Ada 95 did allow some partial views (they're OK if they contain the reserved word **limited**; the full type will necessarily conform and thus also have to meet the rule). Humm. Tucker will take this as an action item to determine if there is a real error and create an AI to fix any error determined.

AI05-0066-1/02 Temporary objects are required to live too long

The change isn't marked in 7.6.1(13); much of the text is new:

"The master of an object is the master enclosing its creation whose accessibility level (see 3.10.2) is equal to that of the object{, except in the case of an anonymous object representing the result of an aggregate or function call. The master of such an anonymous object is the innermost master enclosing the evaluation of the aggregate or function call, which may be the aggregate or function call itself}."

For the case where this object has coextensions, Steve would prefer this to be a hand-off, as it is for return statements. By this he means that a coextension is moved from the temporary object to the result object.

7.6.1(9.1/2): "If the object has coextensions (see 3.10.2), each coextension is finalized after the object whose access discriminant designates it." This implies that the anonymous object's coextensions would be finalized; it needs to be fixed if we stick with this model.

The hand off model seems like a better approach. Tucker will update the AI for tomorrow and we'll consider it again.

AI05-0066-1/03 Temporary objects are required to live too long

The header is missing from the first wording change. It should be "Modify 3.10.2(14.4/2) as follows:"

We have a long, bizarre discussion about "considered". Tucker insists on having it in the wording.

The parenthesized remark in 3.10.2(14.4/2) should be marked as redundant in the AARM.

Approve AI with changes: 8-0-1.

AI05-0067-1/03 More build-in-place issues

Bob notes that the anonymous object "becomes" the final object. Another term probably would be better: "morph" is suggested.

Steve says that this doesn't cover coextension hand-off.

Tucker would like this to say that this does not involve copying. Any copying would be "as if". We could have a To Be Honest note to say that such copying is allowed.

The "immutably limited" wording doesn't quite work, because whether a part is immutably limited depends on the view.

Tucker suggests changing that bullet to:

- If the full type of any part of the object is immutably limited, the anonymous object is build in place.

Steve asks where is the rule that the anonymous object is never finalized. That needs to be there.

We don't want to combine the two bullets, because we do not want to require build in place for non-limited functions (they have to be prepared to make a real assignment in the case of an assignment statement).

Bob would like to make access to the tag of an object during the return statement raise an exception. The changing of the tag in the case of the first question is uncomfortable. Tucker says that the model is that the object ceases to exist and the new object appears in its place; it is not the same object. Bob complains that all of the pointers point to the old object, and still do after the morph. Most seem to think that is OK. But there should be an AARM note that this change is not atomic in any sense.

Randy notes that making such a check would require some sort of distributed overhead (an extended_return_statement can do pretty much anything). He also notes that the same sort of error can occur from dispatching in a user-defined Initialize, and we're not going to try to fix that. Tucker doesn't think that either of these cases represents an error. He notes that some purists think that all redispatch is an error, but in the absence of that, he doesn't see much difference between the place where it happens. Randy explains that the problem is that such a redispatch is guaranteed to be done with a partially initialized object.

No one seems to be changing their opinion. We take a straw poll: should premature access to the tag raise Program_Error? 0-5-2.

We turn to discussing cases that are hard to implement. Should any of them be illegal?

Bob has a long example in the AI's discussion.

Steve notes that they have problems with:

```
type T2 is new T1 with
   record
      B : Boolean;
   end record;
```

There is a distributed overhead to this, in that neither the caller nor the callee know exactly how much space to allocate (if you are forcing contiguous allocation). That's in part because they put all of the dynamic parts of a record together.

Tucker is proposing that the parent part for an extension aggregate that is build in place shall have a nominal subtype that is constrained. (That will allow qualifying the function.) This would be a requirement, not an Implementation Permission. With this rule, either the caller or callee will know the size.

Straw poll on the above: 5-0-2.

This needs to allow null extensions, because the language can construct them. So this rule has to say a "non-null extension aggregate".

[On Sunday, Steve announces that this rule would need an assume-the-worst rule in generic bodies, if the parent is a generic formal subprogram. That could be a problem.]

The AI needs to actually answer the questions (at least in the discussion).

Bob should search the AARM for other wording that would need to be updated; surely there is some in 7.6.

Where should this go? Bob thinks it should be in 7.5, but it is not about limited types (only). Randy suggests putting it in 7.6, Bob thinks that controlled types is an unimportant case. Tucker and others disagree. Randy then suggests making a separate subclause for it and the rest of the assignment stuff. Bob doesn't like that either.

Tucker suggests renaming 7.6 "Assignment and Finalization", then it would be exactly right.

Approve intent: 6-0-2.

AI05-0075-1/02 More access discriminants checks needed

Tucker thinks this has to be integrated with AI05-0051-1. He would like to split the 4.8 part out, but that looks hard. He takes the whole thing with the intent of folding it in.

AI05-0076-1/01 Meaning of "function with a controlling result"

There is a dash character in the question and elsewhere, replace it with a hyphen as it shows up as a funny character on Apples and probably other machines as well. Keep AIs in the 128-bit character set if possible.

Change the Summary to: 'The meaning of "function with a controlling result" is defined.'

"If the call is to a (primitive) function with result type T {(a *function with a controlling result*)}, then the call has a *controlling result* — the context of the call can control the dispatching. Similarly, if the call is to a function with {an} access result type designating T {(a *function with a controlling access result*)}, then the call has a *controlling access result*, and the context can similarly control dispatching."

Drop the AARM note, it doesn't seem that important.

ACATS Test: "...there is no rule change."

Approve AI with changes: 8-0-0.

AI05-0077-1/01 The scope of a declaration does not include any context clause

10.1.1(21/2) should be 10.1.2(21/2) in the Wording section.

Library item is a syntax construct, it should be written library_item in the wording and discussion.

10.1.2(21/2) should read:

- within a context_clause for a library_item that is within the scope of a nonlimited_with_clause that mentions the same library package; or

10.1.2(22/2) should read:

- within a context_clause for a library_item that is within the scope of a use_clause that names an entity declared in the declarative region of the library package.

Ugh, we need an underscore in "library item" in 10.1.2(12). There are also 2 occurrences in the AARM that ought be fixed. Add that to the AI.

Approve AI with changes: 8-0-0.

AI05-0078-1/01 Alignment need not match for Unchecked_Conversion

Steve wonders if this is correct if Target'Alignment is zero. We want the source to be anything in that case. "...or Target'Alignment is zero."

Approve AI with changes: 7-0-1.

AI05-0079-1/01 Other_Format characters should be allowed wherever separators are allowed

Spell this other_format, add "An" in front of it, both in the Subject and in the Summary.

In the Discussion, add a space in "requirethis".

Does the change actually work? Yes, because an identifier cannot start with an other_format character.

We discuss the second question. Randy is not recommending the change now as it could impact existing programs, and it would require more work by implementers. Such substantial changes are best made as part of a significant language change. Tucker would like to make the change, he doesn't think there is any significant code that would be impacted, and the longer we allow the characters, the more likely that there will be. And not messing with other_format characters in identifiers would be easier for vendors that haven't implemented the Amendment changes yet. Randy says that it would make changes in a number of places in the Standard, so it needs to be researched. He thinks that we had added special rules to reserved words and to operator symbols to deal with this possibility.

Split the AI into two with a separate AI for question 2. Randy will research what would need to be changed in order to implement that change.

Approve AI (question 1) with changes: 7-0-1.

AI05-0080-1/01 "view of" is not needed when it is clear from context

Question is missing a "not": "...would {not} make any sense..."

It is noted that Pascal's example of a legality rule that ignores views is wrong: all views of a type have the same size (by definition), so it is impossible to tell whether a view is being used. Tucker doubts that there are any such static rules.

The wording should mention "subtype" as well, as that is more common in the Standard.

Corrections to the AARM note: "....need to look into the private part..."

"compile time" (no hyphen is needed unless it is used as an adjective).

"...run-time rules usually ignore privacy, so "view of" ..."

Approve AI with changes: 7-0-1.

AI05-0081-1/01 4.8(5.1) should be checked in the private part of an instance

We did this yesterday in the context of AI05-0052-1. Nothing else is needed, so Randy will administratively combine this one with AI05-0052-1 and then delete this AI.

AI05-0082-1/01 Accessibility level of generic formal types

Reword the wording change as: "The accessibility level of a descendant of a generic formal type is never statically deeper than another level."

Better, add this to 3.10.2(19). "...specifying an access-to-object type {nor does it apply to a descendant of a generic formal type}; ..."

Tucker claims that 3.10.2(20) is actually redundant as a generic package does not include a new level. He thinks that it in fact should be in brackets (marking it as redundant) as it is just is a warm and fuzzy note. So make no change here.

Delete the discussion about 3.10.2(20); there is no need for this rule so there is no need for a rule change.

Bob would like to put brackets around 3.10.2(20) in the AARM, as it appears to be redundant.

Add an AARM Proof: A generic package does not introduce a new master, so it has the static level of its declaration; the rest follows from the other "statically deeper" rules.

Approve AI with changes: 6-0-3.

AI05-0083-1/01 Representation values of formal parameters

Randy tries to explain the problem: it appears that the alignment of a formal parameter depends on the actual. That's because representation aspects don't depend on the view, while a formal parameter passed by reference is clearly a view of the actual.

Bob claims that there is a language design principle that alignment is always known at compile-time. He finds it in 13.3(21.d-e). So that implies that the Size and Alignment of an object depend on the view of the object.

Tucker notes that 13.1(11) only covers types; there is no requirement that the Size and Alignment of an object be the same for all views.

There does seem to be value to making it clear that there is a potential difference in the 'Size and 'Alignment for formals and actuals when passed by reference. While these are views of the same object, the results of these attributes still can differ. That surely isn't an obvious conclusion, especially as it is exactly the opposite of the situation for types. Tucker thinks that it should be mentioned in an AARM note after 13.1(11).

So this AI should be written as a Ramification with an appropriate AARM note.

There is no interest in looking for other problems with alignment.

Approve intent of AI: 6-0-3.

AI05-0084-1/01 Pragma Remote_Types for Container library units

"another AI" should be "AI05-0060-1".

Third paragraph of the discussion is OBE: it should just say: "If the implementation does not support Annex E features, then pragma Remote_Types can be omitted by a permission added by AI05-0060-1."

The wording is rather vague, change it to include a list of packages.

Approve AI with changes: 8-0-0.

AI05-0085-1/01 Changing Assertion_Policy in a smaller scope

Tucker says that it would be work to change this, and thus we need a compelling argument to make a change.

Ed suggests that we should make this pragma more useful, and it doesn't cost much more.

Randy says that pragma Suppress is not very well-defined in relation to actual practice (for instance, a pragma in a package specification is ignored by most implementations, and this is OK as the permission given by Suppress can always be ignored). It could be dangerous to depend on that definition for other sorts of pragmas.

Tucker says that if you have to touch the source code to change the state, there isn't a lot of benefit over just commenting out the assertions. The big benefit is from the extra capabilities of their proposed pragma Check. They aren't going to ask that that pragma be standardized now, although it might be at some point in the future.

Subject should be "Allow Assertion_Policy to apply to a smaller scope".

Fix the last sentence of the Discussion: "...Suppress doesn't support that, either."

Tucker moves No Action on this proposal.

No action: 7-0-2.

AI05-0086-1/01 Statically compatible needs to take null exclusions into account

Approve AI: 8-0-1.

AI05-0087-1/01 Formal non-limited derived types should not have limited actual types

Correct the Summary: "...if it[']s specific type..."

Change "non-limited" to "nonlimited" everywhere in the AI.

Bob wants the question changed to "Is the instantiation legal? (No.)". He babbles on about "rejection" being an Ada 83 notion.

Revise the discussion as follows:

"There was no intent that the class-wide type associated with a limited specific type [to] be non-limited; that would open up ways to assign tasks and the like. So we add explicit wording clarifying that class-wide types are limited if their specific type is."

"We certainly don't want the example in the question [be] to be legal, so we must add wording to prevent [it]{that}."

In the wording, "associated" should be dropped.

Correct the question: "type Has_Task is {limited} new Ifc with"

Approve AI with changes: 8-0-1.

AI05-0088-1/01 Only the value of "**" is equivalent to repeated "*"s

Approve AI: 8-0-1.

AI05-0089-1/00 Renames of components of Unchecked_Union types

The model of the language would be changed by allowing this, and it would provide an incentive to use Unchecked_Unions outside of their intended purpose of interfacing.

Robert Dewar originally posed the question, and seemed to have convinced himself that the "problem" applies equally well to non-Unchecked_Union types.

"In it" should be "It is" in the Discussion.

No Action: 7-0-2.

AI05-0090-1/01 Ambiguities with prefixed views of synchronized primitives

Gary notes that he has written revised wording since this AI was sent out. A primitive subprogram that is a homograph of an entry should be illegal. That requires new rules to be added to 9.1 and 9.4.

Add after 9.1(9.9/2):

"A non-inherited primitive subprogram of the type declared by a task declaration shall not have a prefixed view that is a homograph of an entry declared in the visible part of the task_definition."

Should this apply for ones in the private part? It could matter inside the task.

Randy points out that 9.1(9.5/2) seems to be this rule; we surely don't need to add it again. Ed notes that 9.1(9.5/2) doesn't seem to require that the identifiers be the same - that seems pretty fundamental! So we need to fix 9.1(9.5/2): add "and has the same defining_identifier…"

Can this happen for operators (for protected types)? It seems like it could, if the PT declares a "+" with one argument and the primitive has two arguments. So 9.4(11.4/2) needs to be worded slightly differently. We'll let Gary try to work that out.

The other problem is an ambiguity between entries and inherited operations.

Add to 4.1.3(9.2/2) after "the designator..." sentence:

"The subprogram shall not be a primitive of type T that overrides an inherited subprogram implemented by an entry or protected subprogram visible at the point of the selected component."

Should this be more or less general?

Tucker and Randy think it should be more general; the entry call can't be written otherwise.

Bob thinks that the component preference exists for compatibility and he thinks that is a bad rule. Bob thinks that if you have two subprograms that do different things, then calling them should be illegal rather than having a preference. Tucker is convinced.

There should be an AARM note to explain that this rule is not intended to be general, it just hides the wrapper subprogram.

Gary asks where the AI05-0042-1 declarations are declared. The wording doesn't say. That also ought to be fixed.

He also wonders if it is clear what these wrappers do. Bob thinks so, they are "implemented by" which is described in 3.9.2(20.1/2). That seems sufficient.

Approve intent of AI: 8-0-1.