Minutes of the 39th ARG Meeting

6-8 November 2009

St. Petersburg, Florida, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Gary Dismukes, Bob Duff, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Bill Thomas (except late Saturday and Sunday).

Observers: Greg Gicca, Matt Heaney (Saturday and Sunday), Steve Michell (Friday only), Joyce Tokar (Friday morning).

Meeting Summary

The meeting convened on 6 November 2009 at 9:05 hours and adjourned at 14:45 hours on 8 November 2009. The meeting was held in a meeting room at the St. Petersburg Hilton Hotel. As expected, the meeting covered just about half of the entire agenda.

AI Summary

The following AIs were approved with editorial changes:

AI05-0136-1/05 Multiway tree container (9-0-3)
AI05-0145-2/02 Pre- and Postconditions (11-0-1)
AI05-0149-1/05 Access type conversion and membership (7-1-3)
AI05-0151-1/04 Allow incomplete types as parameter and result types (11-0-0)
AI05-0157-1/01 Calling Unchecked_Deallocation is illegal for zero-sized pools (10-0-1)
AI05-0160-1/01 Additional ways to invalidate cursors (9-0-2)
AI05-0164-1/01 Parameters of access-to-subprogram parameters and derivation (10-1-0)
AI05-0178-1/01 Incomplete views are limited (11-0-0)
AI05-0181-1/01 Soft hyphen is a non-graphic character (9-0-2)
AI05-0193-1/01 Alignment of allocators (10-0-1)

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

AI05-0031-1/00 Add a From parameter to Find_Token (11-0-0)
AI05-0049-1/01 Extend file name processing Ada.Directories (10-0-1)
AI05-0111-1/06 Specifying a pool on an allocator (6-0-5)
AI05-0113-1/02 Conflicting external tags and other tag issues (10-0-1)
AI05-0142-4/04 Explicitly aliased parameters and accessors for Ada.Containers (7-0-3)
AI05-0144-2/02 Detecting dangerous order dependencies (11-0-1)
AI05-0146-1/03 Type and Package Invariants (8-0-4)
AI05-0147-1/04 Conditional expressions (10-0-1)
AI05-0159-1/03 Queue containers (10-0-2)
AI05-0166-1/01 Yield for non-preemptive dispatching (11-0-1)
AI05-0168-1/01 Extended suspension objects (8-0-4)
AI05-0171-1/01 Ravenscar Profile for Multi-Processor Systems (10-0-2)
AI05-0174-1/01 Implement Task barriers in Ada (10-0-2)
AI05-0179-1/01 Labels at end of a sequence_of_statements (9-0-3)
AI05-0183-1/01 Aspect Specifications (11-0-1)
AI05-0188-1/01 Case expressions (9-0-2)
AI05-0194-1/01 The default value of S'Stream_Size (9-0-2)

The following AIs were discussed and assigned to an editor:

AI05-0117-1/00 Memory barriers and Volatile objects
AI05-0122-1/00 Private with and generic children
AI05-0135-1/05 "Integrated" nested packages
AI05-0139-1/02 User-defined iterators
AI05-0141-1/01 User-dereferencing in Storage Pools
AI05-0153-1/03 Subtype predicates
AI05-0153-2/04 Discontiguous scalar constraints and extended discriminants constraints
AI05-0158-1/03 Generalizing membership tests
AI05-0163-1/00 Pragmas in place of null
AI05-0167-1/01 Managing affinities for programs executing on multiprocessor platforms
AI05-0175-1/01 Cyclic fixed point types
AI05-0176-1/01 Quantified expressions
AI05-0189-1/01 Restriction No_Allocators_after_Elaboration

The following AIs were discussed and voted No Action:

AI05-0010-1/02 Suppressing 11.6 permissions (8-0-3)
AI05-0138-1/00 Improving accessibility (11-0-1)
AI05-0180-1/01 Exits from named blocks (10-0-1)

The following SIs were approved with editorial changes:

SI99-0053-1/03 Open issues in the ASIS syntactic subsystem (8-0-4)
SI99-0057-1/01 Various comments on the ASIS CD (6-0-6)

The following SI was discussed and parts were assigned to editors:

SI99-0054-1/02 Open issues in the ASIS semantic subsystem

Detailed Minutes

Previous Meeting Minutes

Randy notes the editorial fixes from Jean-Pierre and John have been applied, and a revised minutes draft posted Tuesday. John indicated that he thought we had voted AI05-0138-1 "no action", but neither Ed nor Randy had recorded that. Do we want to change this, or just vote it again?? We'll vote it again.

Approve minutes by acclamation.

Date and Venue of the Next Meeting

For the February meeting, Asilomar is too expensive and too hard to get to. We should meet in a more convenient place (especially for the Europeans), such as Boston or New York.

We decide on Boston, Feb 26-28, 2010.

The June meeting will be at Ada-Europe in Valencia,. June 18-20, 2010.

WG 9

WG 9 decided that the vernacular name of this new Amendment will be Ada 2012.

ASIS CD Plan

How are we going to complete the ASIS CD responses and finish that Standard? After discussion, we agreed to the following plan:

- All work (other than by the editors) will be finished by the end of February. That means that all SIs need to be finished at the next ARG meeting.

- A second chapter by chapter review will be assigned at the next ARG meeting. Everyone will get different chapters than last time (so rereading the same text won't be necessary).

RM Review Plan

We also need to plan on reviewing the draft RM. We are supposed to start the National Body review next November, and we had better have reviewed the Standard well before that. So we'll start the ARG review of the Standard after the next meeting (but the ASIS review has priority). That will also be a chapter by chapter review.

Amendment Big Picture

We need to look at the big picture for the Amendment. There is some concern that we are getting too many changes without looking at the overall impact.

Bob isn't ready to discuss this, as he wasn't able to find time to draw up a chart. We'll either do it at the end of this meeting [we didn't – Editor] or next time.

Thanks

Thanks to SIGAda and for Greg for the accommodations and lunches. 11-0-0.

Old Action Items

Tucker didn't do an update to AI05-0155-1. Randy only half-finished work on AI05-0147-1. Pascal has not worked on his AI related to AI05-0006-1.

The Amendment subcommittee did not have a phone call before this meeting (Randy was so swamped with work in October that adding more didn't make sense to him), but will try to do so going forward.

New Action Items

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

Steve Baird:

- AI05-0135-1
- AI05-0194-1
- Determine syntax rules for case and conditional expressions (see discussions of AI05-0147-1 and AI05-0188-1). Help Randy with working for the static expression rules for AI05-0147-1 (and by corollary, Bob for AI05-0188-1).
- Participate in the amendment subcommittee.

John Barnes:

- AI05-0175-1

Randy Brukardt:

- AI05-0031-1
- AI05-0049-1
- AI05-0113-1
- AI05-0139-1 (with Tucker Taft)
- AI05-0141-1 (with Tucker Taft)
- AI05-0142-4 (using Tucker's new syntax from AI05-0139-1)
- AI05-0147-1 (with help from Steve Baird)
- AI05-0153-1 (getting examples from Bob Duff)
- AI05-0153-2
- AI05-0179-1
- Participate in the amendment subcommittee.

Editorial changes only:

- AI05-0136-1
- AI05-0145-1
- AI05-0149-1
- AI05-0157-1
- AI05-0160-1
- AI05-0164-1
- AI05-0178-1
- AI05-0181-1
- AI05-0193-1
- SI99-0053-1
- SI99-0057-1

Alan Burns:

- AI05-0117-1
- AI05-0166-1
- AI05-0167-1
- AI05-0168-1
- AI05-0171-1

Bob Duff

- Examples for AI05-0153-1
- AI05-0163-1
- AI05-0188-1
- AI05-0190-1 (see discussion of AI05-0111-1)
- Participate in the amendment subcommittee.

Greg Gicca:

- Handle various changes split from SI99-0053-1 [now assigned SI99-0058-1], see minutes.

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:

- AI05-0174-1
- Handle various changes split from SI99-0053-1 [now assigned SI99-0058-1], see minutes.

Erhard Ploedereder:

- Provide examples of aliasing problems from code surveys that could be/could not be detected by the proposed rules of AI05-0144-2 to Tucker for inclusion/consideration in that AI.

Jean-Pierre Rosen:

- Find a solution to the Wide_Wide_Character problem for ASIS (enuemrations with very many literals), see discussion of SI99-0057-1 [assigned SI99-0059-1] – with Bill Thomas.

Ed Schonberg:

- Study whether the proposed solution for AI05-0049-1 would make sense for VMS.
- AI05-0158-1
- AI05-0176-1
- Provide examples of incompatibility with the proposed rules of AI05-0144-2 to Tucker for inclusion/consideration in that AI.
- Handle various changes split from SI99-0053-1 [now assigned SI99-0058-1], see minutes.
- Participate in the amendment subcommittee.

Tucker Taft:

- AI05-0122-1
- AI05-0139-1 (with Randy Brukardt)
- AI05-0141-1 (with Randy Brukardt)
- AI05-0144-2 (get examples from Ed Schonberg and Erhard Ploedereder)
- AI05-0146-1
- AI05-0151-1
- AI05-0155-1
- AI05-0183-1
- AI05-0189-1
- Handle various changes split from SI99-0053-1 [now assigned SI99-0058-1], see minutes.
- Handle various updates to parts of SI99-0054-1 (see minutes), as well as comments moved from SI99-0057-1 (see those minutes).
- Participate in the amendment subcommittee.

Bill Thomas:

- Handle various changes split from SI99-0053-1 [now assigned SI99-0058-1], see minutes.
- Find a solution to the Wide_Wide_Character problem for ASIS (enuemrations with very many literals), see discussion of SI99-0057-1 [assigned SI99-0059-1] – with Jean-Pierre Rosen.

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

SI99-0053/03 Open issues in the ASIS syntactic subsystem

"Dependences" - don't change.

"Returns a nil_element". Note that this wording is used for other kinds of "Nil_xxx" as well (for instance, for Nil_Span). Remove the "a" in front of any kind of "Nil_xxx" over the entire document, without marking the change (that would be an insane amount of work).

Setting status. The status is useful for ASIS_Failed, but not for the other exceptions (the status would usually be Value_Error). So it doesn't pay to specify it. Correct the spelling of "Diagnosis" in the SI. Should make Diagnosis

obsolescent, suggesting to use Exception_Message instead, and add Implementation Advice to it that Exception_Message and Diagnosis ought to produce the same string.

Allowed variations:

Bob notes that "**use** A, B;" is not even semantically equivalent to "**use** A; **use** B;". Any implementation which made this transformation would be wrong; surely ASIS shouldn't allow it.

We decide to eliminate all of these "allowed variations".

ASIS doesn't have to be tied to a compiler: We agree with Randy's recommended minimum change. This requires care rewriting 2.1.1. Tucker will help Randy write this paragraph. Index "environment"! Make the other changes as suggested.

1.1.3.2: Change "conformance document" to "conformance documentation". Drop the last two lines of the first paragraph. Drop the definition of "shall be documented", as it is never used. In fact, get rid of the last three paragraphs.

1.1.3.4: Delete the last two items. No one cares if an application uses extensions. It surely doesn't "conform" to anything if it is using extensions; it isn't portable to any other implementation.

2.3: We need a new chart of units. Bill will draw a new chart of units. Greg would like to clean up all of the pictures. They will decide amongst themselves precisely what to do.

In 2.3 "Asis.Elements", move "(see section 13)" to the end of the first sentence. Make the list of kinds into a bulleted list.

2.5.4.1: We just want to fix this bullet. Steal the weasel wording from the 4th bullet: "All queries other than simple Boolean or enumeration value queries,"

9.8: Is_Equal should be defined in terms of Is_Equal on the individual elements. Add "where two compilation units are the same when Is_Equal on the corresponding Compilation_Units returns True."

9.9: This should requires Is_Identical contexts. Is_Equal contexts could be different, and that is not what is meant by Is_Identical anywhere else.

Section 9: Much discussion. We eventually agree that there ought to be a query, returning a list of configuration units. Add after 9.7:

```
function Configuration_Compilation_Units (The_Container : in Container)
            return Asis.Compilation_Unit_List;
```

The_Container specifies the Container to query.

Returns a list of all configuration compilation units contained in the Container. Individual units will appear only once in an order that is not defined. A Nil_Compilation_Unit_List is returned if there are no configuration units within the Container. All units in the result will have an Enclosing_Container value that Is_Identical to the Container. Raises ASIS_Inappropriate_Context if the Enclosing_Context(Container) is not open.

People are confused by the meaning of "nonexistent unit kinds" in 9.6. Of course ASIS doesn't return random kinds of things that don't exist, why say this? It is referring to the explicit kinds containing the word "nonexistent", so it better say that. In all of 9.5, 9.6, and 9.7, change "nonexistent unit kinds" to "A_Nonexistent_Declaration" and "A_Nonexistent_Body".

Why are there two routines, one that only returns bodies, and one that returns everything? Actually, the second routine only returns unit declarations; the text ought to say that. In particular, it does not return pragmas. Fix 9.7 to say that it only returns unit declarations, not bodies or subunits or pragmas.

[Editor's note: The above description is wrong; Library_Unit_Declarations is intended to return declarations and renames; Compilation_Unit_Bodies returns bodies and subunits; and Compilation_Units returns everything except

pragmas. It's pretty clear that the latter really ought to return everything (including pragmas), but since that would be incompatible, there should be no changes to the semantics definition other than the "nonexistent" change noted above.]

Make similar changes to 10,8, 10.9, 10.10, and add a new routine similar to the one above there as well.

13.40: Tucker thinks that this would have value with pragma Import. But this would be very complicated. Jean-Pierre says that there is not a lot of need for a normalized form, because the pragma arguments are in a particular order. You just look at the argument, no need to look at the name.

Brad will attempt to write up what this means; he has to deal with parameters that don't have names or that don't have values (if not given). (Neither of those cases happen for calls, so there is no guidance given in the existing ASIS.)

13.42: (now 13.56) Just use the wording from 21.2, and get rid of this entire thing. Replace 2.2.3.4 Hash by: "returns a numeric hash value for..."

17.1: We don't want to change the decisions here, but we could make them more visible. Add a note (really implementation advice) "Note: We recommend that the type returned in the implementation-defined cases be a type determined by the context."

Move the new Usage Note to the first regular note (so it appears in the Standard).

17.3: Jean-Pierre says it would be nice to properly handle names. We don't want to be allowing some names but not all of them (there doesn't seem to be any reason to stop here). So we'll leave this unchanged. Later, we realize that defining names are not expressions, so this doesn't even accept them. So it doesn't matter (Defining_Name_Image does this).

17.6: After much discussion, we decide everything is fine.

17.7: in all of 17.6, 17.7, and 17.8, change the parameter subtype to Asis.Name, and change the parameter text to "Reference specifies a name to query".

For the second issue, as usual, the existing wording having nothing to do with this problem is bizarre. What does "ambiguous" mean here? Probably "the pragma applies to several entities". And this is "exactly" like Corrsponding_Name_Definition – except for all of the things that are different! Fix that, too.

We also need to actually solve the problem. Add "When Corresponding_Name_Definition returns Nil_Element, the list will have length 0.".

The result reads horribly. All of the text of this clause should be rewritten to make better sense. Ed will turn these fragments and the existing mishmash into beautiful standards prose.

17.22: Tucker will try to figure out how to normalize <> in formal packages, and possibly how formal package parameter lists are normalized.

17.29: Use the longer version of the text from 18.28 for the last bullet here.

22, 22.1, 22.19: We need to find someone who can explain how it is used. We need to ask Sergey what it is being used for (Lockheed), so that we can tailor the rules to the actual usage. Bill will ask Sergey about this, and work with Tucker to wordsmith the suggested improvements.

Move the 1.1.2 and C.2 comments to SI99-0049-1, since they apply to it.

D.4: Tucker will revise this to reflect the current set.

Split out the unfinished items into a separate SI. [Assigned SI99-0058-1 after the meeting.]

Approve SI with changes: 8-0-4.

SI99-0054-1/02 Open issues in the ASIS semantic subsystem

What to do about the order of packages? The current order of packages avoids forward references. It's not ideal, but at least the reading is linear. So don't change it.

Saturday, the subject is re-raised after discussion within the ad-hoc ASIS subcommittee (see below).

It is suggested that the packages should be ordered subtypes, objects, and then other stuff. Otherwise, we can leave the current order.

Erhard suggests that the best order is the one that facilitates random access to the document. Whether the packages are directly compilable is less important. So reorder into subtypes/objects/...

Why don't the names of packages Program_Units and Profiles not contain "View"? Program_Units and Profiles aren't views. Thus, the names don't need to contain "View". Similarly, View_Declaration is not a view (it can be thought of as "Declaration_of_View"). No change here.

The name of queries seem inconsistent. What to do?

We start by discussing what a "statement_view" is. Randy and Jean-Pierre are confused as to the purpose.

Tucker claims that a label is a view of a statement, and corresponding statement allows to get to the statement. But that's wrong: the label might be associated with a statement, but it is a separate (implicit) declaration in the declarative_part. There needs to be a label view.

For an exit statement with a name, you get a view of the loop with that name. This is a statement view.

A statement view represents a label or named statement.

Suggest Corresponding_Statement should be Target_Statement.

Tucker will rework this one.

At this point, we decide to have Jean-Pierre, Randy, and Tucker to take this whole SI offline for resolution.

On Friday evening, an ad-hoc ARG ASIS subcommittee met for 3 1/2 hours to work on this SI. The group consisted of Randy Brukardt, Greg Gicca (who left early), Jean-Pierre Rosen, Tucker Taft, and Bill Thomas. We started after consuming a nutritious dinner of pizza slices, soda, and left-over cookies.

Continuing the discussion on the names of queries. Jean-Pierre asks about Defining_Construct (23.2.7). Tucker explains that this is the construct that defines the region. Jean-Pierre suggests naming this Element_Defining_Region.

"Function Element_Defining_Region returns the Asis.Element for the construct that defines the declarative region R."

Turning to Element_Denoting_View (23.2.2). This returns the name or expression associated with the view. (Not other kind of things.) Tucker will try to draft better wording.

Should try to start Target_Statement with the word Element. Tuck will try this, too. Element_For_Statement is suggested.

23.2.1: Why is there one kind of generic formal parameter, but not all of them? There is a Generic_Formal_Object, Randy missed that. But no Generic_Formal_Package, Generic_Formal_Types. For the last, we don't want to duplicate all of the type routines. We need to be consistent.

So get rid of all of these formal kinds, and then add a query to ask if it is a formal subprogram or formal object.

So add Is_Formal_Subprogram to 23.9.3 (the kind would be A_Noninstance_Subprogram), and add Is_Formal_Object to 23.5.1 (the kind would be Standalone_Object).

Jean-Pierre notes that all of the Is_xxx might as well be class-wide. They are all memberships on calls to Kind. For all of the Is_xxx, make the parameters class-wide, and drop the "is abstract".

Tucker suggests reordering both the subtypes and the Is_xxx in the same order as the packages themselves.

Tucker would like to ask the full ARG about reordering the packages to make them consistent with this definition (the results are previously recorded).

Moving on to 23.2.2.

There probably ought to be an AASIS note here to say that the view depends on the original element. Pack.Var and Var denote a view with the same properties, but you would get elements representing different Names.

When a view is synthesized, we would want to have a rule as to what you get. Tucker would prefer that you get the expanded name (the full name of the entity). We need to have a global statement of that. For example, asking the Nominal_Subtype of an object (which can be an expression), we will need to know how to figure out what element you get.

Jean-Pierre notes that getting an expanded name is messy, because there isn't much that you can do with it directly, but there always is a simple name and you can use that. So it would be better for that to be returned when it is synthesized.

Jean-Pierre wonders if Element_Denoting_View should always returns a simple name (for expanded name). It makes life easier.

Tucker will try to come up with some guidance here.

What does Element_Denoting_View return when passed a view that has no corresponding syntactic element? It should return Nil_Element if there is no such element. (Many views are synthesized; we don’t want it creating elements, it should only return ones that already exist in the program.)

You never get a Defining_Name from Element_Denoting_View, you always (?) get a regular Name (really an Expression).

Is the use of "Denotes" in this name OK? "Denotes" is probably close enough for "2" and other literals. We don't see any need to change the name of this function.

23.2.3: What is the case of the returned string.: Be consistent with the syntactic interface. For instance, 10.20 has a similar rule. So: "If it is Other_Convention, the case of the result is implemention-defined, but the original source casing is preferred".

23.2.5: Declaration should be Element_For_View_Declaration to match our earlier naming.

Turning to the question of anonymous declarations. Tucker explains that his model was that View_Declarations are only for things that are named.

Randy complains that you have anonymous subtypes, you want to be able to find out what region they are declared in. Say you want to enforce a rule that there are no visible anonymous access types used (no parameters, no objects, etc.) You need to find out whether you are in the private part or the body, and the only way to do that is to get to the declarative region. You don't want to have to reproduce that, otherwise you get subtle bugs. (Besides, avoiding such things is the reason that we have created the semantic interface.)

Tucker thinks we need to make certain that there always is a declaration for a subtype; it just might not have a name. He will figure out the needed wording.

We discussed this issue with the full ARG on Saturday, but no new insights were found.

23.2.6: What is a child part? This surely isn't an Ada term. Many other of these other parts aren't Ada terms, either. Tucker will define what all of these mysterious parts mean.

Jean-Pierre asks about the name in a context clause. It seems to need to belong to some region. Tucker hadn't thought about that much (he was concentrating on usage names). He will think about that, too.

Why is Visible_Region_Parts returning a list? The parts mentioned and the child_parts are all visible, so there are multiple parts in a number of cases. So this is not uncommon. But there should be more wording/notes that makes this clearer.

Tucker notes that Ada has just a single visible part; he broke them up into multiple region parts, those are all considered one visible part by Ada. So wording is definitely needed.

23.2.11: Has_Declaration is clearly confusing. Tucker will try to improve the wording here. Not everything has a declaration, but we will treat subtypes as if they do.

23.2.12: Should Is_Aspect_Specified be view-specific? For representation ones, surely not. But for operational ones, that seems necessary (at least to be able to calculate availability).

Subtype Aspects and Representational Object Attributes are the same general idea, they should have a similar title. 23.4.6 should be Representational Subtype Aspects.

We really need Is_Stream_Attribute_Available; that is really hard to calculate, and it seems necessary for Annex E. Takes a Subtype_View and an aspect (a stream attribute aspect). In 23.4.6. Change this title again to "Representation and Operational Subtype Aspects".

23.4.4: Ultimate_Ancestor should return the entire set of ultimate ancestors. Add Root_Noninterface_Ancestor, which returns the ancestor that has no non-interface ancestors. Must raise exception Asis_Inappropriate_View if there isn't any (can only happen if Is_Interface is True).

The note is wrong. There was a parameter suggested. Tucker would prefer another function "Full_View_Root_Noninterface_Ancestor". (There are no hidden interfaces, so we don't need an extra Ultimate_Ancestors.) Fix the note.

23.5.1: The question is what view is returned? Tucker says that the properties of the view are well defined. Randy says sure, but what element would you get if you converted it back to the syntactic subsystem? That matters a lot; the enclosing element for instance would change.

Tucker suggests that it should return a subtype indication, probably from the object declaration. (Asis.Definition).

23.5.2: Tucker thinks that "is of" vs. "denotes" was not intended. Tucker says that his intent was that views always go through renames; he didn't intend them not to. Randy complains that that is confusing in some cases.

We need to think about renames carefully. We'll ask the entire ARG.

Is_Component_Selected_Component is a horrible name. Is_Selected_Object_Component? Is_Non_Array_Composite_Selected_Component? This last is what First_Bit, etc. require. (That surprises everyone, we thought it was record only, but 13.5.2(1) does not lie.)

On Saturday, the full ARG decides that the name of the function should be Has_Storage_Place_Attributes in place of Is_Component_Selected_Component.

23.5.3: We don't want static accessibility levels here any more than we want them in the Ada language itself. So we should have Is_Library_Level, and Has_Static_Accessibility_Level. No explicit levels. Is_Statically_Deeper_Than raising Asis_Inappropriate_View if either view does not have Has_Static_Accessibility_Level = True.

Tucker will redo this clause.

23.8.1: Questions on Contains_Task. "Part" does include the object itself. Does Contains_Task see through privacy? It seems that similar wording in the allocators and Unchecked_Deallocation are runtime; so that seems to be the

closest analog. And usually, you want to know if someone declared a task, even if they didn't know it. So this goes through privacy; we need to say that. More wording for Tuck.

Has_Nondiscriminant_Region_Parts is the same as **not** Is_Array, but using that would be pretty weird. We don't actually need this predicate, just return empty list if there are no parts (as in an array). So delete the predicate.

Tucker will reword this sentence. There is a one-to-one correspondence between region parts and lists.

23.12.4: Jean-Pierre agrees that the reason and meaning is understood. Probably the element would be nil. It's still confusing.

23.14.2 was previously covered, as was the first unknown clause question.

Bill wonders how you get from a procedure call statement to a statement view. You should be able to get to a callable view from the procedure call. Bill will send his question in e-mail.

Should there be an Expected_Type function? Randy complains that it would be expensive to implement. Jean-Pierre says he is getting the universal type for literals. Tucker says it is clear that you want the resolved type for views (specifically literals and named views).

So the nominal subtype of objects is the subtype of the view. For literals, it is the resolved type. It does not seem that an Expected_Type function is necessary. Tucker will make sure that there is some wording or notes to explain what the intended subtype of views of parts of expressions is (especially for literals).

On Saturday, the full ARG considered whether a given semantic characteristic looks through renames or just the properties of the current name. The question arose in 23.5.2.

Tucker had suggested that he intended all views to look through renames. If that's true, Is_Component ought to be be named Denotes_Component, which looks through renames. Or maybe Has_Enclosing_Object.

There are cases where we don't want to look through them, such as default parameters. So we need to say that by default, these operations ignore renames. But we better make it crystal clear when that is not true (Tucker says it follows from the Ada Standard, but we can't expect ASIS implementers and users to figure out whether that is the case.)

Keep alive: 9-0-3.

SI99-0057-1/01 Various comments on the ASIS CD

The header is just a copy of SI-53, it needs to have the correct numbers and dates.

8.3: "If The_Context is not open then" should be deleted from the third paragraph. Add "prior to opening it" to the first sentence of the second paragraph. That's clunky.

Tucker will take this off line to suggest a rewording. Much later, he suggests: Add to the end of the first sentence of the second paragraph " prior to opening." Replace the second sentence of the third paragraph with "A call on Associate will replace any prior name/parameter values with the given Name and Parameters."

8.7: The existing text "all unit versions are included in the comparison" definitely implies that the compilation status matters. We should add some wording to clarify that. Add "where two compilation units are the same when Is_Equal on the corresponding Compilation_Units returns True." This change is consistent with the one we made to 9.7 in SI99-0053-1; these two Is_Equals ought to work the same.

13.10, 13.12, 13.13: We're not trying to cover all types with this, we just meant the normal ones. A_Type_Declaration should be An_Ordinary_Type_Declaration. But in 13.13, we only want things that can have Private in them, so A_Type_Declaration should be A_Private_Extension_Declaration.

13.12: Drop Interface_Type_Definition here, it is covered by A_Type_Definition, and we don't want or need to get more detailed. (Recall that a correct implementation could ignore this list of kinds and simply determine if "limited" occurs textually.)

13.14, 13.16, 13.18: Make the change suggested.

13.17: Remove duplicate (and incorrect) A_Tagged_Record_Type_Definition.

13.25: Follow his suggestion.

13.47: The exception text is just extra junk, just remove last line. Use the 13.48 style wording.

13.50: "rambing" should be "rambling". This (the version with two parameters) is a really weird function, as you are actually guessing what your response is going to be. And the behavior is defined to be non-portable. Let's just make this function obsolescent, with implementation-defined semantics for the second parameter (implementations can keep doing whatever it is they currently do). No one wants this note to show in the standard.

15.9: Randy recommends rejecting this comment. No one disagrees.

15.11: Accept the comment (noting that the "Not_An_Element" is missing from the suggested correction; it is needed for Nil_Element). But note that the point is that the exception is raised if the Name does not designate a constant. We now need wording to say that. We can fold that wording into the exception raising statement, and we don't need the separate list of expected kinds.

"...Value_Error if the element Name has an Element_Kind is not A_Defining_Name or if element Name does not designate a constant declaration."

Greg would like a Comment in the .MSS source saying this is uniquely formatted (to avoid future surprises).

15.30: (was mistakenly described as "15.3"). Drop the last sentence, again a mistake.

15.47: The parameter surely should be "Declaration : in Asis.Declaration" and that should be fixed. But the original comment is wrong as 16.40 is defined for definitions; this routine is only intended for use on declarations. Ada syntax includes the progenitor list in some declarations (such as private extensions) which doesn't have definitions. Thus this routine is needed.

Someone comments that ASIS seems to define "definitions" for some of these, but we have no idea how they are defined (there is no indication in the ASIS standard what these things not found in the Ada grammar mean). Tucker says that Ancestor_Subtype_Indicator is just defined on "definitions". This is very similar to the progentitors case. So certainly the "definition" contains the private kinds. So we need to get rid of 15.47 and beef up the list of expected stuff in 16.40 to include the various sorts of private types. Probably add an AASIS note to explain the ASIS has "definitions" for private types even though Ada does not have any such thing, so private types can come here.

15.48: Fix this silly error. Move the entire thing to 13.19 1/2 (before Mode_Kind).

16.4, 16.8: Fork off this into a separate SI, and assign to Bill Thomas and Jean-Pierre Rosen.

16.18: Fix as suggested.

18.22, 18.23, 18.24: No change.

20, 20.25: Tucker suggests that we add "presuming text exists for the enclosing compilation (see the Ada Standard 10.1(2))." (Compilation means "source file" here, but we don't want to talk about files normatively.)

23 global: Add "Views." to the front of all of these. (We don't need "Asis." as that is the parent unit.)

23 global, parameter name comment: The first parameter is OK to be a single character (calls will typically be in prefix notation, and if not, positional notation is adequate for the first and often only parameter). But other parameters should have meaningful names. Tucker will do a pass through to see which parameters if any ought to be changed. Put this comment into SI99-0054-1.

23.2.1: Fix the order.

The Is_xxx routines are indeed poorly described. Tucker will look at improving the wording for the Is_xxx routines. (Also move this topic to SI99-0054-1.)

23.2.2: Make Randy's fix.

23.11.8: Try to make the wording the same.

Approve SI with changes: 6-0-6.

Detailed Review of Ada 2005 AIs

AI05-0010-1/02 Suppressing 11.6 permissions

Not enough of interest. Steve says the Rational compiler never used 11.6.

No action: 8-0-3.

AI05-0031-1/00 Add a From parameter to Find_Token

The basic idea seems sound.

Approve intent: 11-0-0.

AI05-0049-1/01 Extend file name processing Ada.Directories

Randy notes that he was unable to come up with a reasonable solution to the case-insensitive vs. case-sensitive file name issue. Tucker suggests having a constant of an enumeration type that says whether or not the system is case preserving, case insensitive, and case sensitive.

However, such a constant is a problem, because it is possible to have all of these kinds of file systems on-line at one. (Imagine a network including both Windows and Linux machines. Or even a Windows machine with a POSIX-compatible file system, which is a rarely used option.)

It could be a function, returning the result for a particular path. Not sure how practical that is, though, as it's not clear that there is anyway to find out (on Windows) whether a particular file system is case sensitive or not. Such a function should be added to Ada.Directories.

Tucker would like to name the new package Hierarchical_File_Names. That seems like an improvement.

The terminology for the file name pieces should be more standard: see what Java and/or POSIX use (if anything) to describe these parts of a file name.

Ed will ask someone at AdaCore to look at this vis-a-vis VMS.

Approve intent: 10-0-1.

AI05-0111-1/06 Specifying a pool on an allocator

Tucker explains what he has done.

The default pool is only for the current task; otherwise, it would be a dangerous global value. That means you have to re-establish the default for each task.

Subpools are automatically finalized when the reference count goes to 0. That means that some wording in 7.6 needs to change (allowing early finalization of elements in a subpool). [This last sentence is noted by the editor, who doesn't want to interrupt Tucker's fine explanation.]

You have to have a handle available in order to use an access into a storage pool. The referenceablity check enforces this.

Ed wonders if the reference count (which is an implementation detail) is really needed in the description.

Someone asks about interaction with tasking, particularly tasks allocated from a subpool. Tucker thinks that before you reclaim a subpool, you need to check that there are no tasks that can't be terminated. Tasks act as strong references.

We don't want this to be a "Notwithstanding", so there needs to be at least a hole in 7.6.

Bob would like a way to specify the default storage pool for all access types (which would be a user defined storage pool). Bob will take over AI05-0190-1 and try to add such a capability, since they are related. Randy worries about two competing subsystems trying to make different selections of a default storage pool. Bob says that a hard real-time system will know what everything does; Randy counters that not every system is hard real-time and this is going to be very attractive for solving all manner of problems.

Ed says that this is quite a bit of mechanism. Tucker says that putting it in an annex (probably the Systems Programming Annex) would help reduce complexity.

If we do that, what is the syntax change? We could just allow specifying a storage pool on an allocator. Tucker notes that the storage pool would have to have the same accessibility as the type. Randy worries about convertibility of values between pools.

The basic idea is that the syntax would make sense without referring to the annex.

Tucker will move the bulk of this to the Systems Programming Annex, and make the simple part in the core.

Randy wonders if we want this at all. Gary says he wonders if anyone will use it, it is so complex. Steve Baird says that Rational invented something like this. Java is pushing to add garbage collection to hard real-time systems; we can't do that in Ada. But this is implementable and would work for hard real-time systems.

Approve intent: 6-0-5.

AI05-0113-1/02 Conflicting external tags and other tag issues

Tucker dislikes this being erroneous. Randy points out that we need that because there is a problem with 'Input in a program with such a conflict. The attribute might read the data for a different type than was intended, resulting in chaos.

Tucker would prefer that this was simply detected at run-time; defining such an external tag should raise Program_Error. He thinks most run-times construct their external tag look-up tables dynamically, and such a check would be easy to do then. We ask how implementations do this if known. GNAT does this dynamically. Steve says that the Rational compiler does it at link time; it still should be possible to detect conflicts then.

Bob would also like to have an Implementation Permission to reject the program if this occurs. That usually would be a Post-compilation check.

That will replace all the wording in the AI.

Typo in the question "if there {are} no type{s}..." Summary has "beongs".

...tag of a type{, if one exists, } whose innermost master is {a}[the] master of the point of the function call.

Approve intent: 10-0-1.

AI05-0117-1/00 Memory barriers and Volatile objects

It seems that Atomic does require that the compiler do what is necessary to synchronize for the hardware and requires sequential. Volatile does not have a sequencing requirement.

There doesn't seem to be a proposal here. He would rather not have the compiler insert sequential markers, because it can't do it perfectly.

Alan is asked to investigate the issue of memory barriers and whether what we do currently is adequate.

Steve wonders if task A updates a garden variety object, then an atomic object, then task B (which was spinning on the atomic object) continues and reads the atomic object and then the garden variety object. Do we guarantee that the garden object reads the updated object? No, unless this is "signaling". Atomic is not signaling, protected objects are.

So there is a problem, sort of. You would need a magic protected object to force such signaling (it wouldn't necessarily need to do anything else).

9.10(9.c) [way back in Ada 95] attempts to explain the intended model.

The problem is that a "magic protected object" looks like a lock. However, if its barrier is always True, it wouldn't have to be implemented with a lock.

Keep it alive: 8-1-2.

AI05-0122-1/00 Private with and generic children

Tucker says that this is the sort of thing he loves. So we assign it to him, thinking he may feel differently when he is doing his homework.

AI05-0135-1/05 "Integrated" nested packages

Should we use use-visibility for this? Collision problems are essentially the same as **use** that way. That's preferable to using a new mechanism.

Randy still prefers an illegality if there are homographs, unless there is a renaming "linked" to the instantiation. But most of the support is for use visibility.

Sounds like we want to go with use visibility and angle names. Should children see this? Yes, but we only go out to the immediate scope of the integrated package. That is, if you have a nested package with an integrated package inside of it, the parent of the nested package cannot see the integrated package.

Ed wonders about an integrated package inside of a generic. He is convinced that there is nothing special that would need to happen for that case.

Can you have integrated renames? It seems like it should allowed; it is confusing but it seems inconsistent to not allow it.

Keep alive: 8-0-2.

AI05-0136-1/05 Multiway tree container

Should we renumber the indefinite containers? That would be necessary to keep all of the unbounded forms together, when we add this one. With the bounded forms, we were able to change just one clause number, this would change a whole bunch of numbers. Tucker thinks that it is important to keep things consistent, so all of the unbounded ones should be kept together. Nobody disagrees with that.

Delete_Subtree should raise Program_Error, not Constraint_Error if the cursor designates the root. Check all of the other operations for similar changes. Drop the note from Delete_Subtree.

Matt asks about Equal_Subtree when they are both roots. Since there are no elements in the root, you can't compare them. So we need to say that "Unless both cursors designate a root node, the elements are compared. If the result is False, the function returns False."

"element designated by" should say "node" whenever navigating, again because there is no element root node.

Tucker notes spelling errors in the document, he will send Randy a spell check. He then leans over and whispers them in the editor's ear while the discussion continues: "propogates" "deisgnating"

Matt wonders if Count should be the number of elements. No, it is the number of nodes (which is why it is always 1). Change the name of function Count to Node_Count; function Subtree_Count should be changed to Subtree_Node_Count. Child_Count is only child nodes, so the number of nodes and elements are the same. We don't need to change its name.

Is_Leaf can be called on the root and returns True if tree is empty.

The Delete operation should be called Delete_Leaf.

Find should be split into two routines, so we don't have to use a weird default that Tucker finds confusing. In the non-defaulting version, it is an error to pass No_Element.

Has_Element: AARM note should say "might not". It returns False for the root element. Tucker insists that it should be mentioned. Add "Redundant[This returns False if the cursor designates the root node or No_Element.]"

Tucker would like to say "if any" in the definition of cursor: " A cursor designates a particular node within a tree (and by extension the element contained in that node, if any).

Matt has some questions. He asks about Child_Depth, we want no change.

Insert_Child exception text is weird. Randy notes that it is the same as used in list containers; we want these to be as consistent as possible.

Delete_Child_Subtrees: This is pretty confusing. Tucker suggests just having Delete_Children which is simpler. Drop the Position and Count parameters from this routine.

Splice_Subtree has a check which is Order(Depth). It would be really messy if we didn't make the check and the error occurred: the tree could end up circularly linked or dropped on the floor. You could use a temporary object to avoid the check.

Parent, Matt thinks this ought to work like Next_Sibling, that is No_Element returns No_Element. This seems dubious, but it is consistent, so we agree to make that change.

Approve with changes: 9-0-3.

AI05-0138-1/00 Improving accessibility

All of the ideas that we are going to pursue have been split out into separate AIs. The rest should be dropped.

No Action: 11-0-1.

AI05-0139-1/02 User-defined iterators

There is discomfort with the fact that there is no way to iterate over elements.

Tucker would like the working subcommittee to try to explore this further.

On Sunday, Tucker shows his ideas on this topic. He would like to add syntactic sugar to the currently proposed semantics. See the separate file (now filed in the !appendix of AI05-0139-2).

Tucker is proposing:

```
for Element in Container loop

for Cursor in Iterator loop
```

Erhard asks if Element is writable in the element version. Yes.

Jean-Pierre would like a syntactic distinction between the Element version and the Iterator.

Randy concurs, because it is going to be hard to resolve. This would have an expected type of "any type", legality rules would apply. But one is an object and the other is a subtype.

Erhard says that it really is

```
for each Element in Container loop
```

But of course **each** isn't a reserved word at the moment.

Tucker suggests adding a colon to show that it is a variable:

```
for X:Element_Type in Container loop
```

Jean-Pierre suggests:

```
with Element in loop
```

No way, this ought to look like a for loop.

```
for Element of Container loop
```

A bit better.

For a Map, you would want iterators over keys and elements. That works, you can have multiple interfaces from different instances. That means that we need the element type in the loop syntax to be able to differentiate which one you want.

John notes that you need the array version, not just for containers.

Tucker and Randy will continue to develop this as an alternative to AI-0139-1.

Tucker says that which reference you get depends on the container (constant or variable).

Keep alive 11-0-0.

AI05-0141-1/01 User-dereferencing in Storage Pools

Add_To_Pool should be Import_To_Pool.

Erhard is confused by the use of address. Think of "Address" as handle here (the name isn't changed because we're using the existing interface).

Tucker worries about the absence of an "End_Dereference". Randy notes that alternative [D] included that, but Steve convinced him that it is was too expensive to implement.

Steve wonders if you could pass in the target storage pool to help choose whether to allow exporting; that seems like a reasonable idea.

Tucker notes that differentiating between Read and Read/Write dereferences would be important.

Randy and Tucker will take this back and try to integrate it with the subpools proposal.

Keep alive: 7-0-1.

AI05-0142-4/04 Explicitly aliased parameters and accessors for Ada.Containers

Bob would like to shorten these names because it is a lot of verbiage that means nothing. Bob would like to replace "Reference" with "Ref". "Const_Ref" sounds too much like C.

Bob says that these names are mostly noise, the shorter that they are the better.

Tucker suggests "RO_Ref" and "RW_Ref".

Erhard suggests "Ref" as a reserved word, make that a mode instead of "aliased". That isn't going to fly, several people say that they have many "Ref"s in their code.

Tucker would prefer the functions included the name of the element. "Element_Ref" is suggested. We discuss shorter names.

Steve suggests a really short name could be provided as a rename for the function, then the "official" name can be long.

Steve suggests VRef and CRef.

John: there is an example in the !discussion that includes a type "AT". That's reserved. Steve notes that the second Constant_Reference mistakenly returns a Reference_Type.

Tucker suggested "Data" for the discriminant.

Matt suggests Const_View or View.

We'll take this off-line again to try to come up with better names.

Approve intent: 7-0-3.

On Sunday, Tucker suggests that we add additional syntax sugar to this proposal, along the lines of what he is proposing for iterators. In that case, the names don't matter as much, as no one ought to write them directly other than to declare the implementation of the magic interface. Such magic promises to make access to a vector container look like an array index operation (even though there is much more mechanism going on under the covers).

AI05-0144-2/02 Detecting dangerous order dependencies

Ed found another example of conflicts with this rule (two **in out**s of non-by-reference parameters). There is some concern that for records that are known to be passed by reference in all real implementations, we could get false positives. Ed said that he had some, he will try to provide us with some examples.

Randy also is still concerned about dropping anonymous access from the first bullet. "i**n out**" and "**out**". Anonymous access is just as dangerous. Tucker claims that the matching rules really don't work for that. Randy counters that they could work in obvious cases (such as A and A.**all**). The problem with that is that we don't *know* whether the access type will actually be dereferenced, and there is no problem if it is not. So including anonymous access would not be the conservative rule that we seek. That should be added to the discussion in the AI.

Erhard returns to the first point: he is concerned about the compatibility of the "not known to be by-reference". Ed and Bob think the cases of incompatibility were records, so "elementary" would work and greatly reduce the incompatibility. (That's reverting to the originally proposed rule, it was changed too quickly last time.)

Erhard reports that he has an old paper that examines 8,000,000 lines of code. It includes examples of problems that this rule could detect. He is asked to provide examples of cases either way (where this rule would help, or where it would not).

Approve intent: 11-0-1.

AI05-0145-2/02 Pre- and Postconditions

The syntax usage of aspect_specification belongs in AI05-0183-1, not here.

The Static Semantics should say "the associated expression" because a postcondition does not have a precondition expression.

Should the Dynamic Semantics say "not specified" or "unspecified"? We'll let the editor decide that (off-line). [Editor's note: Either is OK. 1.1.3(18.a) says that "not specified" is used as a synonym for "unspecified". There are plenty of each in the normative wording – 59 pages with "not specified" and 43 with "unspecified". Accordingly, I didn't change the proposed wording.]

Why is the text about what happens when the assertion policy is Ignore marked as redundant? That's because the wording only defines what happens if it is set to Check. Randy is a bit dubious, because of the possibility of implementation-defined policies. Steve agrees. We decide to remove the redundant brackets from the last paragraph of wording.

The penultimate paragraph under Dynamic Semantics is missing the end redundant bracket.

Erhard suggests that the precondition is always inside of the protected action, otherwise you have possible race conditions. That's because the precondition could call a function of the object – and the result of that could change if it is outside of the protected action. Tucker says that you check the precondition before you check the barrier. Erhard notes that the precondition could become False while you are waiting on the barrier. Tucker reiterates that a precondition is really a requirement on the caller; but the caller cannot anticipate (and normally won't care about) the state of the barrier. A precondition that depends on something that could change in this case is dubious at best.

Steve wonders where these expressions freeze. Randy notes that you can evaluate function calls during initialize (in default expressions). Freezing of these expressions is defined by AI05-0183-1, the intent is that the general rules given in that AI are good enough for these expressions as well.

Gary wonders about the second paragraph of the Name Resolution Rules. This wording uses "converted to" some class-wide type. For instance, for an access parameter, the result is an anonymous access-to-T'Class. Tucker suggests that adding "corresponding" in the text might help. Steve wonders if this is really a conversion: he thinks it is means that it "is of type T'Class". He notes that we don't want junk checks happening here. Tucker says this works just like derived types. Those just use substitution, so conversion shouldn't be used here.

Brad asks for an example of Pre'Class and Post'Class use.

Approve AI with changes: 11-0-1 (pending completion of AI05-0183-1).

AI05-0146-1/03 Type and Package Invariants

Remove "Package" from the subject.

The syntax should be moved to AI05-0183-1 (most of it is already there anyway).

These aren't checked on entry; they are checked on exit. If the invariant is "bad", the precondition check could fail, but it doesn't tell you where the bug is and it usually is redundant.

There is a fragment of wording in the dynamic semantics. Gary wonders if an instance of a generic needs to be in this list of places. Tucker agrees.

"...or an instance of type T."

Bob says this doesn't work for scalar types (at least not without Default => expr – several people say that they really would like that, but no more discussion on that is made), because the default initialization is junk. The designer could use <> discriminants to avoid this problem. That's too bad, but obviously happens and no solution is apparent.

Steve says that" 'Access is evaluated" is not determinable statically. For instance, a subunit could take 'Access. Tucker doesn't want this to be non-portable; it should always be determinable.

Randy worries that the 'Access wording sounds dynamic. That's not what we want.

Bob and Jean-Pierre note that you can take 'Access of a subprogram declared in a private part in a child. How can you tell?

Bob suggests just dropping this, as there are other loopholes, and closing this one is just too hard. The group agrees: drop the 'Access rules. This is not hole free.

Add some text about "good" invariants. (There is a subset of invariants that is guaranteed to work properly. Invariants that dereference access types or otherwise use global variables are not guaranteed to hold, because other operations could change the invariant in places where it isn't checked.) Perhaps have a note in the RM and a bit more explanation in the AARM, and more still in the AI.

Randy notes that you can extend a private type with a visible extension, and visible record type with a private extension. A compiler ought to warn if there are visible components in an invariant. The visible extension can't have a new invariant.

Approve intent: 8-0-4.

AI05-0147-1/04 Conditional expressions

Randy did not finish several of the major changes needed to this AI. He did put brief notes about each issue into the text.

The only open issue is how to handle staticness. Randy tries to explain the problem, using Adam's example. The group agrees that Adam's example ought to be legal. Steve Baird will help Randy with that wording.

We then turn to the syntax. Randy notes that from e-mail we had decided on parenthesis being optional for a call or type conversion with a single parameter; an aspect specification; or a pragma with a single argument. Other cases would require parentheses around conditional expressions.

Tucker would like a more general rule. He suggests that a conditional expression should always be preceded and followed by parentheses. But that needs an exception for aspect specifications, and possibly for named notation. He expands his rule to "always preceded by a right arrow or surrounded by parentheses".

Erhard says that these occasionally optional parentheses will be really hard to define. He goes on to say that the user will have to remember complex rules. Tucker says that's why he wants a simple rule.

John thinks it is really ugly having a comma directly after a conditional expression. We try some examples (made more difficult by the lack of a whiteboard in the meeting room).

```
Proc (P1 => if A then B else C, P2 => if A then B else C);
    -- Omitting parens allowed
Proc ((if A then B else C), (if A then B else C)); -- Parens required
Proc (if A then B else C); -- Omitting parens allowed
Pre => if A then B else C,
Post => if A then B else C,
```

We decide to try a series of straw polls on each possible rule. The votes are recorded in the form Like-Hate-Can live with:

- 2-5-4: Aggregate like rules (special case for qualified expressions only)
- 2-4-4: Omit only when surrounded by parenthesis (only)
- 6-1-4: Omit only when surrounded by parenthesis or preceded by right arrow
- 2-5-4: Omit anytime except when ambiguity.

We select the third option (but see the discussion of AI05-0188-1 for more on this topic).

Approve intent of AI: 10-0-1.

AI05-0149-1/05 Access type conversion and membership

Tucker explains this proposal again.

Correctly spell "ambiguous" in the example for the rejected part 3.

!problem, we "dicuss".

Try not to start a line with "type" (it confuses the automatic formatter on the web site).

Approve AI with changes: 7-1-3.

Bob does not believe that the problem that derailed part 3 is unsolvable. Some of us think he's lost his mind. In any case, he says that he doesn't want to try (making the objection moot).

[Shortly after the meeting, Tucker asks that some wording changes be made as he believes too much was deleted removing part 3 from the wording. See mail of November 7, 2009 in the !appendix of the AI for details. These changes were made to version /06 of the AI – Editor.]

AI05-0151-1/04 Allow incomplete types as parameter and result types

This capability already is required for access-to-subprogram profiles, this AI just expands the usage a lot.

Tucker dropped some of the "helpful" text about not deferring to a body, he thinks its reads better without it. Randy would like to have an AARM note.

The entire last paragraph of the wording should be marked redundant (AI05-0178-1 already covers this normatively).

Brad has a typo in last paragraph of the proposal "any call or the body". No, that is right but hard to understand, so replace it with "all calls and the body".

Approve AI with changes: 11-0-0.

[Problems were discovered with this wording after the meeting, so it will have to be reworked. It has been reassigned to Tucker. – Editor.]

AI05-0153-1/03 Subtype predicates

It is probably necessary to evaluate the predicate on default initialization of an object.

Tucker suggests a permission to omit the check when the subtypes are the same. But the compiler cannot assume that the check is true if it doesn't make the check (similar to the rule for Pure function calls).

Lunch discussion suggests that they should be illegal/program_error for arrays.

No checks are performed when you modify components, etc.

Tucker suggests that we should allow compilers to check anywhere (even though it is only required at subtype conversion). Bob suggests it be a bounded error to violate the predicate of a subtype, and it can be checked anywhere.

There is no promise that 'First is in the subtype when the predicate is included.

Ed hates the idea that 'First cannot be guaranteed to be in the subtype. He thinks that it is much more complex. But there doesn't seem to be any sane way to guarantee that property.

Steve would like to see Bob's examples; perhaps he can keep this alive.

Keep alive: 6-2-3.

AI05-0153-2/04 Discontiguous scalar constraints and extended discriminants constraints

The group would prefer that list constraints never are allowed in arrays. So that we should drop the "discontiguous" idea, and that would require changing static matching to match.

Bob complains he doesn't like the name list_constraint. Randy said he thought about naming it set_constraint. Since this is unordered, it should be called set_constraint.

Make null ranges illegal, then we can drop all of the problems with determining the bounds.

One suggestion is that range_constraint of a list_constraint of should be illegal/program_error. That avoids problems. There would be value to allowing a range of the list, however, so we drop this idea.

Sets should use curly brackets. Tucker does not like that.

'Range is defined to be equivalent to 'First .. 'Last. But people think of it as equivalent to the subtype, and that is not true here. Better make it illegal/Program_Error.

Tucker hates the idea of the discriminant constraints. He doesn't like the model of "dead-end" subtypes.

Ed says he never saw a need for this proposal. Tucker says this is a new doodad for a limited need. Both Gary and Bob say they have wanted something like this for years. Erhard also notes that it happens periodically.

There is much argument that predicates are more general. But they are not as safe (because these are guaranteed by the language). You can write predicates for this fairly simply -- but as it is more general, you can't assume anything about their contents.

Tucker says that perhaps we should simply take both of these back and update them again.

Keep alive: 2-5-5.

So kill it. [But we forgot to vote to kill it, so it is sort of a zombie AI, assigned to me for a light update.]

AI05-0157-1/01 Calling Unchecked_Deallocation is illegal for zero-sized pools

"embarrassed" is misspelled in the last editor's note.

We don't actually need the generic contract issue rule, because it is impossible to put a call to Free in the spec. But we would need to explain why we don't need it, which takes almost as much wording as having the rule.

Drop the words "We recommend that" from the summary.

Typo in AARM note: "Since the execution of this call {is} erroneous..."

The last sentence of the legality rule seems unnecessary. We should drop it. We don't know the size for the formal anyway. Either the formal has a dynamic storage size, or we have a formal derived type and it might as well be illegal. But Randy objects that this is exactly the rule for allocators – we don't want the rules for allocators and deallocators to be different. So no change here. [After the meeting, the amendment subcommittee decided that this text is in fact unnecessary for both allocators and deallocators and decided to remove the sentence from both.]

Approve AI with changes: 10-0-1.

AI05-0158-1/03 Generalizing membership tests

The type of the value being checked is "any type". That simplifies resolution and makes this generally usable.

What equality operator is used here? We just made records compose, so it better be primitive equality here for records. If it is limited, there better be a primitive (usually user-defined) "=" operator.

It is important that scalar types work the same way as they currently do for ranges – we surely don't want any incompatibility from adding this.

The equality requirements means this ends up with the same rules as generic reemergence of "=".

There is a suggestion that the membership be equivalent to A = B or else A = C ... But that brings in visibility issues that we definitely don't want to deal with.

Probably the best plan is that it is illegal for limited types.

Ed and Erhard talk about allowing this syntax in a **for** loop. Tucker points out that it would have problems identifying the subtype of the loop object (there is no subtype name given or implied).

Tucker would like to see a proposal for extending this to **for** loops, with the objects presented in the specified order. There is no staticness requirement on these items.

Keep alive: 11-0-0.

AI05-0159-1/03 Queue containers

Steve thinks Num_in_Use and Max_in_Use are lousy names. Max_in_Use should be Max_Used. Bob would like High_Watermark. Matt wonders why we don't use Length. Tucker suggests Current_Use and Peak_Use. As often happens, Tucker's suggestions are the best, and we decide to use them.

Peek_Head, etc. are race conditions. Thus we don't want them. Tucker thinks that they are useless. If you need this, keep a local copy, don't expect the container to do it for you.

For the Enqueue wording, reverse the two sentences. "Waits on the entry queue for storage to become available."

"Some queue types have a bounded *capacity*."

"If the queue has a bounded capacity, and if the number of existing elements equals the capacity, then enqueue blocks until storage is available."

Same changes to Dequeue.

Erhard wonders why the unbounded queues have a blocking enqueue. (Meaning you can't call it in protected objects.) That's necessary because of the interface.

Jean-Pierre wonders if there is value to having the interface. Tucker argues that when using queuing usually you don't care how the queue is implemented. He gives an example of the Ada runtime system and the queuing policy. Most of the interfaces to the queue don't care. Alan agrees that the interface is valuable.

Steve wonders if child generics would make this easier. Semantically, they're essentially equivalent, but it surely won't make things easier to understand. Hardly anyone really understands how child generics work.

Ed suggests calling it Containers.Synchronized_Queues. That makes it clear that this isn't a simple data structure. Bounded_Synchronized_Queues, etc. But don't change the names of the priority queues.

Jean-Pierre asks for a straw vote on whether to have the interface at all. He is asked if the implementation would be visibly protected. No, he says limited private; but that doesn't allow requeuing on the entries.

In favor of the interface: 9, get rid of: 2, don't care/abstain: 3

John suggests that we include a small example in the RM.

Unbounded: Should just say that the capacity is unbounded, don't need any other wording. Also need to say that "type Queue needs finalization."

Bounded: Need wording to say that discriminant Capacity specifies the bounded capacity.

Unbounded priority queue: "Type Queue needs finalization." This unbounded queue also needs a statement about the capacity being unbounded.

Bounded priority queue: "<" is supposed to be Before.

Approve intent: 10-0-2.

AI05-0160-1/01 Additional ways to invalidate cursors

The order is wrong in all of the bounded error cases. It should be "... tamper with elements [Redundant: or cursors]." All of these rules also need to say "bounded vector" (or list or set or map), so that they are not assumed to apply to all of the container forms.

Typos in discussion: "assignment_statment", "distrubing".

Approve AI with changes: 9-0-2.

AI05-0163-1/00 Pragmas in place of null

Pragmas in place of "null;" gets some support (Tucker, Steve Baird, more). Pragmas in parameter lists have no interest.

Bob Duff will write this up.

Approve intent: 9-0-2.

AI05-0164-1/01 Parameters of access-to-subprogram parameters and derivation

Access should be in lower case in the subject.

Erhard: "other than those found in the profile of any access-to-subprogram type"

Tucker: "other than those found in the profile in an access_definition"

An AARM note should say that the above only is anonymous access-to-subprogram types present in the profile.

Gary would like commas around the inserted text.

We need the word designated in our new wording, because we need it in this other wording, too.

"other than those found in the designated profile in an access_definition"

Erhard notes that 6.1(28.2/2) also needs "designated".

Tucker claims he's using syntax meaning of profile (which is a parameter_profile or parameter_and_result_profile), so "designated" is not needed.

After more discussion, we decide that we ought to use similar wording to 6.1. But that requires extra wording. Tucker decides that "designated" is OK.

Approve AI with changes: 10-1-0.

Bob says he voted against because he would prefer to specify positively the places where substitution occurs -- but it's certainly not important enough to continue arguing about it.

AI05-0166-1/01 Yield for non-preemptive dispatching

This defines yield routines. Tucker argues that the second should be called "Yield_To_Same_Or_Higher". Randy says that is very long; if you don't care about the details (such as when only one priority is in use), we ought to have the shorter name. So add a rename:

```
procedure Yield renames Yield_To_Same_Or_Higher;
```

In the preemptive case, what happens? Yield is **delay** 0.0;, Yield_To_Higher is a no-op.

Steve Michell suggests that since Yield_To_Same_Or_Higher makes sense for all of the policies, it should be directly in Ada.Dispatching.

Tucker suggests putting Yield in Ada.Dispatching, and then in Ada.Dispatching.Non_Preemptive, add the rename:

```
procedure Yield_To_Same_Or_Higher renames Yield;
```

Approve intent: 11-0-1.

AI05-0167-1/01 Managing affinities for programs executing on multiprocessor platforms

Tucker notes that the exception names (other than CPU) should be in Upper and lower case.

Alan says that "Dispatching_Domain" is used in the standard somewhere, so we might want to call "Allocation_Domain" that. That seems better.

CPUs start in the system domain, once they are mapped elsewhere they are removed from the system domain.

"Allocate_Task" sounds like memory allocation. "Assign_Task" is better.

Bob wonders why we want to standardize these. What Windows has doesn't match this, so people would not be able to use this sort of thing on Windows. That isn't the main point of the real-time annex. There is no EDF on Windows, either (or most other systems, for that matter). It's still in the real-time annex.

Alan says that there aren't that many systems that could run multicore processes in any way.

Tucker says that with multicore, we want to be ahead of the curve. This is not a place we want Ada to be falling behind.

Each Dispatching_Domain would have its own scheduler.

Tucker looks at Windows documents; he thinks it is close to what is needed here. Randy notes that soft real-time systems could use this sort of capability, and they might want to run on Windows.

There is some discussion that it would be valuable that a reference implementation of this existed when the new Standard comes out in 2012. Alan agrees to try to get funding for a reference implementation.

We need to check what POSIX is doing so that this would be implementable on POSIX. It would be bad if that couldn't happen.

Ada.System is wrong, it should just be System.

No camel case: "Multiprocessors" (and make sure there are enough 's's).

Erhard wonders what is the default assignment for CPUs. Everything is in the system Dispatching_Domain, and the system handles assignments to tasks.

Domains are either global (task may run on any CPU in the domain) or partitioned (task may run only on one CPU). Erhard thinks 'partition' is bad name; the global scheduler applies in both cases.

Steve asks what Get_CPU does if the task is not running? Alan doesn't know, that better be figured out. Get_CPU returns the one it is assigned to, not the one you are running on (that is impossible to use usefully, because you could move between the time you ask and the time you could use the information).

Add the following to the first package:

```
Not_Assigned_CPU : constant CPU := 0;
```

Probably define a subtype that doesn't include the 0 for use in most of the interfaces.

Erhard wonders why Dispatching_Domains are not under Multiprocessors.
System.Multiprocessors.Dispatching_Domains

We discuss the name of the exception Non_Empty_System_Dispatching_Domain. Tucker would prefer "Empty_System_Dispatching_Domain", since it is the domain becoming empty that is an error. Steve suggests "Would_Become_Empty_System_Dispatching_Domain". Bob suggests a vaguer name.

Maybe we just need just one "Dispatching_Domain_Error". Use the Exception_Message to find out the exact reason.

CPU_Set is an array of Boolean. Randy asks what happens if there are 10000 CPUs – this is going to be a lot of memory. Should it be private? Tucker suggests making it unconstrained, so you can just give the bounds you care about. Steve wonders about wanting to set 1 and 10_000_000 only. That seems weird.

Jean-Pierre notes that System_Dispatching_Domain isn't really constant. After some discussion, we think this is very much like Standard_Output, this is just a handle. So it could be a private type (with reference semantics), so you can make copies.

The CPU should be inherited from the activating task, not from the "parent task" (or the master).

Tucker asks that interrupt Get_CPUs and the task Get_CPU_Set have the same name.

Is CPU appropriate; should this be "processor"? CPU is ancient but short. Someone suggests "core". We don't seem to have a conclusion. Randy notes later than D.14 uses "CPU_Time".

Erhard comments that if an interrupt comes back handling multiple CPUs, we don't know where the interrupt will show up.

Alan notes that we are only talking about protected entry interrupt handlers, not task entries. So if an interrupt ends up on the wrong CPU, it is just more expensive – it will still work.

Erhard notes that getting an array of bits is nasty in this case, because you have to search through all of the zero bits to see if one is on. So there should be a version of this routine that just returns a CPU.

Alan shows routines to process Dispatching_Domains. Tucker wonders why these are just numbers. What would the maximum number of domains be; probably the number of CPUs (one per domain).

Tucker suggests using ranges of Dispatching_Domains to specify pragmas.

Number_Of_CPUs isn't static; the program will determine this at startup (it won't change while running). That's surely true on Windows or POSIX (the compiler won't know how many CPUs there are). So perhaps this should be a function.

CPU_Set is not a good data structure for large numbers of CPUs; look at character ranges, which is a similar problem. We handled wide_wide_characters there, and that ought to work for CPU_Sets, too.

Tucker says that there is a limited number of "kinds" of scheduling domains. Then you could "instantiate" that when you create a dispatching domain.

Tucker says that it's important that the domain knows its scheduling when it is created.

Keep alive: 11-0-1.

AI05-0168-1/01 Extended suspension objects

The deadline in suspend and change deadline is is a relative deadline. The absolute deadline is calculated when the task is woken up; that's necessary as you can't know when the task will be woken up after it has been suspended..

Alan notes that you can do this currently with a protected object (the deadline can be set before the task a released). This is intended to be lighter-weight.

John asks why this is a separate package. We don't want to drag deadlines into existing code that uses suspension objects.

Probably there should be a note noteingthat protected objects could be used to generate more complex schemes (such as when you want to use the "set True" time).

Approve intent: 8-0-4

AI05-0171-1/01 Ravenscar Profile for Multi-Processor Systems

The second paragraph seems to not allow suspension objects. That's weird. The rest of that paragraph is "do it right". That's pointless. Just drop the whole thing.

Randy would like some !discussion to explain the reason for this model.

How are the CPUs assigned? It sounds like Ravenscar needs a pragma to assign, as it is a static assignment. AI05-0167-1 has "pragma CPU" for static assignment. That would be enough.

Tucker suggests that CPU_Set be pulled out of System.Multiprocessor. (Bob notes that it might have an access type). And then define pragma CPU. That would be a separate simple CPU AI (for use in Ravenscar). And then the larger one would define the rest.

The type CPU shouldn't be bounded by the function.

```
type CPU is range 1 .. <implementation-defined>;
```

Might as well move that here.

Approve intent: 10-0-2.

AI05-0174-1/01 Implement Task barriers in Ada

Tucker complains that calling these "Simple_Barriers" is weird. They're not that simple, at least compared to entry barriers. Randy is confused by comparing this to entry barriers. He would like to at least give an adjective ("synchronization"??) in front of the word barrier, else it is confusing.

This AI desperately needs an example.

Tucker doesn't think this should be a child package; there is no reason that you would have to use suspension object when you are using them. Perhaps the package should be called Synchronous_Barriers?

Steve Baird asks what happens when an ATC tries to cancel a call suspended on one of these barriers. That needs to be defined. The description doesn't seem very compatible with abort. Steve Baird suggests that it could work like a rendezvous.

"The_Barrier" is an unusual parameter name, come up with a better name. "Last_Released" isn't much better. Perhaps "Release_Last" would be better. The semantics are more "You_Are_It", actually, "I_am_It" is closer. We

chuckle over these names and the even shorter "Im_It". Perhaps use "Distiguished_Caller" for this parameter's name.

Tucker says that finalization has to raise Program_Error in any tasks waiting on one of these barriers. That's what protected objects do, and it should be possible to replace a protected object with one of these barriers (or vice versa) with minimum disruption.

Steve Baird wonders what happens if you call Wait_For_Release during its finalization. The obvious answer is to use the same bounded error rules that protected objects have (see 9.4(20.1/2)). We worry that it becomes harder to implement on top of POSIX or dedicated hardware with such a requirement (there most likely would have to be a wrapper in that case). POSIX seems to say it is unspecified what happens if a barrier is finalized when called. So maybe it should be implementation defined what happens for existing tasks when finalized (or tasks that come in after finalization started). [Editor's remark: It doesn't seem that hard to include a initialized/finalized bit with the Barrier object; the check only needs to be made when a new task is queued, presumably before adding it to the POSIX structure. So I don't see an implementation problem with the bounded error rule for finalization.]

Steve wonders why the discriminant Number_Waiting is not limited to a subtype bounded by Maximum_Parallel_Release, with a lower bound of 1. It should be. And Maximum_Parallel_Release needs to be defined in the package (not just the text).

Setting Number_Waiting to 1 is weird, but it might make sense during testing. So we'll keep that as a possibility.

Brad Moore will take this AI, with Steve Michell's help.

Approve intent of AI: 10-0-2.

AI05-0175-1/01 Cyclic fixed point types

John describes the proposal. John notes that the **range** or **mod** could be optional, as they have to match.

Steve wonders why you even give an upper bound, it is determined from the lower bound and the **mod**.

Jean-Pierre suggests `delta 0.01 mod range -180.0 .. +180.00;` Here Last - First is the mod.

Erhard asks about `subtype First_Quadrant is Bearing range 0.0 .. 90.00;` are such subtypes supported? Either answer is unpleasant.

John says there is no mod*mod operation. Randy says there no such operation for any fixed point type (you have to go through universal-fixed – are you going to try to ban that???). Also drops "**abs**". Tucker thinks it wouldn't be hard to make a proper definition. Negate and a reduction seems OK.

Steve and Gary worry about making the language bigger. John says this came from British Aerospace.

Tucker suggests requiring either 0.0 .. x.0 or an even -x .. +x, not allowing off-kilter ones. That would simplify the proposal.

Keep it alive: 7-4-0.

Bob says he voted against because of high implementation cost, and low utility. Randy agrees with Bob. Jean-Pierre thinks that for an application that would use this, they would be better off with a full library, and that is not for us to deal with. Someone failed to explain why they voted against.

John interjects that he don't believe in the high implementation cost (he thinks it is trivial); he thinks the big cost is integrating the rules into the Standard.

AI05-0176-1/01 Quantified expressions

This is an expression: otherwise you could write this as a loop and we wouldn't need a new construct.

We immediately leap into syntax. John wonders about "**for some**". Ed says that is "**not for all** (**not** condition)". Ugh. Tucker says "**for is**". No conclusions on syntax.

Without this, you would have to write a function to call.

It should short circuit (once you know the answer, you stop).

This supports the same things here as in a regular **for** loop. We don't want this to work differently, else there is a strange desire to use one or the other for no good reason. So if we add capabilities to regular **for** loops (via iterators), that ought to work here as well.

Keep alive: 9-0-1.

AI05-0178-1/01 Incomplete views are limited

Add (No.) to the !question

3.10.1(2.1/2) actually says that an incomplete view is limited. But that is not where all other limited views are defined, so it is confusing (obviously it confused the questioner and the author of this AI). We still want the changes in the AI.

Approve AI with changes: 11-0-0.

AI05-0179-1/01 Labels at end of a sequence_of_statements

Randy says that the model is that labels go between statements. Nobody agrees with that.

The wording says that a **goto** goes to a statement marked by a label. So the proposed syntax-only change is not enough. A trailing label needs to be on an implicit **null** statement.

Approve intent: 9-0-3.

AI05-0180-1/01 Exits from named blocks

Randy notes that there is the likelihood of confusion – users may expect an unnamed exit to exit a block (rather than a less-nested loop), even though that isn't the case. Removing a name from an exit could cause it to change what is exited.

Someone notes that this would have been a good idea if done originally (and consistently), but it is way too late for that now.

No Action: 10-0-1.

AI05-0181-1/01 Soft hyphen is a non-graphic character

Gary asks that the hyphen in "non-graphic" become a soft hyphen, in each place it occurs.

Approve AI with changes: 9-0-2.

AI05-0183-1/01 Aspect Specifications

Make the list of supported declarations an AARM note, not part of the normative text. It's pretty obvious from the syntax changes. Bob would also like to see the list of the ones not supported (and why). Subtype_Declaration seems to be missing here, add it.

These expressions cause freezing at the first freezing point of the declaration. That is, their freezing is delayed, much like default expressions.

The problem and summary need to talk about the same things. (The summary mentions pre/post conditions, we need to do the same in the problem.)

The dynamic semantics wording is confusing. Tucker explains that some aspects need values, and some need expressions; what happens is different for the two cases. OK, but the wording is still confusing; Tucker should improve it.

Bob asks if implementation-defined aspects are allowed. Yes, of course. What about unrecognized ones? They could be ignored, but that would be nasty (since the expression type isn't known, so resolution is difficult). This is similar to the difference between an implementation-defined attribute definition clause versus and implementation-defined pragma. We take an inconclusive straw poll: Ignore: 4 Reject: 3 Don't care: 5.

We definitely need to talk about how they affect the conformance of subprogram specifications. They also might participate in static matching – but maybe it is better if they don't. This probably should be decided on a case-by-case basis. It surely should be considered either way.

This probably should apply to all declarations if possible. Bob wants a good reason for any declarations that we don't allow this on. Tucker wonders if they should go before the **is** or at the very end for large items.

Bob wonders if the names of the aspects should be the same as the "aspect names" of the language. Ed thinks that the names should be same as the pragma and/or attributes.

Erhard says that half of the names read well. It is suggested that => True will make many of them read better (such as the "packing" aspect).

In any case, there needs to be a clear indication in the wording of what exactly the aspect_marks are: the current wording only says that an aspect_mark identifies an aspect, but never says how that is done.

Later, during the discussion of AI05-0145-2, Ed suggests that there be an annex that defines all of the aspects (similar to the existing ones for pragmas and attributes).

John notes that the ways that aspects are introduced seem to vary widely. That will make it hard to see what is an aspect. We need a specific style for introducing aspects (both old and new ones).

Approve intent: 11-0-1.

AI05-0188-1/01 Case expressions

You could write a function body to handle places where cases are needed. But that is the same argument as against conditional expressions, and that has already been decided.

The parenthesis rules should be the same as for conditional expressions. Someone points out that that would not work, as a case expression involves right arrows. So it would have to use "surrounded by parens (only)". That is uncomfortable, we'd probably want the same rules for case expressions and conditional expressions just to avoid insanity.

Erhard says that the arguments about coverage are pretty compelling. Someone suggests that you could use case statements to enumulate if statements.

Probably have the same rule for invalid values for these that we have for regular case statements.

Approve intent of AI: 9-0-2.

The syntax for case and conditionals need to be done together. Steve will try to determine rules. Bob will create the other rules needed (for staticness, invalid values, and so on).

AI05-0189-1/01 Restriction No_Allocators_after_Elaboration

There is a discussion about allocations in non-environment tasks. The proposal makes it wrong for a task to execute an allocator, even during the elaboration of the environment task. That seems to require a convoluted program structure: memory would have to be allocated by the environment task and somehow passed to the task that owns it.

Tucker thought that there was a race condition between the start of the main program and the various tasks which are running independently. But library elaboration does not finish until the **begin** of the main subprogram is reached. All of the library-level tasks have to be activated by then. So there does not have to be a race condition (as long as the allocations are restricted to the declarative part of any task body, there should not be a race condition). Thus, the restriction of allocators to the environment task should be removed.

Tucker says that we could say that such allocators aren't allowed in task bodies, as there surely would be a race condition for such allocators (as to whether the allocator or the **begin** of the main subprogram is reached first). Bob argues that doing it statically is impossible in any useful way, so there is no reason to bother with such a rule (many allocators would be in called subprograms, which usually cannot be statically checked).

Straw poll: Do we want rules preventing allocators in task bodies? No static limitations on task body: 3; Static rule after the **begin** of a task: 5; Abstain: 3.

Someone comments that you don't want to raise Storage_Error here -- because a failure of this restriction is not running out of storage. But the other restrictions with run-time checks use Storage_Error, so it should be consistent (and thus left as it is).

This restriction should be added to the list H.4(23.3/2).

Keep alive: 11-0-0.

AI05-0193-1/01 Alignment of allocators

Erhard does not like the attribute Max_Alignment_For_Allocation. Randy says that you could build your storage pool based on the maximum alignment. He further notes that he has pools that would be more general if they had this capability (they currently use named constants to provide the value).

It seems like a reasonable capability, and it certainly is similar to the existing Max_Size_Storage_Elements. So we do want the attribute.

Drop the paragraph in the !discussion about choosing the name of the attribute.

Approve AI with changes: 10-0-1.

[Editor's note: After the meeting, it was noted that some wording had disappeared in the rearrangement of 13.11.1. The last paragraph of that clause ought to read:

> For a type with access discriminants, if the implementation allocates space for a coextension in the same pool as that of the object having the access discriminant, then these attributes account for any calls on Allocate that could be performed to provide space for such coextensions.]

AI05-0194-1/01 The default value of S'Stream_Size

Randy still thinks that there is nothing that needs changing here. He claims that the combination of 13.13.2(1.6/2) and 13.13.2(9/2) define the meaning of the Stream_Size attribute. Any of the postulated scenarios violates one or the other of those rules; thus he believes nothing other than an AARM note is needed. Adding more text to 13.13.2(1.6/2) would just cause a maintenance hazard by repeating the definition of the attribute in two places.

Tucker and Jean-Pierre think that the wording could be improved. Bob says he agrees with Randy that the wording is fine.

Steve Baird will take this and try to clarify the wording. We all agree what this means, just not on whether the wording can be improved.

Approve intent of AI: 9-0-2.