Minutes of the 26th ARG Meeting

12-14 February 2005

Arcueil, France

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns (except Monday afternoon), Gary Dismukes, Pascal Leroy, Steve Michell, Erhard Ploedereder, Jean-Pierre Rosen, Tucker Taft.

Observers: None.

Meeting Summary

The meeting convened on 12 February 2005 at 09:15 hours and adjourned at 18:15 hours on 14 February 2005. The meeting was held in the conference room at AdaLog in Arcueil, France. The meeting covered the entire agenda.

AI Summary

The following AIs were approved:

AI-51-02/01 Size and Alignment clauses (7-0-3)
AI-404/03 Not null and all in access parameters and types (6-0-2)
AI-407/03 Terminology and semantics of prefixed views (10-0-0)

The following AIs were approved with editorial changes:

AI-158/05 Renamings of primitives of a class-wide generic actual type (7-0-2)
AI-291-02/02 By-reference types and the recommended level of support (4-0-6)
AI-392/01 Prohibit unsafe array type conversions (10-0-0)
AI-394/01 Redundant Restriction Identifiers and completing Ravenscar definition (9-0-1)
AI-396/01 The “no hidden interfaces” rule (9-0-1)
AI-397/01 Conformance rules and overriding indicators for entries and protected operations (10-0-0)
AI-398/01 A formal parameter should be usable only once (10-0-0)
AI-399/01 Single task and protected objects implementing interfaces (8-0-2)
AI-400/01 Wide_ and Wide_Wide_ image of identifiers (10-0-0)
AI-401/01 Terminology for interfaces (7-0-3)
AI-402/01 Access discriminants of non-limited types (7-0-3)
AI-405/02 Determining the ancestor interfaces in Ada.Tag (9-0-1)
AI-406/01 Aliased permitted with anonymous access types (9-0-0)
AI-408/01 Visibility for attribute_definition_clauses (8-0-2)
AI-409/01 Conformance and overload resolution related to anon access types (8-0-2)
AI-411/01 Equality for types derived from interfaces (10-0-0)
AI-412/02 Subtypes of incomplete types; renamings of limited views (5-0-5)
AI-414/01 pragma No_Return for overriding procedures (8-0-2)

The corrections to the following approved AIs were approved with changes (as a group - 8-0-0):

AI-85/09 Append_File, Reset, and positioning for Stream_IO
AI-216/15 Unchecked unions -- variant records with no run-time discriminant
AI-230/15 Generalized use of anonymous access types
AI-231/13 Access-to-constant parameters and null-excluding access subtypes
AI-235/08 Resolving 'Access
AI-248/11 Directory Operations
AI-249/10 Ravenscar profile for high-integrity systems
AI-251/18 Abstract Interfaces to provide multiple inheritance
AI-252/10 Object.Operation notation
AI-254/11 Anonymous access to subprogram types

AI-260-02/04 Abstract formal subprograms and dispatching constructors
AI-279/08 Tag read by T'Class'Input
AI-285/14 Support for 16-bit and 32-bit characters
AI-287/10 Limited aggregates allowed
AI-301/10 Operations on language-defined string types
AI-302-03/10 Container library
AI-305/08 New pragma and additional restriction identifiers for real-time systems
AI-318-02/08 Limited and anonymous access return types
AI-326/09 Incomplete types
AI-344/09 Allow nested type extensions
AI-357/08 Support for Deadlines and Earliest Deadline First Scheduling
AI-360/06 Types that need finalization
AI-362/06 Some predefined packages should be recategorized
AI-363/07 Eliminating access subtype problems
AI-366/08 More liberal rule for Pure units
AI-373/05 Undefined discriminants caused by loose order of init requirements

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

AI-395/04 Various clarifications regarding 16-bit and 32-bit characters (9-0-1)
AI-403/01 Preelaboration checks and formal objects (7-0-2)
AI-416/03 Access results, accessibility, return statements (7-0-3)

The following AIs were voted No Action:

AI-51-01/15 Size and Alignment clauses (8-0-1)
AI-291-01/06 By-reference types and the recommended level of support (8-0-1)
AI-353/03 New Restrictions identifier No_Synchronous_Control (9-0-1)
AI-389/03 Allow aggregates of partial views (10-0-0)
AI-393/01 Defaulted generic parameter evaluation and elaboration check (7-0-3)
AI-410/01 Limited with should be allowed on bodies (10-0-0)
AI-413/01 Partial view, task, and protected aggregates (10-0-0)

Detailed Minutes

Welcome

Jean-Pierre Rosen welcomes the ARG to Arcueil and AdaLog.

The group is amazed to see that John Barnes has brought a laptop to the meeting. He said that it was necessary to avoid killing his printer printing the AARM. The amazement subsides somewhat when it is learned that the laptop is actually Pascal Leroy's wife's.

Meeting Minutes

Randy made changes requested by Pascal in the draft version of the minutes for the 25th meeting of the ARG. John has two typos in the minutes. In AI-291, second paragraph, remove an extra "that". His second typo is in AI-330, and the text there is actually correct

Approve the minutes of the 25th meeting with changes: 10-0-0.

Next Meeting

Tucker will host the next meeting at SofCheck's offices in Burlington, Massachusetts, on April 15-17. Erhard would like April 16-18. After much scrambling in PDAs, we agree to the change in dates.

After that, we'll meet in York. We hope that there won't be any work left, but we seem to be getting behind, so it would be prudent to schedule dates for a meeting. Tucker again has a conflict after the meeting; moreover, it makes sense to meet before WG9 if there are problems. The WG9 meeting is Friday, June 24th. We decide to meet the weekend before the Ada Europe meeting — Saturday-Sunday-Monday 18-19-20, June 2005. Alan notes that it may be difficult to get hotel rooms for that weekend.

Thanks

The ARG thanks our hosts, AdaLog and Jean-Pierre Rosen for the fine room and view (especially of the aqueduct and the fascinating weather). Finally, the ARG thanks Randy Brukardt for taking the minutes, and especially for his getting through a meeting without his computer breaking.

Old Action Items

Most action items were completed. Randy didn't include section 13 in the AARM for review, so Pascal and Steve Michell didn't review it. Steve Baird didn't update AI-383. Steve Michell did not create an AI as assigned. Tucker didn't review the examples in the Standard. All other items were finished.

New Action Items

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

Steve Baird:

- AI-383
- Review AARM Clause A.18 and Annex B.

John Barnes

- Review AARM Annex F, Annex G, Annex J, Annex M, Annex N, and automatically generated annexes (K, L, P, index).

Randy Brukardt:

- Post new consolidated AARM including drafts of all sections by early March.
- Create a presentation AI to explain the reorganization of Annex M. (It will not include the actual content, just the headers and sections.)
- Create an AI to insure that stream attributes can produce abnormal values (see discussion of AI-405).

Editorial changes only:

- AI-158
- AI-230
- AI-231
- AI-249
- AI-251
- AI-254
- AI-260-02
- AI-291-02
- AI-301
- AI-302-03
- AI-363
- AI-392
- AI-394
- AI-396
- AI-397

- AI-398
- AI-399
- AI-400
- AI-401
- AI-402
- AI-404
- AI-405
- AI-406
- AI-408
- AI-409
- AI-411
- AI-412
- AI-414

Alan Burns:

- Review AARM Annex D and Annex H.

Gary Dismukes:

- Review AARM Annex A (other than A.18) and Annex C

Pascal Leroy:

- AI-395
- AI-403
- Review AARM Section 13, Annex F, and Annex G.

Steve Michell:

- Review AARM Section 13 and Annex E.
- AI on seconds and minutes from a Time_Span value (see discussion of AI-386).

Erhard Ploedereder:

- Review AARM Annex C and Annex D.
- Review wording changes of AI-401 once they are integrated into the AARM.

Jean-Pierre Rosen:

- Review AARM Clause A.18 and Annex H.

Tucker Taft:

- AI-416
- Review AARM Annex B and Annex E.
- Review the examples in the entire Standard and Amendment; suggest new examples as needed (especially for new features of Ada 2005).

Tullio Vardanega:

- Review AARM Annex A (other than A.18), Annex J, Annex M, Annex N, and automatically generated annexes (K, L, P, index).

Draft AARM Review

Randy tells the group that the AARM and RM distributed on Tuesday includes drafts of the entire AARM except section 13, annex B, annex C, annex D, and the list of packages at the start of annex A (all of the rest of annex A is complete). Review comments (other than those turned into AIs now on the agenda for this meeting) have been incorporated through 4.5.6.

We need to assign the review of the other half of AARM. The rules will be the same as the previous review.

Section 13: Pascal Leroy, Steve Michell

Annex A (other than A.18): Tullio Vardanega, Gary Dismukes

Clause A.18: Jean-Pierre Rosen, Steve Baird

Annex B: Tucker Taft, Steve Baird

Annex C: Erhard Ploedereder, Gary Dismukes

Annex D: Alan Burns, Erhard Ploedereder

Annex E: Tucker Taft, Steve Michell

Annex F and G: Pascal Leroy, John Barnes

Annex H: Alan Burns, Jean-Pierre Rosen

Annex J, Annex M, Annex N, and automatically generated annexes (K, L, P, index): Tullio Vardanega, John Barnes.

The review deadline is March 11 (close of business CST) for those annexes already done (A, E, F, G, H, J, M) and March 25 for the rest (13, B, C, D, N, auto generated).

Randy has a few questions that have come up during the work.

Q) There is an integration issue of looking for possible uses of **not null**, **access constant**, and iterators using anonymous access to subprogram types in language-defined packages. Who is going to volunteer?

A) Everyone should look for such uses when doing their review. Virtually everything that could be changed is still to be reviewed. Assign to everyone the tasks of looking for possible uses of **not null**, **access constant**, possible iterators.

Q) As previously discussed, Annex M was updated to include all of the documentation requirements in one place. It was retitled and broken into three clauses to do this. Is this what the ARG intended?

A) Pascal asks if we have resources to make these changes; Randy replies that it is already completed (other than editing). The annex is automatically created from special annotations, and those were easy to create. Randy is directed to create a presentation AI on these changes to present to the ARG. It shouldn't include the actual bullets, just an explanation of the changes and new organization.

Q) A.3.2 adds a bunch of stuff in front of a note. It looks much better to renumber the note paragraph than to have two dozen inserted paragraphs. Is this OK?

A) Renumbering of notes paragraphs in A.3.2 is fine. (This decision was overtaken by events; now the text will be deleted, see AI-395.)

Q) In A.1, the definition of Wide_Character and Ascii do not have paragraph numbers. This is weird, as these are fairly important declarations. Is it OK to add numbers to these items?

A) Yes, these declarations should have (inserted) paragraph numbers.

Detailed Review

The minutes for the detailed review of AIs are divided into corrections, existing amendment AIs and non-amendment AIs. The AIs 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.

Corrections to Approved AIs

All corrections were approved as a group (including the changes to them noted below). The vote was 8-0-0.

AI-85/09 Append_File, Reset, and positioning for Stream_IO

Making a wording change to A.8.2(16) seems better than saying we didn’t mean what we said.

AI-216/15 Unchecked unions — variant records with no run-time discriminant

The syntax was different in the proposal and wording sections. The description of which checks are suppressed was clarified.

AI-230/15 Generalized use of anonymous access types

Minor changes were made to 3.2.1(7), 3.2.1(8), 3.4.1(7), and 3.4.1(10). There were problems with accessibility in 3.10.2(12).

The correction to 3.2.1(8) should be changed to “...anonymous type that is defined by an access_definition or as part of declaring an object...” The “view of” wording is too tricky; let’s just say what we mean (and we want this to apply anywhere that an access_definition occurs). We still need to mention anonymous to cover arrays, tasks, and protected types.

AI-231/13 Access-to-constant parameters and null-excluding access subtypes

Erhard notes that we usually use italics to show the definition of a new technical term. In 3.2(8), “of” is in italics; we hardly are defining “of” as a technical term! These “of”s should be in the normal font. Change those in the presentation AI-415.

3.2(7) and 3.2(8) were modified to mention null exclusions. In 3.2(7) “subject” should be “subjected”.

The syntax changes proposed for 3.2.2 are junk (specifically the deletion of 3.2.2(5)); these were reversed.

3.10(12) was made more specific as to which reserved word **constant** is meant.

3.10(13) was written to please several reviewers. Tucker suggests changing “Other values” to “Non-null values” (in two places). (Apparently, the reviewers never are pleased.)

Membership operations (4.5.2(30)) needed to be updated to include null exclusions. The first inserted bullet should end with “;”.

8.6(20) deletes “numeric” so that *universal_access* is covered. AARM 8.6(20) should have “numeric” deleted, not inserted!

AI-235/08 Resolving 'Access

Fixed a typo in 3.10.2(2.3).

AI-248/11 Directory Operations

Added a passive iterator to be consistent with Containers and Environment_Variables. Added an AARM note giving the minimum contents for Ada.Directories.Information on Windows and Unix.

Should there be a file attributes modifications package like Ada.Directories.Information? That seems like it could be complex to do portably, so don't bother with that.

AI-249/10 Ravenscar profile for high-integrity systems

The syntax of pragma Profile was corrected.

D.13 should be split into two clauses, D.13 that defines pragma Profile, and D.13.1 that defines Ravenscar. We should do that so we can easily add new profiles in the future; otherwise we'd have to split it like we just did at D.2.

AI-251/18 Abstract Interfaces to provide multiple inheritance

12.5.1(15.1) wasn't clear, so the type of formal type was clarified. This paragraph should be moved after 12.5.1(5).

Added "itself" to 3.4.1(10). "Each" should be changed to "Every" to make John happy. (He claims that "each" can only be used for two alternatives, which meets with general confusion.)

3.9.4(1) was reworded. It should say "may have". "A tagged type, task type, or protected type may have…" The AI references are missing the "-01" in parts of this section.

Added 3.9.4(13) to say that the elaboration of an interface_definition has no effect.

AI-252/10 Object.Operation notation

A number of small changes were made to subclause 4.1.3, in paragraphs 9.1, 9.2, 13.1, 15.1.

AI-254/11 Anonymous access to subprogram types

Reworded 3.10.2(19) and 3.10.2(37). Removed obsolete junk from 3.10(17).

Changed 3.10.2(13.1) to avoid "infinite". Correct this change to say "statically deeper than {that of} any master".

AI-260-2/04 Abstract formal subprograms and dispatching constructors

There were minor changes in 3.9.2(1), 3.9.2(22), and 3.9.3(11).

Change the corrected 3.9.2(1) by getting rid of the comma before the new text, and change to the plural form: "[a] formal_abstract_subprogram_declaration{s}".

The erroneous execution paragraph (3.9(25.3) in draft 10 of the AARM) should say "whose tag has not been created", and should have a cross-reference to 13.14.

AI-279/08 Tag read by T'Class'Input

Added a cross-reference to 3.9(12). This is missing the closing parenthesis in the AARM.

AI-285/14 Support for 16-bit and 32-bit characters

2.1 was substantially changed by comments from the reviewers (John and Pascal). Updated 2.2(2) and 2.2(5) to reflect new terminology. Added digit to 2.4.1(5) — it better be defined somewhere. Changed 2.6(6.1) to a note. Deleted 3.5.2(4). No problems with these changes are noted.

AI-287/10 Limited aggregates allowed

4.3.3(23.1) was rewritten.

AI-301/10 Operations on language-defined string types

A.10.1(48.1) is missing from the AARM.The function Get_Line isn't well-specified if the string read won't fit in type String. Get_Line should raise Constraint_Error in this case. There is no normative wording needed, that will happen following the normal rules of the language. We definitely don't want to say what is left in the file. There should be at least an AARM note saying that this can happen, and Constraint_Error is raised. Several people want normative wording. The contents left in the file is unspecified. The current column number is unspecified.

"Constraint_Error is raised if the length of line exceeds Positive'Last; in the case, the line number and page number are unchanged, and the column number is unspecified but no less than it was before the call."

Perhaps this should be part of the second paragraph.

The From parameters for Index need a check. The check should be in the range of the string. If From not in Source'Range, then Index_Error is propagated.

In AARM A.4.7(48.a/2), Preelaborable_Initialization is misspelled. Check the others.

AI-302-03/10 Container library

Randy goes over the changes that he and Pascal made since the Atlanta meeting.

Should the Update_Element check be per-element? It seems reasonable to use multiple instances of Update_Element on different elements (say to copy a component from one element to another).Pascal says this is not sufficiently broken to change. No one else seems to have a strong opinion (we miss Matt ☺).

A.18.3(57) [lists] uses C, it should be L.

AI-305/08 New pragma and additional restriction identifiers for real-time systems

The discussion talked about pragmas when the items are actually restrictions identifiers.

AI-318-02/08 Limited and anonymous access return types

6.1(28) was updated to take into account anonymous access returns. We no longer need to define the accessibility of return-by-reference results in 3.10.2(10).

AI-326/09 Incomplete types

3.2(4) was rewritten. Note 3.9(30) was deleted as it is all too obscure to mention. Changed 3.10.1(2.1) to cover all discriminant parts. Changed "in" to "within" in 3.10.2(2.3) and 3.10.2(2.4). Added wording that the ways that incomplete types can be used is exhaustive after 3.10.2(9.3).

AI-344/09 Allow nested type extensions

Moved the fix to 13.13.2(31) here, as it is related here. 3.9.1(7) applies to all bodies, not just package bodies. Made a minor wording change to 3.9(4).

AI-357/08 Support for Deadlines and Earliest Deadline First Scheduling

Added a rule to require that Ceiling_Locking is used if EDF is used in a Priority_Specific_Dispatching pragma.

AI-360/06 Types that need finalization

Fixed the naming of packages to reflect our usual conventions.

AI-362/06 Some predefined packages should be recategorized

Wording was needed so that Wide_Maps.Wide_Constants remain preelaborated (Maps.Constants is pure, but we don’t want that for the wide versions).

AI-363/07 Eliminating access subtype problems

Dropped possessive forms. Added missing rule to 3.7.1(9). Replaced “defaults” by “default_expressions” in 3.7.1(7), as “defaults” is not a defined technical term.

In 3.7.1(7), the last word “defaults” should be changed to “default_expressions for its discriminants” (there were two places that needed to be changed).

AI-366/08 More liberal rule for Pure units

13.13.1(1) incorrectly defined external streaming for limited types. Added cross-references in Annex E to the definition of external streaming.

Why does E.2.2(17) define the Storage_Size of a remote access type to be zero? You could declare a package to be both Pure and Remote_Types, and if you do (and you gave 'Storage_Size use 0), it would be goofy to say that Storage_Size was undefined.

AI-373/05 Undefined discriminants caused by loose order of init requirements

The bullets were changed to complete sentences.

AI-389/03 Allow aggregates of partial views

This AI was killed earlier in the meeting (see discussion of AI-413), so its corrections were not discussed or approved.

Detailed Review of Amendment AIs

AI-353/03 New Restrictions identifier No_Synchronous_Control

This identifier is subsumed by No_Dependence. Similar existing identifiers are made obsolescent by AI-394, so we certainly don’t want to define any more.

No Action: 9-0-1.

AI-389/03 Allow aggregates of partial views

Discussion of AI-413 uncovered many problems with this AI. As it is late in the process, this AI was dropped. (See the discussion of AI-413/01 for details.)

No Action: 10-0-0.

AI-392/01 Prohibit unsafe array type conversions

This comes from structural array conversion; cases where the types are related by derivation are already covered.

John finds the typo that every AI must have: X (1) and X(1) are inconsistent in the examples. Make these the same.

Approve AI with change: 10-0-0.

AI-393/01 Defaulted generic parameter evaluation and elaboration check

Pascal thought that defaulted generic parameters aren’t covered; Tucker points out that 12.3(20) covers their evaluation. There is an ordering requirement there, we have to be careful to not break that. This paragraph doesn’t make much of a constraint; Tucker thinks that we don’t need to change this. It isn’t broken enough to fix. This is just imposing an order on checks; and there isn’t any order required. Pascal thinks that it doesn’t reflect the meaning of defaulted parameters.

No action: 7-0-3.

AI-394/01 Redundant Restriction Identifiers and completing Ravenscar definition

Should we move old restrictions to Annex J? No_IO should not be moved to Annex J, it covers a set of packages (six, currently).

Add a (user) note in H.4 pointing out possible uses of No_Dependence. H.4(2) says “apply in this Annex”, what does that mean??? No one knows what this means. But we don’t want to rewrite this wording; it occurs all over in this Annex. No_Dependence should be added to this list of restrictions.

Jean-Pierre wonders if No_Task_Termination should require no terminate alternatives. Lots of other things (like abort) could fall in there too. Forget that, Alan thinks that the restrictions are properties of the server, not of clients.

Tucker asks for a cross-reference to the text where the fall-back handlers are defined (C.7.3).

Rearrange the Ravenscar pragma so that positional restrictions precede named restrictions.

AI-353 is now redundant. We should vote it No Action. (See above.)

Approve AI with changes: 9-0-1.

AI-395/04 Various clarifications regarding 16-bit and 32-bit characters

This AI consists of nine separate items. We consider each item in turn.

Item 1: This relates to other_format characters in identifiers and reserved words. In the following discussion, [] represents an other_format character.

Other_format characters usually don’t display; they are things like “soft-hyphen”. (Of course, an editor could have an option to show them.)

Is A_[]_B a legal identifier? (This AI says no.) Is protect[]ed a reserved word or an identifier? (This AI says reserved word.)

Steve Michell asks why we allow other format characters in ids at all. Because Unicode recommends it and it was in what we sent to SC22. And we don't want to change what we sent to SC22 any more than necessary.

In paragraph 2.3(5.2), Tucker does not like "…uppercase version of reserved word". After a lengthy discussion, Tucker suggests "…reserved word (in uppercase)".

Similarly, in 2.9(2) change "…uppercase version.." to "…words (in uppercase)."

Tucker does not want to allow funny characters that happen to convert to reserved words. (Yes, some characters have normal uppercase equivalents.) Otherwise "if" and "acceß" are reserved words. Tucker also thinks we don't need a BNF grammar for reserved words. He takes an action item to provide wording. Attribute designators are insufficiently broken, and they are still reserved words — see 2.9(8), he doesn't believe that they need any change.

Approve intent of item 1: 9-0-1.

Item 2: Soft-hyphen is allowed in string literals now (it is a Latin-1 character); they would not be allowed by AI-285's rules. "ab[]cd" would be illegal. Same applies to '[]'. Other standardized languages with Unicode support (for instance, C#) do allow these sorts of things in strings (limitations only apply to identifiers).

Tucker wonders about operator symbols. He thinks that you would apply the same transformations that we do to identifiers. Erhard concurs. Pascal says he thought the same, but Robert Dewar hated it. Tucker wonders why we can't start identifiers with []. That's because we are following Unicode's recommendations for identifiers; we cannot know accurately why they made those recommendations, and it would be madness to strike off on our own here.

Tucker says that an operator symbol like ">=" shouldn't allow other_format characters; while ones that are reserved words should allow other_format characters. The AI currently says "m[]od" (X,Y) is illegal; Tucker points out that X m[]od Y is OK.

Tucker's point seems to be lost on the group, so he continues to explain. For instance, "mod", "MOD", and "Mod" are all equivalent; why shouldn't "m[]od" or "M[]od" match? The rules ought to be the same whether or not the symbol is in quotes. This doesn't apply to ">=", since other-format characters are not allowed in white space or in compound delimiters. Thus, "<[]=" (X, Y) is illegal; since X <[]= Y is illegal.

Approve intent of item 2: 6-0-4.

Item 3: We need a better definition of upper case. 1.1.4(14.1), change the last sentence of the AARM note to: "It sometimes can make a difference."

Approve item 3 with changes: 10-0-0.

Item 4: Pascal noted that the language-defined names (for the Image of characters that are not graphic characters) are "Character_12345678". That increases the 'Width attribute of the existing types (which is currently 12 for Ada 95) for no good reason. He proposes changing this to "Chr_12345678", which is not liked. Randy comically suggests "Jnk_12345678". Another suggestion is "Char12345678". There is an objection to the last: the digits are hex digits. Thus Characaaacaa or Chardeadbeef are possibilities. Yuck, we need the underscore. Erhard suggests "WWC_12345678". Someone suggests "Hex_12345678". That seems to get general approval.

Approve item 4 with changes: 10-0-0.

After a break, Tucker provides his wording for paragraph 2.9(2) in item 1:

"The following are the reserved words. Within a program, some or all of the letters of a reserved word may be in upper case, and one or more characters in category other_format may be inserted within or at the end of the reserved word."

This is a syntax rule.

The changes in 4.1.4 should be dropped.

Approve item 1 with changes: 10-0-0.

Item 5: We forgot to define Ada.Strings.Wide_Wide_Hash. The text should say "functions", not "packages" for the hashing stuff, because these aren't packages.

Approve item 5 with changes: 10-0-0.

Item 6: The meaning of Lower_Case_Map and Upper_Case_Map need to be defined normatively. So move the existing note to normative text, and add similar normative text to the double wide subclause (A.4.8).

Approve item 6 with changes: 7-0-3.

Item 7: Wide_Wide_Character has 2**31 values, so some fixes are needed. The !proposal says Wide_Character where is means Wide_Wide_Character.

The solution is to a put a size clause on Wide_Wide_Character to force it to 32 bits; we want a signed integer to be able to hold the 'Pos of a Wide_Wide_Character.

Approve item 7: 8-0-2.

Item 8: Pascal discovered that each plane needs to reserve FFFF and FFFE. Erhard would like to change "each" to "in their respective planes" in the wording.

Change 2.1(4) to: "A character whose relative code position in its plane is 16#FFFE# or 16#FFFF# is not allowed anywhere in the text of a program."

Make sure that the inconsistency of 'Image and 'Value is documented in 3.5. That is, non-graphic characters have different names in Ada 95 and Ada 2005.

Change 2.1(3) to: "A character is any character defined within ISO/IEC 10646:2003 other than those whose relative code position in its plane is 16#FFFE# or 16#FFFF#."

Change 2.1(14) to: "…whose relative code position in its plane is neither 16#FFFE# nor 16#FFFF#".

Tucker proposes that the special names FFFE and FFFF defined in Ada 95 be dropped. The compatibility problem is no worse than we already have for other special names, and doing so ought to simplify implementations. There is agreement with this idea.

Approve item 8 with changes: 8-0-2.

Item 9: Randy notes that Ada.Characters.Handling has wide character stuff. There is a compatibility issue with the new functions; calls with literals or overloaded functions may become ambiguous. Should there be an Ada.Wide_Characters.Handling?

It is suggested that implementations provide these packages if needed. Steve Michell thinks that packages not in the standard are going to be ignored.

Pascal suggests removing all of the changes to Ada.Characters.Handling. Jean-Pierre Rosen suggests defining the empty packages Ada.Wide_Characters and Ada.Wide_Wide_Characters, since Ada.Characters is empty. We'll put those into A.3.1. And add implementation advice that they be used as parent units for services specific to these character types.

Approve intent of item 9: 10-0-0.

After the vote, Tucker asks why characters should be different from Ada.Strings, in which everything is under Ada.Strings. The child packages are Wide_xxx for wide versions. There is no consensus for a change here, apparently the organization of Ada.Strings is not widely liked.

On Sunday, Randy asks for reconsideration of the resolution of item 9, as he thinks that we may have tossed the baby with the bathwater. With this resolution, if you have a String, and you want to get a Wide_Wide_String, you don't have a language-defined way do it. That seems like a nasty hole, given that the intent is that these are different views of the same thing. He goes on to say that the problem seems to be one of too many unrelated things being stuck into one package (Handling). That's especially bad for the cross-cut functions that do conversions (they don't clearly belong to any particular type). It would have been much better to have Ada.Characters.Classification, Ada.Characters.Conversions, and Ada.Characters.Casing; rather than just Ada.Characters.Handling.

That's too much change at this point, so Randy suggests simply creating Ada.Characters.Conversions, which would contain A.3.2(14-18.6). He'd suggest moving the existing functions into Annex J, but it seems odd to move part of a package there. Someone points out that package Ascii is part of a language-defined package, and it is obsolescent. So we do have a precedent for doing that after all.

The group agrees that should move all of the existing conversion functions into Annex J, and turn them into renames of the new ones (defined in A.3.4).

There is no interest in the idea of suggesting in implementation advice (similarly to the advice for Ada.Directories) that implementations provide Classification children.

Approve intent of item 9: 7-0-1.

Approve intent of AI as a whole: 9-0-1.

AI-396/01 The "no hidden interfaces" rule

[Aside: Snow starts falling outside, along with thunder. Pascal tells us snow never sticks in Paris. Soon, the roofs and grass are snow-covered. The snow turns to sleet; by the time we finish work on this AI, we decide to order delivery pizza for lunch. The decision on which pizzas to order takes longer than some AIs, and of course the snow stops and melts as soon as they are delivered.]

Typo in proposal: "…to be completed…".

Tucker says the wording is confusing as written; he proposes using

"A full view shall be a descendant of an interface type only if the corresponding partial view (if any) is also a descendant of the interface type, or if the partial view is untagged."

Erhard says that descendant is defined on types; this wording talks about "views". Tucker thinks that everywhere we say "type" we probably mean "view of a type". But we're not going to reword the entire Standard to fix this.

Steve Baird says that the following ought to be illegal:

```
      type T is new Parent and Ifc with private;
   private
      type T is new Parent with null record;
```

This doesn't seem to be covered by Tucker's proposed wording. So "if and only if" *is* needed.

"A full view shall be a descendant of an interface type if and only if the corresponding partial view (if any) is also a descendant of the interface type, or if the partial view is untagged."

Approve AI with changes: 9-0-1.

AI-397/01 Conformance rules and overriding indicators for entries and protected operations

"Implements" is different than but similar to "overriding", so we want to allow both to have indicators.

Tucker suggests making the technical term "prefixed view profile", to match AI-407.

Erhard is worried about components of access-to-subprograms. He shows an example:

```
   type T is tagged
      record
         ...
         P : access procedure (...);
      end record;

   procedure P (A : T; X, Y : Integer);

   O : T;
   O.P (2, 5); -- Which P?
```

There is a preference for components; else prefixed views would be too incompatible with Ada 95.

The second sentence of 6.3.1(24.1) should say "There is no prefixed view profile for a parameterless subprogram."

Tucker would like to add an English syntax rule that an entry family does not allow an overriding indicator.

Why aren't indicators allowed on entry bodies? We don't need this on them, because "implements" cannot happen "late". And we certainly do not want to have to allow indicators on accept statements.

Steve Baird asks if you can requeue on a synchronized procedure. No, we don't allow that, because the synchronized procedure doesn't necessarily denote an entry. And what such a requeue would do if the procedure wasn't an entry is rather unclear.

Pascal will provide an example for this AI.

In 9.1(9.3), "he task type" should be "the task type".

Approve AI with changes: 10-0-0.

AI-398/01 A formal parameter should be usable only once

After a couple of false starts on explaining this, Pascal writes a short and incomplete example of what this AI is talking about. It is a formal parameter like:

```
   with package P is new G (X => ..., X => ..., <>);
```

That isn't covered by any existing rules because this is not a normal instantiation. It was a normal instantiation (syntactically) in Ada 95, so it was covered there. Pascal proposed wording that fixes this hole.

The second rule covers:

```
   with package P is new G (X => 1, Y => <>);
```

All other parameters (if any) better have defaults. These two rules are equivalent to the ones in 12.3(10).

Do we need to duplicate 12.3(9) as well? No, because generic_association is still used here, so 12.3(9) still applies.

The subject of the AI should be changed. Use “A parameter of a formal package should be given at most once”.

Fix the !discussion; it should be “(See wording.)”

Put something in the example (like the above).

Tucker would like some sort of annotation that this is based on Draft 10. The AI should avoid being based on any AARM draft, rather it should be based on other AIs and the existing Standard.

Approve AI with changes: 10-0-0.

AI-399/01 Single task and protected objects implementing interfaces

Add (See discussion.) under !example.

Pascal is certain that this does not introduce any new problems.

Add **is abstract** to the definition for Put.

Approve AI with changes: 8-0-2.

AI-400/01 Wide_ and Wide_Wide_ variants for the subprograms that return the image of identifiers

Randy asks for a shorter title. John suggests “Wide_ and Wide_Wide_ images”.

Pascal says that these functions returns image of identifiers. Identifiers now can include Wide_Wide_Characters, and we can’t allow garbage to be returned.

Erhard wonders if we need to clutter these packages with this stuff, as opposed to making child packages. Pascal replies that we are only changing a handful of things that return identifiers, not exception messages, and so on.

Steve Michell asks why we don’t allow wide-wide file names. The short answer is that no one has asked for it. That leads into a long discussion, which concludes that an implementation could implement a UTF-8 format for file names via a Form if it needed to do so.

John notes that 11.4 should be 11.4.1 in the AI (three places).

Approve AI with changes: 10-0-0.

AI-401/01 Terminology for interfaces

We should change subtype_indication to subtype_mark in interface_lists. We don’t allow constraints on interfaces, after all. Then we need to change 3.4(3/2): “defines” to “names”. Once that is a subtype_mark, we don’t need to define elaboration for interface_lists, as subtype_marks are not elaborated, so we can remove the paragraph that the AI adds after 3.4(26). [Editor’s note: This was an error in the AARM; the Amendment and AIs use subtype_mark.]

3.4(8) should say “…or {a} progenitor type…”.

Erhard has volunteered to review the wording after it is integrated into the AARM.

Approve AI with changes: 7-0-3.

AI-402/01 Access discriminants of non-limited types

As things currently stands, the rules for limited and non-limited access discriminants are different. That makes it dangerous to switch between limited and non-limited types that have access discriminants.

There were a large number of alternatives suggested; the rule chosen is to disallow defaults on non-limited types (or types that might be non-limited).

Why is there no check for allocators? Tucker claims that there is an existing rule, but no one can find it. Moreover, he added one in AI-416 (unclear as to why). The text of AI-416 is just a tad confusing as it contains rules for both allocators and return statements.

The rule on allocators in AI-416 really belongs in this AI, because it exists mainly to prevent problems with accessibility of these discriminants. But we don’t move it.

AI-354 should be AI-344. Summary contains “whih”. The second bullet in the discussion has quote marks which are junk; also in the fifth bullet.

Approve AI with changes: 7-0-3.

AI-403/01 Preelaboration checks and formal objects

Typo: “wou” should be “you”.

We can’t figure out why this would need an extra rule. The formal object is never static. Why then are formal subprograms covered here, since they aren’t ever static?

This rule (10.2.1(10)) is wrong anyway; it needs to say that the private type should not have preelaborable initialization (it wasn’t updated by AI-161).

The AARM note is very confusing. It needs to say that we need an assume-the-worst rule to allow preelaboration on generic units, so we have this rule.

This rule is telling us to use an imaginary worst case instance, and check that. 10.2.1(8) doesn’t apply in this case, because the elaboration of the generic body does nothing at all. We certainly don’t want to allow everything in a preelaborable generic unit.

So we agree that we do need this rule (although it took an hour to figure that out). The AARM should explain why we need this rule and how it works, so we don’t have to spend an hour figuring it out again in the future.

Erhard notes that we need to say something about formal packages. Probably use words like “any formal type declared in the formal part” that we’ve used before. Formal types always need to be assumed to be nonstatic, as well.

Pascal will take this AI and try to fix this wording.

Approve intent of the AI: 7-0-2.

AI-404/01 Not null and all in access parameters and types

Straw poll on dropping **all** from general access: 9-1-0.

Randy opposes because these are general access types, and they should say so. Pascal and Tucker claim that they really are different from general access.

No one changes his vote.

Tucker writes an example to illustrate the issues with **not null**:

```
package Pack is

   type T is tagged ...
   procedure P (X : access T);
   procedure P2 (X : not null access T);

   package Pkg is
      procedure Q (X : access T) renames P; -- Legal?
      procedure Q2 (X : not null access T) renames P2;
      procedure R (X : not null access T); -- Not dispatching.
      procedure S (Y : access T);
   end Pkg;

   -- The following are primitive:
   procedure X (Z : access T) renames Pkg.R; -- Legal?
   procedure U (V : access T) renames Pkg.S; -- Legal? (Legal in Ada 95).
   procedure W (X : not null access T) renames Pkg.R;

end Pack;
```

Tucker proposes that you shouldn't be able to give **not null** on controlling parameters. Randy objects because you can't see syntactically that a parameter is controlling; the maintenance programmer needs to know whether a parameter is **not null** without having to remember to figure out whether it is controlling.

Tucker asks what are the semantics of P when it is not a dispatching operation (as in the rename Q)? We could change it to allow **null** in a non-dispatching call. Pascal complains that that would cause one to drop into a different handler by a minor change to the code. That's ugly.

So we agree that there is an implicit **not null** and it is enforced always.

Tucker lists the questions to be answered:

1) Can the **not null** for a controlling parameter be given explicitly?

2) Can we rename these as in Q?

3) What does U do? Tucker suggests that that U should be illegal.

The choice for (3) would be incompatible. But why would someone carefully declare something in a nested package to make it non-primitive, and then rename it to be primitive?

Tucker suggests that every dispatching operation is null excluding, and every rename must be null-excluding. That would be a legality rule. A similar rule is needed for generic instantiations.

For (1), we should allow **not null**; the group agrees that it's useful at least for documentation, and we can imagine a compiler switch to warn if it is not given on a controlling parameter.

For (2), it seems that Q is legal, and gets its null exclusion from the renamed subprogram. This is similar to saying Positive for a parameter of subtype Natural in a rename; that works and Positive will be ignored. We want this to work the same. Tucker suggests not allowing **not null**, but that would make it impossible to copy the specification of a subprogram. That would be annoying.

Gary would like to say that if you say **not null** in a renaming, then it must be a **not null** parameter. He says that we should get this right, rather than being wrong. But the standard is already screwed up, and there isn't much benefit to fixing this one only.

Tucker will try to update this AI tonight.

Steve Baird notes that the renames rule also needs to apply to generic instances as well.

Approve intent of AI: 9-0-1.

AI-404/02 Not null and all in access parameters and types

In 3.9.2(11), change "controlling result" to "controlling access result".

Also, 3.9.2(11) should say "generic_instantiation of a subprogram", it doesn't apply to operations declared inside a package. "instantiation of a generic subprogram" reads better.

Erhard is concerned that controlling access results could not return null. Mechanical changing of named types to anonymous types won't necessarily work. Randy suggests that query functions should be T'Class, and constructors aren't ever going to return null.

Pascal says that changing from return T to return access T seems like something people would do, and we certainly would hope that the dispatching would work the same way for either notation.

Jean-Pierre is confused about the renaming rule.

```
type T is tagged ...
procedure P (X : access T);

procedure Q (X : access T) renames P;
```

You have to change P. The rule seems to be confused.

Steve Baird suggests "shall already be". Gary suggests saying "dispatching subprogram" rather than "dispatching operation". Tucker takes an action to update this.

Approve intent of AI: 8-0-2.

AI-404/03 Not null and all in access parameters and types

Tucker reads the new text for 3.9.2(11).

Approve AI: 6-0-2.

AI-405/01 Mechanism for scanning the progenitor interfaces in Ada.Tags

Tucker would prefer a list of all of the interfaces that are ancestors (not including the type itself), not just the ones that happen to be at the top level. That's what a compiler is likely to have available, anyway.

Four people suggest a strong preference for returning an array. We will do that. The order is unspecified.

Call the function Interface_Ancestor_Tags and the return type Tag_Array.

Approve intent of AI: 7-0-3.

Erhard comments that he hates the name "Process" for iterators. Alan says he said that before, and he didn't get anywhere. Erhard suggests "Action". This discussion is met with groans from the rest of the group.

Randy will update this AI tonight.

AI-405/02 Determining the ancestor interfaces in Ada.Tags

Randy explains the Beach Boys remark in the AI by singing part of "Wouldn't it be nice…". He obviously had too much wine last night. Make sure that remark is removed from the AI. Steve Michell suggests replacing the question by "There are occasions when the user needs to know the tags of the interfaces that are implemented by a type."

Erhard asks why we are returning all of the interfaces rather than just the progenitors. Pascal says that the progenitors may not be available to the implementation at run-time (the ancestors are what is needed to implement type conversions and memberships). Jean-Pierre would like to be able to figure out the entire hierarchy. The position of an interface in the hierarchy doesn't have any interesting semantics; so there is no value to knowing the hierarchy.

Jean-Pierre asks if a parent interface is included. Randy says yes, and he had forgotten to add an AARM note to that effect.

Steve Baird asks that wording be added so that no interface occurs more than once in the returned array. Tucker suggests says adding "exactly once" to 3.9(12.3).

Add "other than T itself" to 3.9(12.3), T is not included.

Randy had tried to think like Steve Baird, and thus wondered what should happen if the tag is bad in some way. We would hope that that didn't put any burden on implementations. Certainly 3.9(26) applies, so if the value is a legitimate tag of this partition, Tag_Error can be raised, or the function should work properly. If the tag is garbage, it should be abnormal (as the result of an Unchecked_Conversion, or reading junk). Reading an abnormal object is erroneous, so what these functions do is irrelevant. But 13.9.1(6) doesn't seem to cover 'Input (which is a function, not a procedure). So its result cannot be abnormal, which is a pretty amazing result, given that it could be given any old junk in its stream.

Tucker thinks that 13.9.1(6) doesn't cover stream attributes at all, as they aren't "language-defined input procedures"; they don't necessarily have anything to do with input (you can stream from a memory buffer, like the Windows clipboard, after all). The list in the paragraph seems to be exhaustive, but it doesn't include Unchecked_Conversion, or dereferencing an address-to-access conversion. It should include those cases, stream attributes, and say "language-defined input subprograms" in order to cover function results.

Randy will create an AI.

Approve AI with changes: 9-0-1.

AI-406/01 Aliased permitted with anonymous access types

Remove "Remember that…" from the discussion because it isn't true.

Approve AI with change: 9-0-0.

AI-407/02 Terminology and semantics of prefixed names

Erhard does not like "prefixed name", because it doesn't seem to be clear what it is. He suggests "method name". That is not well liked.

Pascal notes that this is often known as "distinguished receiver notation", but that refers to the object, not the entire name.

Tucker had suggested "prefixed subprogram name". Gary suggests "prefixed operation name". Those seem too long.

Add a cross-reference in 6.4(10): "prefixed name (see 4.1.3)".

Someone suggests defining "prefixed view" in 4.1.3(9). Indeed, the way that "prefixed name" is used, that seems like all we need. Pascal doesn't believe that.

Tucker suggests turning around the wording of the legality rules: "For any subprogram, the prefix of the prefix view…".

Approve intent of AI: 10-0-0.

Pascal will provide an updated AI tomorrow.

AI-407/03 Terminology and semantics of prefixed views

Pascal says that he changed consistently to “prefixed views”. This meets with general approval. There is a discussion of an e-mail comment by Bob Duff, and a similar comment by Jean-Pierre as to whether the “prefix of a prefixed view” includes the implicit dereference. We conclude that the answer is yes, and that they were confused.

Erhard wonders why the wording in 6.4 says prefix. That follows from the syntax of a subprogram call. Tucker wonders if there is a need for an AARM note to explain this; such a note is not necessary because that is what all of the paragraphs in this part of 6.4 say.

Approve AI: 10-0-0.

AI-408/01 Visibility for attribute_definition_clauses

Tucker explains that the existing wording is wrong: “immediately” visible is wrong, and saying that this is similar to a visible declaration has problems with homographs.

Typo: We need parenthesis around (See proposal.)

Tucker would like to say “... everywhere within its scope”. That’s OK.

8.3.1 should be 8.3 in the AI.

Add a note that no example is necessary, referring the reader to AI-195.

Approve AI with changes: 8-0-2.

AI-409/01 Conformance and overload resolution related to anon access types

In 6.3.1, we need to say something about profiles for anonymous access types, especially anonymous access-to-subprogram types.

For renaming of an object with an anonymous access type, the null exclusion comes from the renamed object, so we disallow giving it explicitly.

In paragraph 8.5.1(3/2), the wording is clear as it is, just drop “specific” and change subtype conformance to type conformance. Don’t rewrite wording for the heck of it.

In paragraph 8.5.1(4/2), reword to split into access-to-object and access-to-subprogram parts, rather than the current object, subprogram, object wording.

The 8.6(25) change is incompatible with Ada 95, but we can’t figure why we would want this in the first place.

The existing wording for 8.6(25) ends with “; or”, which needs to be preserved.

Approve intent of AI: 7-0-3

On Sunday morning, Tucker gives his rewording for 8.5.1(4/2):

In the case where the type is defined by an access_definition, the type of the renamed object and the type defined by the access_definition:

- shall both be access-to-object types with statically matching designated subtypes and with both or neither being access-to-constant types; or
- shall both be access-to-subprogram types with subtype conformant designated profiles.

Approve AI with changes: 8-0-2.

AI-410/01 Limited with should be allowed on bodies

Tucker claims that this would be useful for pragma Elaborate.

Pascal reports that the minutes of the Toulouse ARG meeting state that “These problems get worse if we can have procedure calls. We are glad that we can’t have **limited with** on a body.” We can’t determine what “these problems” are, or whether we still have them.

Jean-Pierre asks for an example, so Tucker writes one:

```
package P is
   pragma Elaborate_Body;
   procedure M;
end P;

limited with Q;
with Q2;
package body P is
   procedure M is
      N : access Q.T := null;
   begin
      Q2.R (N, 3);
   end M;
end P;

with P;
package Q is
   type T is tagged ...
   ...
end Q;

limited with Q;
package Q2 is
   procedure R (X : access Q.T; Y : Integer);
end Q2;
```

In the above example, the “**limited with** Q” on the body of P makes it possible to find an elaboration order: having a “**with** Q” would prevent the code from linking. But it’s hard to get excited about this. Tucker says that a workaround would be to put a “**limited private with** Q” on the package specification. That seems good enough.

No action: 10-0-0.

AI-411/01 Equality for types derived from interfaces

Steve Baird asks if we really need to mention limited interface in 7.5. The word **limited** is included in the declaration of a limited interface, and any descendant has to get it from the parent. Randy says that a task interface is not covered by the existing wording. We decide to leave Tucker’s wording.

Steve Michell would like to change the title to “Predefined equality…”

Part of the proposal section should become the wording section for the AI.

Approve AI with changes: 10-0-0.

AI-412/02 Subtypes of incomplete types; renamings of limited views

Tucker explains the problem and the proposal. The last question needs to be decided in any case.

Steve Baird asks if there is a case where the subtype name and type name could denote different things or have different properties. Hopefully not. Tucker writes an example:

```
package Q is
   type T is ...
end Q;

limited with Q;
package P is
   subtype S is Q.T;
   package PQ renames Q;
end P;

with P;
package R is
   ...
   X : P.S; -- Legal or not? (1)
end R;
```

(1) is OK if there is **with** Q on R; illegal if **limited with** Q; on R; illegal if no **with** or **limited with** Q on R.

```
with Q;
package N is
   package NQ renames Q;
   subtype NS is Q.T;
end N;

with N;
with P;
package R2 is
   ...
   X : N.NQ.T; -- OK, unless a limited with Q is added.
   Y : N.NS; -- Legal always.
end R2;

with P;
package R3 is
   ...
   X : P.PQ.T; -- PQ is full if with Q; is given on R3;
               -- limited if limited with Q;
               -- no view if neither (the name P.PQ is illegal).
end R3;

with N;
with P;
package R4 is
    ...
    Z : N.NQ.T; -- OK, Ada 95 rules gives us a full view.
    X : P.PQ.T; -- Illegal, no view; otherwise we would depend
                -- on the view that N has (to avoid having two
                -- different views in scope), which is a ripple effect.
end R4;
```

Jean-Pierre asks if adding a limited with will cause a loss of visibility. Yes, in uncommon cases, but there is an architecture problem with the program in that case. The unit is in the scope of the full view, so why would it possibly need a limited with? Limited with is not some sort of general feature, after all.

Pascal asks that there is something to tie the name of subtype where the name of an incomplete type is used. No, the new wording talks about “incomplete view”, not the type.

Pascal asks about the wording in 8.3(20): “All views of the package are limited.” That seems to imply that the full view is a limited view. Pascal suggests rewording this as “All renamings denote the limited view.” That doesn’t seem to cover all of the possibilities. We decide on “Any name that denotes the package denotes the limited view.”

8.5.3(3) should be reworded as: “ If the package_name denotes a limited view of a package, then a name that denotes the package_renaming_declaration shall occur only within the immediate scope of the renaming or the immediate scope of a with_clause that mentions the package or (for a nested package) the nearest enclosing library package.” 8.5.3(4) should have a Proof AARM note that references 8.3 where the rule really is.

Approve AI with changes: 5-0-5.

AI-413/01 Partial view, task, and protected aggregates

We decide to discuss `others => <>` for records (ignoring all of the items mentioned in the title of the AI) first. That should be relatively easy to decide. (No chance. ☺)

Pascal explains how we got to where we are, i.e., “`others => <>`” matches anything. We also start talking about “<>” used to provide uninitialized integer components. Tucker would like to allow <> on composite only; if the inner components are uninitialized, that’s OK because the designer of the composite abstraction presumably intended that.

Tucker enumerates the possible rules for <>:

1) <> is allowed for uninitialized scalars (that is, for everything);
2) <> not allowed for uninitialized scalars but is allowed for composite types;
    a) in records;
    b) in records and arrays;
3) <> not allowed for uninitialized scalars and composite types with them (privacy breaking).

`others => <>` is:

1) not allowed for records (OK for arrays);
2) not allowed if uninitialized scalar components;
3) not allowed if uninitialized scalar SUBcomponents (privacy breaking);
4) allowed where <> is allowed;
5) not allowed for arrays or records.

Forget both 3’s, as they are generic contract and privacy breaking. Presuming Data is a String:

```
Text'(Size => 0, Data => (others => <>)); -- Illegal with 2, 5.
Text'(Size => 0, Data => <>); -- OK.
Text'(Size => 3, Data => ('A','R','G',others => <>) -- Illegal with 2, 5.
```

The third case would be useful, but Pascal is willing to give up on it. We reduce the possibilities to a shorter list:

A. Disallow <> and **others** => <> for uninitialized visible scalar components.

B. Allow Name => <>, but disallow **others** => <> for uninitialized visible scalar components.

C. Anything goes for <> and **others** => <>.

D. Allow only if guaranteed fully initialized.

We take a "can live with" straw poll: A: 7; B: 4; C: 8; D: 5. Since that was so conclusive, we take a preference poll: A: 5; B: 0; C: 4; D: 0; Undecided: 1.

We decide that **`others => <>`** is allowed for records. We'll decide on the semantics of <> later.

Tucker explains the model he is proposing for partial view aggregates. Essentially, they have their own syntax, so we separate issues of completeness from partialness. Randy complains that that reintroduces the disappearing aggregate problem. That is, an aggregate that is legal for the partial view is not legal for the full view. It is uncomfortable for a partial view to have properties that the full view does not have. Steve Baird points out that using this notation in a default parameter could not be completed by any possible subprogram in the scope of the full view; that seems nasty.

Pascal says that what we had does have the property that the full view can use the same aggregate as the partial view. Erhard says the problem is that full coverage is lost for private types; you have to turn off coverage checking to get partial views. Since one of the major benefits of aggregates is coverage checking, this also is uncomfortable.

Steve Baird says that he thinks that with Tucker's proposal, changing from a partial view to a full type would cause aggregates of the partial view to be illegal. He doesn't think that a partial view should provide operations that the full view does not have.

We seem stuck here; either we lose coverage checking or we give properties to the partial view that the full view does not have. Pascal would like to speak in favor of dropping partial view aggregates altogether. Randy notes that this was intended to be a simple extension, and it doesn't seem simple anymore. It seems to be taking a lot of language, and the benefit isn't big enough.

Therefore, we decide to kill both 389 and 413.

No action for AI-413: 10-0-0.

We return to deciding the meaning of <>; we have to choose between the alternatives A and C presented previously.

Erhard presents an example. Given:

```
type T is tagged record ...;
type TT is new T with
   record
      A, B, C : Integer;
   end record;

(T with A | B | C => 3); -- OK.
(A | B |C => 3, others => <>); -- OK with C, depends with A.
(T with A | B => 3, others => <>); -- OK with C, illegal with A.
```

The second aggregate's legality would depend on the components given for T. (These have to be visible now.)

Steve Baird is concerned about the semantics of <>. Say the types above are declared as:

```
type R is
   record
      F1 : Integer := 10;
   end record;

type T is tagged
   record
      F2 : R := (F1 => 11);
      F3 : Float;
   end record;

type TT is new T with
    record
       A, B, C : Integer;
    end record;
```

He thinks that <> would get the wrong value (10 rather than 11) for F2, but that seems to be a mistaken understanding of the rules. If a default expression is given, that is always used before dropping to default initialization.

Erhard comments that he thinks that the three aggregates should have the same meaning, and thus you must choose C. But it is true that A is OK by this measure; if the aggregate is legal, it has the same meaning. The only difference is whether or not it is legal.

Pascal comments that choice A gives some freedom to the maintenance programmer who adds a new component; if a default value makes sense, give it; else, the component can be added without it, and users will have to change their aggregates. In neither case, will anything bad happen.

Steve Baird worries that choice A could cause problems in generics; <> could be illegal in a generic specification (for a formal private type). Tucker says that it would have to be an assume-the-worst in the generic body; <> would not be allowed for generic formal private components. Yuck. Randy notes that we'd need a special rule to make this illegal in the generic body; a formal private type is assumed to be composite and thus would allow <>.

Gary suggests (somewhat as a joke) that if a <> appears in the specification, it can be allowed in the body. That is met with groans from the implementers at the table.

Tucker is asked to write an example of what he and Steve are talking about. He obliges:

```
generic
   type T is private;
package P is
   type R is record
      X : T;
   end record;
end P;

package body P is
    R'(X => <>); -- Illegal if choice A is selected.
end P;
```

The <> would have to be illegal because T could be instantiated with an elementary type. As a work-around, the programmer could give a default expression for X (with an object or by passing in a function).

We discuss the "implicit variant" sort of record. This is a record where the value of some component (a pseudo-discriminant) controls whether the value of some other component is meaningful. In that case, the components controlled by the pseudo-discriminant would not be initialized in the default case. But that might be inappropriate for some values of the pseudo-discriminants.

John says that formal analysis people don't want to have to use junk initialize values. They add paths and hide errors.

Tucker thinks aggregates should have the property that giving a reasonable default means that you don't need to revisit all the aggregates. That's important for maintenance; similarly, it is best if not giving a default requires revisiting aggregates. But we can't mandate the latter.

We again try a straw poll: A: 4; C: 4 Undecided: 2.

We seem hopelessly deadlocked. Randy notes that we need a consensus to change AI-287 (which is already WG9 approved), and we don't seem to have that. (And two hours have passed without getting closer to a consensus.) AI-287 uses choice C. After additional discussion, Pascal reluctantly agrees, and we decide to move on.

AI-414/01 pragma No_Return for overriding procedures

This replaces the wording of 6.5.1, but not the rest of AI-329.

Jean-Pierre comments that pragma No_Return should not be allowed for null_procedures; they couldn't have the correct semantics. It's OK for abstract subprograms, since No_Return can be inherited.

Approve with changes: 8-0-2.

AI-416/01 Access results, accessibility, and return statements

[This was the first AI discussed at the meeting.]

Tucker explains that access returns can be tag-indeterminate.

Pascal asks if `function Empty_Rec return function access T;` is primitive for T. We hope not.

Tucker explains accessibility of return objects. He doesn't want to have to pass in the accessibility level of the return objects. Randy notes that doing that would present a safety issue; the function might work or fail depending on how it is called and used. So static levels are preferred.

Tucker explains that a return object is in between levels — deeper than the outside, and shallower than locals to the function. Steve Baird says that if there is a local variable that points at the return object, then we need the black-hole model (must return once return starts); else you could get a dangling pointer. We certainly don't want the dangling pointer. If we don't have the black-hole model, the return object must be more local than the locals.

Erhard would like function returns to stay similar to procedure **out** parameters. Tucker claims that they essentially are.

We discuss the black-hole model. The problem is that once the object is created, some sort of premature finalization/task waiting is required if an exception is raised and another return can be started for the same return object. Pascal points out that this is an issue for limited regular returns (which are build-in-place), as well as extended return statements. There is also a compatibility issue with non-limited types; exceptions in non-limited return expressions can be handled locally, and we wouldn't want limited types to be different.

Since we're not using the black-hole model, we have to prevent dangling pointers to the return object. Steve Baird suggests that accessibility of the return object is "incomparable" — it matches nothing. In Tuckers list of OK for accessibility, the second bullet is changed to not OK; add another bullet for things local to extended return statements.

Steve Baird asks about:

```
type T is array (Positive range <>) of Tsk;

function Blah return T;

A : T(1..10) := Blah;
```

What if Blah tries to return an array of 20 tasks? It should fail because the storage pool passed in will only allow allocating 10 tasks. But we need to allow raising the exception "early" in that case; we don't want to have to create some sort of temporary specifically to allow creating and messing with 20 tasks only to later raise an exception. There also is a concern that raising Storage_Error is the wrong answer here; it really is violating a constraint.

We turn to discussing the rules for allocators. The check for class-wide objects already exists (from AI-344). The access discriminants case is currently interesting for limited access discriminants (the non-limited case is in AI-402, which we will discuss later).

```
type Lim (Disc : access T) ...
type Acc_Lim is access Lim;

begin
   declare
      Q : aliased T;
      function Flim (...) return Lim; -- Could use Q.

      Y : Acc_Lim := new Lim'(Flim(...)); -- Illegal
         -- (because the access discriminant could point to Q).
```

Suddenly reverting to a previous topic, Steve Baird notes that you have to finalize when you leave on a failure, because the finalizer could be declared in a nested scope, and may not exist in the caller. (This case would require an accessibility check failure.)

Erhard thinks that you can’t get out of a return statement; it is just like an exception handler, which you can’t get out of. Pascal, Jean-Pierre, and Steve Baird suggest that converting Ada 95 code to use extended return statement would change exception handling. That seems bad. Tucker writes an example to make it clear what we are talking about:

```
begin
   return (...);

   return R : T := (...) do
      ...
   end return;

exception
   when others => ...
      -- Can this handle the exceptions from the second return
      -- statement? It could handle exceptions from the expression
      -- of the first return statement.
end;
```

This makes it clear that the black-hole model would be too inconsistent with existing code.

Jean-Pierre asks about a goto out of an extended return. The object is finalized whenever you exit other than return; it’s like a local block.

We review the wording changes. 3.9.2(18.1) reads oddly; Tucker will look at rewording the paragraph as it is new anyway.

3.10.2(10) is this “nearest” or “innermost”? Tucker will check existing wording.

Why is constrained subtype mentioned in 3.10.2(14)? Tucker writes an example:

```
subtype S is T(Obj'Access);

new T (Obj'Access);
new S;
```

The two allocators should be treated the same. But this wording is still goofy; subtypes aren’t created, they’re elaborated.

Erhard suggests saying “created by an allocator and its constrained subtype” here.

4.8(5.1) defines accessibility levels. Accessibility levels should *always* be defined in 3.10.2. That may have been a bad design of Ada 95, but we should stick to it. Tucker claims that we should not move these rules, because they apply only here. But that’s true of many accessibility level rules; they’re still in 3.10.2.

You need at least some of these rules for return statements:

```
return X : T (Obj'Access) do ...
```

This certainly is allowed for unconstrained subtypes.

John notes that uninitialized_allocator and initialized_allocator are not syntactic categories.

For 4.8(7/2), Jean-Pierre asks why the “such”? Tucker thinks it is necessary in the full context.

Erhard suggests combining the 9.2(4) rule with the allocators rule, since they are the same. So this should replace 9.2(4), and cover allocators. Allocators should be mentioned as a note. Subcomponents are initialized before the enclosing object, of course. Mention that in the AARM note.

Steve Baird is still thinking about the unconstrained subtype issue. He has a new, nastier example (and no one is surprised):

```
type T1;
type T2 (D : access T1) is ...
type T1 (D : access T2 := new T2 (D => T1'Access)) is ...
type T1_Vector is array (Positive range <>) of T1;

function F return T1_Vector;

X : T1_Vector (1..10) := F;
```

A storage pool model doesn’t work. But that isn’t a language rule. We just need an implementation permission to raise the exception early. Steve doesn’t want that to be an implementation permission, because that could change the handler of the exception. Ugh. We don’t want the handler to change, but we do want the implementation to be allowed to raise it early. That means that the early exception has to go to the right —non-local — handler). This would be similar to Ada 95, where 11.6 gives us the permission to avoid evaluating things solely for their side-effects.

Approve Intent of AI: 10-0-0.

Tucker will try to create AI updates for tomorrow.

AI-416/02 Access results, accessibility, return statements

Tucker explains that accessibility of access discriminants is all wrong.

```
X : T(Disc => A'Access); -- Accessibility check as part
                         -- of conversion to anon access.

Y : T := X; -- Where does the accessibility check occur??

return (D, D2, ...); -- Especially here.

new T'(D, D2, ...); -- And here.
```

So he changed the wording to make that happen.

In 3.9.2(18/2), change to “If the call (possibly parenthesized or qualified) has a controlling result …”

Make the same change in 3.9.2(18.1/2): “If the call (possibly …”

Gary asks if “access result” in these paragraphs means “controlling access result”. Yes, and both paragraphs should say that. Gary asks why “controlling result” can’t include “controlling access result”. Tucker thinks that would cause trouble with other rules. We decide not to do that.

3.10.2(9) should have an underscore in “qualified_expression”, and no comma after “parenthesized_expression”.

3.10.2(10.1) should just say “within an extended_return_statement…”

We get side-tracked on 7.6.1(3). Why doesn’t it just say compound_statement? Most of them wouldn’t do anything interesting, but that seems harmless and simpler.

3.10.2(10.1/2) shouldn’t say “Redundant”, because this seems to be an important rule!

3.10.2(10.1/2) should say “return_statement”, not “extended_return_statement”. (Pascal wants the syntax of return statements changed, but that’s not relevant here.)

3.10.2(10.1/2) should have a To Be Honest note that you do not wait for tasks or do finalization on this object before returning (which will change the master).

The idea behind 3.10.2(12/2) is that if you know the value of the discriminant, the level comes from that. Given:

```
   subtype S is T (A'Access);
```

The following are equivalent:

```
    ... new S;
    ... new T(A'Access)
```

This is not incompatible; it allows more than in Ada 95 (in the cases that were possible in Ada 95).

The third bullet is talking about the caller. It would be something like: F().X.Y.Z.Acc_Disc.

“That” seems to tie to subtype, not object. Change “that” to “and the object” in the third bullet.

Steve Baird is worried about allocators as discriminants, because those are supposed to be “bound together”. The hand-off rule has to be careful.

Tucker wonders about “**null**” as well; that seems to have a problem.

The paragraph added after 6.5(5.3) includes the missing static check for class-wide types. (Remember that accessibility checks usually come in pairs, with both static and run-time checks.)

In the AARM note for 6.5(5.5) replace “e.g.” by “for example,” .

Pascal makes an unfriendly reading of the Implementation Permission. The intent is that this gives a permission to raise the exception early, but at the normal point that the exception would be raised.

Drop limited, even through it is minor inconsistency. (Side effects during the evaluation of a return expression may not happen. But 11.6 might actually allow those to not happen anyway, because you never have to evaluate something only for a side effect. And it could only matter in a program that raises an exception during or after a return expression.)

“If the result subtype of a function is unconstrained, and a call on the function is used to provide the initial value of an object with a constrained nominal subtype, Constraint_Error may be raised at the point of the call while elaborating a return_subtype_indication or evaluating a return expression within the function if it is determined that the value of the result will violate the constraint of this object’s subtype.”

This is less likely to have an unfriendly reading.

Paragraph 7.6.1(3) is fixing a bug noticed by Adam Beneschan; we don’t want a type declaration to be a master.

Paragraph 7.6.1(13) is unchanged; get rid of the header in the AI.

9.2(4) needs to be updated as previously discussed.

Approve intent of AI: 7-0-3.

AI-416/03 Access results, accessibility, return statements

[This was the last AI discussed at the meeting.]

Tucker describes the changes he made to the AI.

In paragraph 3.10.2(14), change “called function” to “function call”.

Delete “add 7.6.1(13)” (still).

Jean-Pierre doesn’t like “not stand-alone”. Tucker doesn’t want to say “anonymous object” in normative wording, because it isn’t well-defined.

9.2(3) needs to be changed because we don’t want to talk about temporary tasks that happen to occur as part of the elaboration.

Steve Baird is concerned about the discriminant living too long; there is no hand-off rule. Tucker explains that the accessibility level is the level of the function in this case:

```
   G (F (A).X.Acc_Disc.all);
```

That seems weird. It seems like that is too long - that’s usually going to be library-level. It seems that the accessibility should be the accessibility level of the object.

```
   return T (Disc => A'Access);
```

After much discussion, we agree that we don’t understand this.

Tucker will take another crack at this.

Detailed Review of Regular AIs

AI-51-01/15 Size and Alignment clauses

WG9 selected the other alternative.

No action: 8-0-1.

AI-51-02/01 Size and Alignment clauses

No one has any changes for this AI.

Approve AI: 7-0-3.

AI-158/05 Renamings of primitives of a class-wide generic actual type

Tucker tries to explain the wording. The problem with calling the operation of the root type in the tag-indeterminate case is that the function in question might be abstract. We certainly don’t want to call that!

Delete the last comment from the example in the AARM note.

The wording of the new paragraphs after 12.5.1(23) is still not right. Try “the actual type is considered to have primitive operations …”. How about “Within the instance, the actual type …”? Neither of these is liked much. It is suggested that Randy will figure it out, but he is skeptical.

Drop “renamings of…” from the subject.

There is a discussion whether Randy can implement this. He replies that certainly most of it can be implemented without trouble, there are a few cases that would be difficult. Some detailed implementation discussion ensues, which suggests that the problem cases might be easier than he thinks (since he has to generate two branches for each call anyway, the class-wide branch for a tag-indeterminate call can just raise Program_Error).

There are more concerns about the wording: Change the first part of the second paragraph to: “If the corresponding operation of T is a function that has no…” (so the last sentence has a function to talk about.)

We decide to rearrange the paragraphs for 12.5.1(23) some more, and write the result:

In the case where a formal type is tagged with unknown discriminants, and the actual type is a class-wide type T'Class:

- Each of the primitive operations of the actual type is considered to be a subprogram (with an intrinsic calling convention -- see 6.3.1) whose body consists of a dispatching call upon the corresponding operation of T, with its formal parameters as the actual parameters. If it is a function, the result of the dispatching call is returned.

- If the corresponding operation of T has no controlling formal parameters, then the controlling tag value is determined by the context of the call, according to the rules for tag-indeterminate calls (see 3.9.2 and 5.2). In the case where the tag would be statically determined to be that of the formal type, the call raises Program_Error. If such a function is renamed, any call on the renaming raises Program_Error.

Delete the TBD in the AARM Note of the wording.

Randy still needs to reword this as previously discussed.

Approve AI with changes: 7-0-2.

AI-291-01/06 By-reference types and the recommended level of support

WG9 selected the other alternative.

No action: 8-0-1.

AI-291-02/02 By-reference types and the recommended level of support

We need look at 13.1(7); Tucker added that at the last minute without review. We look at Tucker's wording and Randy's rewrite. Tucker's wording seems to imply that operations are defined by components, which isn't true for assignment. After much discussion (more heat than light), we decide to drop part of the last sentence in Tucker's version:

"For a composite object[, composite operations are generally defined in terms of operations on components, so] padding bits might not be read or updated in any given composite operation, depending on the implementation."

Change "an objects" to "an object" in the first sentence.

Approve AI with changes: 4-0-6.