Minutes of the 25th ARG Meeting

19-21 November 2004

Atlanta, Georgia, USA

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

Observers: Matthew Heaney (Saturday midday only).

Meeting Summary

The meeting convened on 19 November 2004 at 09:00 hours and adjourned at 14:00 hours on 21 November 2004. The meeting was held in the frigid room at the Doubletree Buckhead, in Atlanta, Georgia, USA. The meeting was hosted by SIGAda in association with their annual conference. The meeting covered the entire agenda.

AI Summary

The following AIs were approved:

AI-226/02 Cyclic Elaboration Dependences (7-0-0)
AI-312/01 Environment-level visibility rules and generic children (6-0-1)
AI-343/01 C_Pass_By_Copy_Convention is required (8-0-2)
AI-386/01 Further functions returning Time_Span values (9-0-1)
AI-388/01 Add Greek pi to Ada.Numerics (8-1-1)

The following AIs were approved with editorial changes:

AI-162/05 Anonymous access types and tasks termination/controlled type finalization (7-0-2)
AI-188-02/01 Setting a task base priority is immediate (8-0-2)
AI-237/06 Finalization of task attributes (8-0-1)
AI-260-02/02 How to control the tag representation in a stream (9-0-1)
AI-302-03/08 Container library (10-0-0)
AI-303/02 Library-level requirement for interrupt handler objects (10-0-0)
AI-309/01 Pragma Inline issues (10-0-0)
AI-311/02 Static matching of formal scalar subtypes (9-0-1)
AI-330/01 Generic actual parameters are always value conversions (9-0-1)
AI-331/01 10.1.1(19) doesn’t handle multiple nesting? (8-0-2)
AI-332/03 Resolution of qualified expressions and object renamings (8-0-2)
AI-337/01 Applicability of C interfacing advice to private types (8-0-2)
AI-357/06 Support for Deadlines and Earliest Deadline First Scheduling (10-0-0)
AI-366/06 More liberal rules for Pure units (10-0-0)
AI-370/04 Environment variables (10-0-0)
AI-373/03 Undefined discriminants caused by loose order of init requirements (9-0-1)
AI-377/02 Naming of generic child packages (6-0-1)
AI-382/02 Current instance rule and anonymous access types (8-0-2)
AI-385/02 Stand-alone objects of anonymous access types (8-0-1)
AI-387/02 Introduction to Amendment (10-0-0)
AI-389/02 Allow aggregates of partial views (5-0-5)
AI-391/01 Functions with controlling results in null extensions (5-0-5)

The corrections to the following approved AIs were approved (as a group - 9-0-0):

AI-161/08 Default-initialized objects
AI-196/04 Assignment and tag-indeterminate calls with controlling results
AI-216/14 Unchecked unions -- variant records with no run-time discriminant

AI-217-06/12 Limited with clauses
AI-218-03/07 Accidental overloading when overriding
AI-229/06 Accessibility rules and generics
AI-230/13 Generalized use of anonymous access types
AI-235/06 Resolving 'Access
AI-251/16 Abstract interfaces to provide multiple inheritance
AI-252/09 Object.Operation notation
AI-254/10 Anonymous access to subprogram types
AI-256/09 Various wording changes to the standard
AI-262/07 Access to private units in the private part
AI-287/09 Limited aggregates allowed
AI-301/09 Operations on language-defined string types
AI-326/08 Incomplete types
AI-351/06 Time Operations

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

AI-158/04 Renamings of primitives of a class-wide generic actual type (6-0-4)

The following AIs were discussed and require rewriting for further discussion or vote:

AI-383/00 Unconstrained arrays and C interfacing

The following AI was discussed and resolution was delegated to WG9:

AI-291/05 By-reference types and the recommended level of support

The following AIs were voted No Action:

AI-149/04 Miscellaneous confirmations (10-0-0)
AI-186/04 Range of root integer (5-0-4)
AI-188-01/07 Setting a task base priority is vague (10-0-0)
AI-359-04/04 Partial view of generic instantiations (9-0-1)
AI-379/01 Static evaluation of numeric attributes (7-0-2)
AI-390/01 Defining by renaming an inherited subprogram (9-0-1)

Detailed Minutes

WG9 Meeting Report

All of the AIs we submitted were approved, except AI-307 and AI-354 which were withdrawn for further editorial review. (They were changed too extensively to present to WG9 without allowing for a review period.)

Meeting Minutes

The minutes of the 24th ARG meeting were approved by acclamation.

Next Meeting

The next meeting (26th) will be held in Paris, France, February 12-14th, 2005. Jean-Pierre Rosen will host. This meeting will concentrate on reviewing the integration of the core features of the language.

We'll need a meeting before the York meeting to review the integration of the remainder of the language. Tucker volunteers to host a meeting. April 15-17th, 2005 are the dates chosen.

We'll meet in York after the WG9 meeting, but perhaps mainly to celebrate the completion of the Amendment.

Thanks

The ARG thanks our host, SIGAda, for the top-notch water and the fine room where we never needed to worry about being too warm. The ARG also thanks Tucker Taft for his increasingly successful selection of restaurants. Finally, the ARG thanks Randy Brukardt for taking the minutes, with help from Steve Michell when Randy's computer malfunctioned on Friday.

Old Action Items

For the second consecutive meeting, all old action items were completed. We didn't review the list of items for this reason.

New Action Items

The combined unfinished old action items (of which there are none) and new action items from the meeting are shown below.

Steve Baird:

- Create an AI to fill any holes in conversions of anonymous access types (see discussion of AI-385).
- AI-383
- Review AARM Section 5, Section 6, and Section 7.

John Barnes

- Review AARM Section 2 and Section 12.

Randy Brukardt:

- Post new consolidated AARM including all core changes by December 13th.

Editorial changes only:

- AI-162
- AI-188-02
- AI-237
- AI-260-02
- AI-302-03
- AI-303
- AI-309
- AI-311
- AI-330
- AI-331
- AI-332
- AI-337
- AI-357
- AI-366
- AI-370
- AI-373
- AI-377
- AI-382
- AI-385
- AI-387
- AI-389
- AI-391

Alan Burns:

- Provide the wording change for AI-188-02 as soon as possible.
- Provide the wording change for AI-357, Dynamic Semantics, paragraph 5 as soon as possible.
- Review AARM Foreword, Introduction, Section 1, and Section 9.

Gary Dismukes:

- Review AARM Section 5, Section 7, and Section 12.

Bob Duff:

- Review AARM Section 4.

Pascal Leroy:

- Check the introductory text of AI-302-03 for changes needed because of the reorganization and addition of Hashed_Sets and Ordered_Maps.
- Review AARM Section 3 and Section 13.
- Provide the full reference to the ISO time standard (8601). Determine if UTC is correct ISO usage.

Steve Michell:

- Review AARM Section 9 and Section 13.
- AI on seconds and minutes from a Time_Span value (see discussion of AI-386).

Erhard Ploedereder:

- Review AARM Section 4 and Section 11.

Jean-Pierre Rosen:

- Review AARM Section 6 and Section 11.

Ed Schonberg:

- Review AARM Section 8 and Section 10.

Tucker Taft:

- Create an alternative to AI-51 for decision by WG 9 (to complement the Tucker Taft version of AI-291) (by December 10th)
- Update AI-158 (by December 10th)
- Create an AI to complement AI-318-2 - handle primitiveness and controllingness of access result types. (See discussion of AI-391)
- Update AI-391 to cover access return types (by December 10th)
- Review AARM Section 3, Section 8, and Section 10.
- 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 Foreword, Introduction, Section 1, and Section 2.

Detailed Review

The minutes for the detailed review of AIs are divided into 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.

Draft AARM Review

Randy tells the group that the AARM and RM distributed on Tuesday includes the introductory sections (including section 1), and clauses 3.9.1 through 11.6. All ARG approved changes (along with John's introduction text) are included in this work.

Randy notes that because of Ada Europe's requirements, he will need to finish another version around December 13th. He expects this version to include the entire core language. He hopes to have the entire language finished in time for the Paris meeting.

Pascal announces that he wants two reviewers (in addition to Randy, of course) for each section (chapter) of the AARM. For this round, we'll just review the core of the version Randy will produce by December 13th. AARM notes and unchanged part of the Standard also need to be reviewed for problems. This review should complete by close of business January 21st, 2005.

After some kind of white elephant gift exchange, reviewers were assigned to each section as follows:

Foreword/Introduction/Section 1: Alan Burns, Tullio Vardanega

Section 2: Tullio Vardanega, John Barnes

Section 3: Pascal Leroy, Tucker Taft

Section 4: Erhard Ploedereder, Bob Duff

Section 5: Steve Baird, Gary Dismukes

Section 6: Jean-Pierre Rosen, Steve Baird

Section 7: Steve Baird, Gary Dismukes

Section 8: Tucker Taft, Ed Schonberg

Section 9: Alan Burns, Steve Michell

Section 10: Tucker Taft, Ed Schonberg

Section 11: Erhard Ploedereder, Jean-Pierre Rosen

Section 12: John Barnes, Gary Dismukes

Section 13: Pascal Leroy, Steve Michell

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

Q) The rule used for paragraph numbers is to use normal numbers for insertions at the end of a clause; also use normal numbers for new AARM notes so long as the number doesn't change the number of any existing AARM notes. All new and changed paragraphs still will have /2, so there cannot be any confusion. Paragraph numbers of things added in the Corrigendum will not be changed, even if they should be by the above rules.

A) The group thinks that this is acceptable.

Q) AI references occur on every revised paragraph. A reference occurs on every added paragraph unless there are a set of clearly related new paragraphs (like a set of bullets or the parts of an example or package). Then the reference only occurs on the first new paragraph. This decreases clutter.

For AARM notes, the same rules apply. In addition, references are omitted from AARM notes that are new and belong to new paragraphs (the note then clearly shares the reference with the paragraph — it's part of the same set).

For items in the Corrigendum, the defect report index is also included (this was the only reference in the RM+Corrigendum), as those are more readable and can be included with (and linked to) the HTML version.

A) The group thinks these are reasonable rules.

Q) At the last meeting, we decided to use regular numbers for the new paragraphs of 10.1.2, even though this changes the paragraph number of the Note, and it also changes the numbers of several AARM notes. There doesn't appear to be any possibility of confusion, as there are no AARM notes with numbers similar to the old ones. Is this change acceptable?

A) Yes.

Q) The Legality Rules portion of 4.6 was reorganized and reformatted. As the tools do not support changing the format of the text, nearly the entire block had to be deleted and reinserted. Is this acceptable?

A) The large block of deleted paragraphs looks silly in the RM. Tucker suggests trying to mark all of the deletions as a single sentence. Something like: 8-20 *These paragraphs were deleted.* It may be preferable to do this by hand.

Change the Amendment document to match the change order in the AARM. Mark AARM 4.6(8.a/2) as inserted text.

Q) Most of the existing clauses have many examples (especially of syntactic features), and hardly any of the AIs add examples for the new features that they include. Should additional examples be added to the Standard?

A) Are these examples useful? Steve Michell says that they are. Tucker agrees, and goes on to say that the examples build on each other. Randy points out that there isn't any reasonable tagged type example, so it's hard to do anything useful there (for interfaces, Object.Operation notation, etc.). Tucker has talked himself into being assigned the action item to improve the examples, and include the new features.

Q) We added access_definition (anonymous access types) as a function result in part because it was inconsistent not to have it. We added null_exclusion to formal parameters (for named access types), but we didn't add it to return types. This seems inconsistent, should this be added to return types?

A) Yes, **not null** should be allowed on function results.

Q) *Foreword to this version of the Ada Reference Manual* was originally added for political reasons; thus it was retained. Is that OK?

A) Yes. We'll discuss the contents as part of the discussion of AI-387.

Q) 3.9.4(9/2) declares a rule suspiciously similar to 3.9.1(3), which was deleted by AI-344. Should this rule be dropped, too?

A) Yes. But we need a rule like 3.9.1(4/2). [Editor's note: This is clearly wrong. Interfaces are tagged types, as are any types that can declare them as an ancestor. The declared type is clearly descended from any interface used, so 3.9.1(4/2) already applies to a type that references a formal interface.]

Q) AI-280 adds a bounded error, while the existing case is a check. This seems inconsistent.

A) Tucker says that using a bounded error was intended to avoid the distributed overhead of a check. This case is uncomfortably close to reopening a previously decided issue; we decide to make no change here.

Corrections to Approved AIs

All corrections were approved as a group. The vote was 9-0-0.

AI-161/08 Default-initialized objects

Randy explains that Calendar is not Preelaborated, so Time doesn't need Preelaborable_Initialization.

AI-196/04 Assignment and tag-indeterminate calls with controlling results

Randy says that he corrected “assignment statement” to “assignment_statement”.

AI-216/14 Unchecked unions — variant records with no run-time discriminant

Adam Beneschan noticed that formal derived types never have a known_discriminant_part; the wording should say known discriminants. He also noted that we describe this in terms of suppressing checks on a type; this is not well-defined, indeed we moved it to Annex J. How do we fix this? It’s too hard to specify, we should add a To Be Honest AARM note saying that the meaning isn’t well-defined. Randy and Steve B. will write this.

AI-217-06/12 Limited with clauses

Pascal asked that 8.3(20) be changed to exclude visibility on the limited view within the declarative region of the view.

Add a “within” after “nor” in 8.3(20). This is a double negative. Can it be improved? Tucker will try to improve these two sentences. Later during the meeting he provides:

> The declaration of a library unit (including a library_unit_renaming_declaration) is hidden from all visibility at places outside its declarative region that are not within the scope of a nonlimited_with_clause that mentions it. The limited view of a library package is hidden from all visibility at places that are not within the scope of a limited_with_clause that mentions it; in addition, the limited view is hidden from all visibility within the declarative region of the package, as well as within the scope of any nonlimited_with_clause that mentions it.

AI-218-03/07 Accidental overloading when overriding

Randy notes that there needs to be a cross-reference from 6.1 to 8.3 (where the rules are) for overriding_indicator; otherwise someone looking in the syntax index couldn’t easily find the semantics and legality rules.

Tucker notes that the new text says overriding_clause rather than overriding_indicator. Also in AARM notes 6.1(42.f/2) and 6.3(11.f/2).

Several people think this isn’t important enough to mention right up front. Abstract is a similar case, and it is covered in 6.1(30), this should follow it with the text: “An overriding_indicator is used to indicate whether overriding is intended (see 8.3).”

AI-229/06 Accessibility rules and generics

Adam Beneschan noted that this rule needs to apply to the unit itself (if it is a generic subprogram) as well as to the contents. Added the phrase “is a generic unit or” to accomplish this.

Erhard notes that this wording needs “unit” after “declarative unit of the generic”.

AI-230/13 Generalized use of anonymous access types

Randy noticed two Notes (4.6(61) and 3.10.2(34)) that needed correction, and also some words that were inadvertently dropped from the wording of 3.10.2(12).

Tucker says to delete “all instances are constant” from 3.10.2(34.a), because that’s not true. ":=" is defined for anonymous access types, and might not be useless. So just add ":=" to the previous list, and delete the rest of the text.

AI-235/06 Resolving 'Access

Randy added italics to the type names in the new wording, to be consistent with the usage in the rest of the section. He also reworded the note of 4.1.4(14/1); the need to do this was noted in the e-mail discussion on the AI, but it was not done.

The Note should say “expected type for the attribute_reference”, not prefix (this is an existing bug in the Standard). Fix the AARM note to say “expected type” rather than “expected profile”.

AI-251/16 Abstract interfaces to provide multiple inheritance

Randy noted that the lead-in to the bulleted list failed to end with a colon. He also noted that the wording of which rules the generic legality rules apply to was very vague; the wording was improved to make it clear which rules were intended.

AI-252/09 Object.Operation notation

Randy added a colon to the new bullet to be consistent with the existing text. He also added an example of an Object.Operation name.

Erhard would like parameters on the example. That would be wrong; these are examples of names, not of calls. None of the other subprograms here have parameters either, for the same reason. The fact that this is a normal name is exactly what we’re trying to illustrate here. Erhard seems unconvinced. Tucker will look at this example when he looks at examples generally; he may have a better example.

AI-254/10 Anonymous access to subprogram types

Erhard asked that the replacement wording of 3.10.2(32) be changed to add “unit” after “generic”. Randy added a correction to the Note of 3.10.2(37), which used to say that downward closures aren’t allowed.

John Barnes says that Erhard’s change wasn’t made in the !corrigendum section, only in the !wording.

Possibly reorder the Note. The reviewers (Pascal, Tucker) will suggest an improvement if they feel it is necessary.

AI-256/09 Various wording changes to the standard

Randy noted that *assignment statement* is not defined; he changed it to assignment_statement. This is not only in the new wording — 7.6.1(16) — but also in 7.6.1(12).

AI-262/07 Access to private units in the private part

Randy changed "keyword" (which is not defined) to "reserved word".

AI-287/09 Limited aggregates allowed

Randy added an example of using <> to the record aggregate case.

AI-301/09 Operations on language-defined string types

Bjorn Persson noted that the Drop parameters on Bounded_Slice, Unbounded_Slice, and Set_Unbounded_String do not appear to have any use. They were removed; they appear to have been a cut-and-paste error.

AI-326/08 Incomplete types

Randy added italics to the type names in the new wording, to be consistent with the usage in the rest of the section.

AI-351/06 Time Operations

Tucker asked that one of the notes be rewritten.

Make sure that UTC is an ISO term. Add a reference to the ISO time standard into 1.2. Pascal will find the proper reference to the standard; and will look into whether UTC is correct usage.

After the meeting, Pascal provided the following reference:

ISO 8601:2004 — Data elements and interchange formats — Information interchange — Representation of dates and times.

ISO 8601 references UTC, as does the ISO web site. UTC is maintained for the benefit of all mankind by the Bureau International des Poids et Mesures (BIPM. ISO TC 12 is the correspondent of BIPM within ISO.

UTC means Coordinated Universal Time.

Detailed Review of Amendment AIs

AI-260-02/02 How to control the tag representation in a stream

Change the subject to “Abstract formal subprograms and dispatching constructors”

Correct typo in third paragraph of the summary: “A operation”should be “An operation”.

In the problem, change “But the tag will always be streamed as an unbounded string object” to “But the tag will always be streamed as an object of type String”, as this is not talking about Unbounded_Strings.

Overriding is spelled with two 'd's in several places (but careful: overridden has two 'd's).

We should specify than when you declare a variable of type Tag without giving it an initial value, it gets a value of No_Tag.

Why are two versions of Dispatching_Constructor needed? Non-limited tagged types do not match limited tagged private types, and we need to provide a limited version since we have added limited function returns.

In the wording added after 3.9(18), in the paragraph that begins "Generic_Dispatching_Constructor provides...", remove the sentence “This capability is known as a factory.”. That should be made into a note.

Change the next sentence to “The function Constructor is expected to create the object given a reference to an object of type Parameters.”

In the wording added after 3.9(25), in the paragraph that begins “An instance of Generic_Dispatching_Constructor...”, add a sentence that any exception raised by a call to Constructor is propagated to the caller.

Change the second sentence of that paragraph to “Otherwise, it dispatches to the primitive denoted by the formal Constructor for the type identified by the tag The_Tag passing Params, and returns the result.”.

The AARM Note needs a reference to wherever AI 279 is placed (it will be 13.13.2) under “reason”.

Confirm that after 3.9(25) is the proper place for Erroneous Execution. [Editor’s note: It is.]

The change in 3.9.2(2) defines "call on a dispatching operation"; the change is needed because an abstract formal subprogram is not a primitive, but we certainly want it to be dispatching. The wording should read: "A call on a dispatching operation is a call whose name or prefix denotes the declaration of a dispatching operation".

For the change after 3.9.3(3), there is general discomfort about the hair-splitting of the rules. Tucker thinks that there is no real problem; a formal subprogram is never primitive. So just change the first sentence of 3.9.3(3) to "A subprogram declared by an abstract_subprogram_declaration (see 6.1) or declared by a formal_abstract_subprogram_declaration (see 12.6) is an abstract subprogram."

The change for 3.9.3(11) appears to be a double negative. Try "A generic actual subprogram may only be abstract if the generic formal subprogram is a formal_abstract_subprogram_declaration." Ugh, a Legality Rule with "may" is bad news. Leave the wording as originally proposed.

Tucker wants to permit "**is abstract** <>" or "**is abstract** T" in formal_abstract_subprogram_declaration. Randy and Pascal complain that the syntax is ugly. Tucker says it isn't that bad, and we don't want to force users to choose between defaults or primitive operations. The lack of defaults would be weird. We agree to add the defaults to the syntax (no other changes should be needed).

12.6(8) defines the term *dispatching type*. Do we really need this term? Randy says yes, because the rules for anonymous access parameters and returns take a lot of wording, and we don't want to repeat that. After more discussion, we conclude that the problem is more the term itself than the need to define it. We decide to change the term to *controlling type*. In the text added after 12.6(8), the legality rule for instantiations should read: "The actual subprogram for a formal_abstract_subprogram_declaration shall be a dispatching operation of the controlling type or of the actual type corresponding to the controlling type".

Change the beginning of the text added after 12.6(8) to read: "If a formal parameter of a formal_abstract_subprogram_declaration is of a specific tagged type T or of an anonymous access type designating...". Change similarly the wording that talks about function results.

In the rule after the AARM note, change "primitive" to "dispatching", as we need to allow the actual to be an abstract formal subprogram (if the instantiation is nested in a generic).

We need to change 3.9.2(1) to include the notion that a dispatching operation includes abstract formal subprograms.

The references to 12.6(18) should be to 12.6(16).

Approve AI with changes: 9-0-1.

AI-302-03/08 Container library

Pascal explains the changes that he made to the packages. He separated the sets and maps into introductory sections defining general semantics and specific implementations with differences from the general semantics.

Jean-Pierre wonders whether the AI documents that AI-254 is required. If not, it should be clear.

John notes a typo in !problem: "If is..." => "It is..."

Tucker suggests replacing the upper bound expression of Vectors.Extended_Index with:

```
    Index_Type'Min (Index_Type'Base'Last-1,Index_Type'Last)+1;
```

Randy and others prefer the 'Pos formulation because it is the idiom for conditional expressions in Ada, and it is good to show that to users. Discussion is inconclusive, so we take a straw poll: Pos: 2, Min: 4, Abstain: 4. Switch to 'Min as outlined above.

We discuss the Replace operation for Sets.Generic_Keys. Tucker and Pascal are in favor of keeping it. Tucker would like Insert/Include, but that is bad because the key has to be ignored. Pascal notes that the description of Replace for Generic_Keys is wrong; it should reference Replace_Element, not Replace.

Pascal thinks that Replace_Element for sets replaces the whole thing, and thus should be called Replace. Randy points out that the name of this operation in all of the other containers is Replace_Element. Pascal groans and agrees to leave the name unchanged.

Pascal will check out the introductory text for changes needed because of his reorganization.

Randy explains the wording for Update_Element. He says that he'd rather have a check, he didn't include it because it takes one bit per element. Pascal says that he would prefer to eliminate erroneousness, and one bit per element is not a significant cost. Tucker suggests that modifying the container here is not the intended use, so we should just prevent all deletions from the container (of the element being modified, or of any other). Randy suggests that we could use the same check as Iterators (which also would prevent inserts). That's a bit weird (but good for vectors, where an insert could move the element), but not a significant problem.

Does anyone want to support the notion of not having a check for Update_Element (presumably because of too much overhead)? Tucker says he is concerned about the one bit per element, but otherwise it is not an issue for him. No one else has a concern. The container level checking is preferred, so we use the same check for all three.

We talk a bit about Matt's A(3) concern for calling things concurrently. Jean-Pierre suggests that a user could extend a container and add a lock. Pascal notes that implementations may want to cache items or make other changes on a read; we shouldn't prevent such implementations. Steve Michell suggests adding Implementation Advice to suggest concurrent reads be allowed. Pascal and Tuck object, as that is saying that we don't *really* want to allow implementations to make modifications on reading — and we do want to allow that. We decide to leave A(3) as is, and it will apply to the containers packages.

Randy asks whether the parameters to Query_Element, Update_Element, and Iterate should be **in out**. Matt had been concerned about the under-the-covers modifications needed to make the required concurrency checks (the locking bit has to be set). Pascal notes that the logical properties of the container are not modified; what happens to the physical properties is irrelevant. We decide to leave these as **in** parameters.

Randy notes that he made all of the versions of Delete for cursors consistent. However, Delete for cursors doesn't raise an exception; this is inconsistent with the key version of Delete (the version that doesn't raise an exception is called Exclude). Pascal thinks that this is a bug. Delete should raise an exception. (Randy mentions that Matt wanted to match Unchecked_Deallocation, and also that the post condition is satisfied). Tucker comments that exceptions are raised to catch bugs, and this is going to be a bug 90% of the time. We decide that Delete for cursors should raise Constraint_Error.

Steve Baird worries that there is a problem with the AARM note suggesting that "unspecified" is not "erroneous". We have a discussion of what "unspecified" means. Tucker reads the Section 1 definition, and then concludes that we've been abusing it throughout the standard. (It's really similar to implementation-defined, with the choices enumerated, but with no documentation requirement.) We decide to not try to fix this here in the containers; Randy had tried to limit the possibilities previously, without success.

The wording of the Iterate check is just fine, since it isn't specifying exactly where the check is made. We want that behavior, so that the check can be made in the body of Iterate or on the calls to Insert/Delete.

Matt arrives at this point during the discussion.

Matt asks about "=" for Sets. Pascal had defined "=" in terms of the equivalence function. If "=" is defined in terms of element equality, then we should have an Equivalent_Sets function. Tucker had suggested that, and we agree to add an Equivalent_Sets function.

Hashed_Sets is missing a formal "=" operator; it should default to "is <>". The "=" operator should come last so it can be skipped in instantiations.

Matt asks whether the Equivalent_Elements in Hashed_Sets should be called Equivalent_Keys. There is no particular support for this idea.

Matt now argues that Checked_Update_Element has a different profile than Update_Element, which is enough to avoid confusion between the versions. He would like the name changed to Update_Element. Tucker worries that the "Checked" suggests that the others aren't checked (and we just decided that they are). Tucker suggests

Key_Preserving_Update_Element. That leads into a discussion of whether when the key is changed the delete is followed by Program_Error, or if reinsertion happens. Matt points out that if you reinsert you can change the key without a copy. However, people are uncomfortable with the notion that a subprogram that is supposed to not change the key can actually change it silently. Plus, whether you get Program_Error or not depends on the state of the container, and this nondeterminism can lead to hard-to-debug programs.

Change the semantics of Checked_Update_Element to delete and raise Program_Error if the key is changed; do not try to reinsert.

We try to find a name for this operation:

Checked_Update_Element
Key_Preserving_Update_Element
Update_Element_Preserving_Key
Stable_Update_Element
Update_Element_Part
Update_Non_Key_Part
Update_Element_Not_Key

Update_Element_Preserving_Key is chosen.

Approve AI with changes: 10-0-0.

AI-357/06 Support for Deadlines and Earliest Deadline First Scheduling

Alan says that there isn't any change in semantics in the AI, just improved wording.

Pascal wonders why the priority in Dynamic Semantics is named “R”, it should be “P”.

Parameter TS should be called “Deadline_Offset”.

Package name should be “EDF”; this is not the Department of Redundancy Department.

Dynamic Semantics, paragraph 5. The “as soon as practical” part relates to AI-188-2, and it should be consistent with that.

In Dynamic Semantics, paragraph 10, it should say “added to the ready queue”, not “queues”.

Tucker suggests that the bullets starting “a change to the deadline of T” and “a decrease” should say “occurs” rather than “takes effect” (because there is no delay in between).

Steve Michell notes that there are two occurrences of “EDF_within_Priorities”, which should be “EDF_Across_Priorities”.

Gary asks that the hyphen be removed from “time-line” to make it “time line”.

Approve AI with changes: 10-0-0.

Alan will provide the wording change for the fifth paragraph of Dynamic Semantics to the editor ASAP. This should be similar to the wording for AI-188-02.

AI-359-04/03 Partial generic instantiations

Fix spelling of instantiation in title and first paragraph of wording. Syntax isn't changed to “partial”. Neither is first paragraph of wording. Generally, read “partial” whenever you see “abstract”, and “full” whenever you see “concrete” in this version of the AI..

Tucker explains that he changed the syntax and terminology as determined by the e-mail discussion. He also dropped the part about <> parameters (they seemed to add only confusion).

Should the matching rules be full conformance? No, we don’t want any evaluations to take place in the partial instantiation; these rules prevent that.

Erhard wonders about the first sentence of the static semantics wording. This comes from 12.7(10). But change “includes” to “is”, because there is no second part. The rest is junk that Tucker forgot to delete.

Tucker thinks perhaps that sentence is unnecessary and should be dropped. So now the whole paragraph is gone.

Tucker has changed the freezing rules to require freezing at the end of an instance and then have piecemeal instantiation. This is necessary so a type can contain a component of its own type. Pascal objects that this is a change that we had previously agreed to not make. Randy agrees; he notes that it is privacy breaking. Pascal says that piecemeal freezing is a dead boy issue. He continues that the important issue is to solve the export of containers from specs of private types. Tucker notes also that the signature package problem is also covered even without piecemeal freezing. Randy notes that we could still allow this in 2015 if it actually proves to be important.

Tucker gives in on this issue. Drop the AARM note and the “no later than at the end of the instantiation” from 13.14(5). No new text is needed, but an AARM note to the effect that it happens at the start is needed.

Erhard writes an example:

```
generic
   ...
package G ...
   type T is ...
end G;

with G;
package P is
   package X is new G (...) with private;

   package Y is new G (X.T); -- Illegal, X.T is not fully defined.
   package Y is new G (X.T) with private; -- OK
   package Z is new G (X.T) with private; -- OK
   ...
end P;
```

The full instantiations of Y and Z have to follow that of X.

Steve Baird is worried that a static expression in a generic is freezing. Tucker says that doesn’t happen. Steve writes an example of what he is worried about:

```
generic
package G is
   subtype S is Integer;
   X : constant := S'Size; -- Static expression, frozen here.
end G;

package I is new G with private;
```

Pascal says we don’t care; they aren’t frozen in this instance or any instance.

But a static expression can affect legality:

```
procedure P (Y : Integer := I.X); -- There is a legality check here.
```

So we do care. We need to say that no entity declared by a partial view is static until the full view.

“The view of the entity is never a static view.” Randy objects, because the client view of the partial instance would then be non-static. Tucker revises, “The view of the entity is static only if it is static in the template.”

13.14(18) “An entity with a non-static view shall not be frozen.”

Pascal and Steve Baird are worried about overriding. They give an example:

```
package P is
   type T ....
   function F return T;
   type D is new T; -- F is inherited [F2]

   generic
   package G is
      type ND is new D; -- F is inherited [F3]
   end G;

   package I is new G with private;

   function F return D; -- overriding [F4]

   procedure P (X : I.ND := I.F); -- inherited F [F3].

   package I is new G;
end P;

-- in body:

   procedure P (X : I.ND := I.F); -- I.F is inherited from [F4].
```

The conformance check should fail because I.F has changed.

This is extra implementation work; but it doesn't seem to be a real problem. Tucker thinks that this happens in the existing language, we don't need this feature to cause it.

Do we want this feature? Tucker says that is natural to want to use generics with private types.

Pascal thinks that this is a big language baggage to solve this problem. Randy agrees with Tucker that users will want to export sets of private types. Jean-Pierre notes that you could use a child package if it doesn't self-reference. That is complex if the parent is generic.

Jean-Pierre thinks that it would somewhat hard to explain to a user. He also worries that there may be more worms.

Pascal wonders if we restricted the partial view to the types would that simplify it?. John says that users will just say, "oh, I have to write that in two parts"; it will not be a big deal to understand.

Erhard would prefer a different term to explain these. Partial instance seems like a part of an instance. It's more of a partial view of an instance. That seems like a better term.

Approve intent of AI: 5-2-3. Tucker should update the AI for approval tomorrow.

AI-359-04/04 Partial view of generic instantiations

Pascal wonders why we are defining template in this wording, because it doesn't seem to be a new definition. It doesn't seem to be used much. Drop the definition, and replace the next use by: "of the same generic package".

Change "statically identify" to "statically denote".

Jean-Pierre asks why this is limited to packages. Because you can use renaming-as-body to handle this for subprograms; we don't need a new feature. Additionally, you can't export a type from a generic subprogram, so there is no need for circular references.

Jean-Pierre worries about the restrictions on the generic actuals for the partial instance. An arbitrary expression isn't allowed. That can be handled by declaring a constant before the partial instantiation and passing the constant. Why can’t the compiler do that? It would bring up the question of whether the expression is re-evaluated, and would require full conformance. Moreover, expressions are rare as actual parameters to instantiations.

The static semantics paragraph is confusing. Write everything in singular. We detail the changes, resulting in the wording:

“The declarations within a partial package instance provide views of the entities declared by the corresponding declaration within the full instance. The view is static only if the entity was static in the generic package. Prior to this corresponding declaration, certain restrictions apply to the use of the view (see 13.14). After the corresponding declaration, the view is the same as those defined by the corresponding declaration. Prior to the full definition of a type in the full instance, the type declared within a partial package instance is not completely defined.”

What is the view outside of the package? Are additional operations available? Tucker wonders if the wording can be simplified, which might reduce the confusion.

We look at an example to understand the question:

```
generic
   type P is private;
package G is
   type D is new P;
end G;

package PP is
   type Priv is private;
   package Inst is new G(Priv) with private;
private
   type Priv is access Inst.D;
   package Inst is new G(Priv);
   Ptr : Inst.D;
   -- Ptr.all legal here.
end PP;

O : PP.Inst.D;
```

Is O.**all** legal? That would break privacy (the operations allowed would depend on the full definition). We certainly don’t want that. So no additional operations get added. The external view isn’t the “same”, it’s just completely defined.

What happens to the declarations inside the unit? They’re not handled by the existing completion rules (as is the instantiation itself).

Inside the scope of the full instantiation, the view is the “same”. Outside the scope of the full instantiation, the view is “complete”. Do views change already in the language? Yes, the properties of a view can change (such as adding .**all** in a body) — this is not new.

Jean-Pierre suggests that one of the Legality Rules needs immediately added: “and shall appear {immediately} in the private part”. This is not needed, that is covered by the Legality Rules in 3.11.1(2-3) — that is, by the rules for completion.

Tucker provides wording to handle the issue with the views:

“The declarations within a partial package instance provide views of the entities declared by the corresponding declaration within the full instance. The view is static only if the entity was static in the generic package. Prior to this corresponding declaration, certain restrictions apply to the use of the view (see 13.14). After the corresponding declaration, these restrictions on the view no longer apply. Within the scope of this corresponding declaration, the view from the partial package instance is the same as those defined by the corresponding declaration. Prior to the full definition of a type in the full instance, the type declared within a partial package instance is not completely defined.”

The “after the corresponding” sentence is redundant in the AARM sense (it follows from the 13.14 rules).

Erhard worries that you now can do things that you can’t do directly. For instance, you can derive from a type that isn’t completed; this is thought to be very bad. Pascal worries that we’re introducing various things through the back door. There is general consternation from the group.

Randy thinks that this proposal isn’t mature enough to put in at this point. Jean-Pierre agrees. Steve Michell says that an alternative would be to delay the Amendment to get this in. There is not much support for that.

Tucker asks if we could finish this in Paris. Pascal does not think so; we aren’t clear enough on the solution.

We again argue about whether this AI should be voted no action. There is concern about losing good ideas that we just didn’t have time to finish. We decided last time that John would mention these in the Rationale; that resolution certainly applies to this idea as well.

No Action: 9-0-1.

AI-366/06 More liberal rules for Pure units

10.2.1(16) “0” should be “zero (0)”. Drop “immutably”. Tucker would like to straighten that paragraph out. Pascal thinks that a bulleted list would help. We finally decide not to change it.

Approve AI with changes: 10-0-0.

AI-370/04 Environment variables

Pascal says that he made a number of clean-ups to the AI, but no major changes.

Why does Set raise Constraint_Error? If the environment rejects it because it is too long, has bad syntax, etc. Tucker would prefer the sentence started with “If”. He’s asked to suggest wording.

Meanwhile, Jean-Pierre wonders why concurrent access is erroneous. That seems too strong. On a very simple system, such access could cause real trouble (remember MS-DOS??). Jean-Pierre agrees with the wording of the AI regarding concurrent access.

Steve Michell does not like the wording for Exists as it has two ifs and one otherwise. Pascal suggests “If the external execution environment supports environment variables and an environment variable with the given name currently exists, then Exists returns True; otherwise it returns False.”

Tucker reads his wording for Set paragraph 2.

“If the external execution environment does not allow the definition of an environment variable with the given name and value, Constraint_Error is propagated. The rules associated with such a check are implementation defined.”

The last phrase is widely disliked. We try a number of alternatives:

“The allowed definitions are implementation defined.”

“The constraints on the choices of names and values are implementation defined.”

“The constraints on the allowed definitions are implemented defined.”

“The set of allowed definitions are implemented defined.”

Erhard suggests:

“If implementation-defined circumstances prohibit the definition of an environment variable with the given name and value, then Constraint_Error is propagated.”

We decide to use Erhard's wording.

Pascal notes that he didn't do one thing that was suggested; the Implementation Advice doesn't seem like an Implementation Permission. The group agrees by silence.

Tucker thinks the package lead-in text is weird; Randy and Pascal note that it is the same as Standard and Text_IO. But it shouldn't include "Ada.".

In the Implementation Permission, change "at the beginning of" to "prior to". Should nonempty be there? Yes, otherwise you could meet the letter of the rule by creating an empty environment (which would be trivial and useless).

The example needs to be fixed to eliminate the generic instantiation for Iterate (it's not generic anymore). "ADA" should be "Ada", and "05" should be "2005".

Approve AI with changes: 10-0-0.

AI-382/02 Current instance rule and anonymous access types

Pascal has added to 8.6(17)

"This rule does not apply if the usage name appears within the subtype_mark of an access_definition for an access-to-object type, or of a parameter of an access-to-subprogram type."

That seems weird; why are result types not included? Change it to:

"This rule does not apply if the usage name appears within the subtype_mark of an access_definition for an access-to-object type, or within the subtype_mark of a parameter or result of an access-to-subprogram type."

Change the example callback2's answer illegal to legal. Also change the proposal and discussion to include this case.

Randy worries about Dan Eiler's comment about this not being allowed in stand-alone access-to-subprogram declarations. Randy thinks that he has a point if this is allowed here (and it seems useless).Tucker says that in the infrequent case where you need to do that, you can always wrap the type in a record (simply using an incomplete type wouldn't help, because its name would be hidden from all visibility too). We decide on no change.

Approve AI with changes: 8-0-2.

AI-385/02 Stand-alone objects of anonymous access types

John explains that stand-alone objects are the first thing that someone will naturally try with an anonymous access type, and it is incredibly strange that they are allowed everywhere else but not as stand-alone objects.

This leads to a lengthy discussion of how similar constant and renames are. They are not very similar, in part because the constraints are different (the constraint is as declared for a constant, but are inherited from the renamed object for a rename).

We also discuss the possible confusion between **constant access** and **access constant**. Certainly the meaning is clear, the question is one of whether the syntax is too prone to errors. The alternative, of course, is to be inconsistent in what is allowed.

We eventually agree that option 2 (allow constants and variables) is what is needed.

To do that, we need to clean up the AI to remove other options:

- Get rid of parenthetical text (that is, Nanny Ada).
- Get rid of text marked <<...>>, as that is for other options.
- Get rid of remarks about option 1 or 3, except for discussion section.
- Get rid of AARM note after 3.3.1(24)

What happens when you assign these objects? The designated subtypes must statically match, and obey the accessibility rules. This is detected as the implicit conversion has the same rules as for explicit conversion. There seems to be a hole for conversion to array types, as these must have the same constraints. Should this be a Legality Rule or a rule similar to that of 4.6(48)?

The rule needs to cover dynamic semantics of view conversions for arrays of anonymous access types and generics of same. We decide we need both a Legality Rule and a dynamic semantics rule, similar to accessibility checks.

We believe that we already have rules for all conversions except array conversions.

Steve Baird is given an action item to study conversions of anonymous access types, and write up an AI adding rules to fill any holes (especially of access-to-array conversions).

Approve AI with changes (using option 2): 8-1-1.

Steve Michell votes no because he believes that the **access constant** and **constant access** will be confusing and dangerous.

AI-386/01 Further functions returning Time_Span values

There is general agreement that these functions are harmless and useful.

Steve Michell wonders if there shouldn't be an operation or operations for extracting seconds and minutes from Time_Span — something like Split.

Steve Michell is asked to propose something further for consideration in Paris.

Approve AI: 9-0-1. (Gary doesn't care.)

AI-387/02 Introduction to Amendment

We review the AARM Foreword and Introduction sections, which already incorporate this AI. Some changes found there are only in the consolidated RM and AARM, and will not appear in the Amendment (and thus do not appear in the AI).

In the "Foreword to this Version", paragraph 0.add the missing word in "...publish {a} document..."

Put "changes from Technical Corrigendum", instead of. "Technical Corrigendum changes" at several places in 0.6.

Paragraph 0.5 should be marked as a change from the Amendment (green in HTML). Paragraphs 0.6 and 0.7 have the wrong version number.

Paragraph 0.8 "two" should be "three".

Change paragraph 4 of the Foreword; check on ISO wording here.

In the Introduction, the AI references all have the wrong AI number; it should be 387.

Various corrections to the wording in the AI and AARM are suggested, and are listed by paragraph number.

3/2: "{should} read this first" (only in the AARM).

3.1/2: "This give{s} an introduction."

5.2/2 has two periods at the end of line one (only in the AARM).

38.1/2 :"task-safe" should be "concurrent".

42.2/2: delete "especial".

44.2: “revised” should be “amended”. This whole sentence should be revised. John will do that this evening.

47.2: delete the extra “a” before “on a heap” (AARM only).

49.2: delete “-level”.

57.1/2: “Amendment 1 modifies...”.

57.2/2: “significant” rather than “major”.

57.5/2: “subprogram [sub]types”. Delete “to support downward closures”. John will try to come up with a better reason.

57.6/2: “In addition{,} limited ...constants{,} and...”.

57.7/2: “the predefined environment has been extended...” “major” should be “comprehensive”. “execution environment” should be “environment variables” followed by “file and directory management”.

57.8/2: Kill this bullet, it’s not important enough.

57.9/2 should end with a colon.

57.10/2: get rid of the colon and capital T.

57.11/2: Add :1997 to 13813.

57.12/2: “improved” should be “enhanced”; “introductions” should be “improvements”. Drop “Finally, “.

Acknowledgements {for the Ada 95 edition of the Ada Reference Manual}

“Kept the editor on his toes” ought to be changed in the most recent Acknowledgements (it’s a repeat of the previous acknowledgement.

Will wait on approval until we see John’s new wording for 44.2/2.

On Saturday morning, John reads his new text for 44.2/2: “This amended International Standard updates the edition of 1995, which replaced the first edition of 1987...”

Approve AI with changes: 10-0-0.

AI-388/01 Add Greek pi to Ada.Numerics

Randy says that this would require all Ada programming environments to be able to display such characters. Jean-Pierre says that no one will know to use it.

Pascal says that AI-285 will be required anyway. But Randy points out that that only requires the *compiler* to recognize the characters, not all of the tools. Moreover, that can (and probably will) be accomplished with a practically useless hack until/unless there is sufficient customer demand for the expensive changes needed. But a toolset that can’t edit/display the standard libraries is junk, so Randy thinks that AI-388 is making the users and implementers' life more complicated.

Pascal argues that Greek pi would be much more readable in numeric code than the identifier “Pi”. The Greek pi notation is universal, and we do care about the readability of source code. And at any rate, if your toolset does not display Greek pi correctly, you can always use the good ol’ “Pi”. We are being friendly to users of modern computer systems.

Steve Michell comments that upper case PI would also match but that has a different meaning.

Approve AI: 8-1-1. Randy votes against, as this effectively requires implementations to give a much higher priority to AI-285 than it deserves.

AI-389/02 Allow aggregates of partial views

Tucker would like to disallow it in the scope where the type is completed if it is not composite. That is weird, we don't necessarily want the legality to change when we cross the private part.

Jean-Pierre points out that you couldn't use such a default initialization as a default if the full type is scalar, because the body uses wouldn't be legal.

We need to add wording like "Later in the immediate scope, the full type..." etc.

Jean-Pierre suggests simply banning it inside the package. "The type shall not be completely defined." That wouldn't cover children's visible parts.

There is a suggestion to restrict these to partial views with discriminants. Those have to be composite. That carries the day. We take a straw poll in favor of dinner: 10-0-0. We decide it is too late to continue tonight.

On Saturday, we take up where we had left off.

All of the references to 4.3.2 in the AI should be 4.3.1.

4.3.1(8) OK as is (we will make the discriminant rule a legality rule)

4.3.1(9) delete the To Be Honest note, as it describes what it means for an aggregate of an elementary type.

4.3.1(17.1) change second rule to "shall have known discriminants", and update the AARM note.

Tucker suggests allowing any partial view that is tagged. That seems OK, because the full view has to be composite (we don't have elementary tagged types). The model is that the full view has to be composite, without breaking privacy. But we want to prevent class-wide types and unknown discriminants. So we end up with: "shall have known discriminants or be a tagged, definite type".

4.3.1(17.1) first bullet should say "record_component_association_list" instead of "aggregate". (To avoid problems with nesting.)

Pascal argues that this should be allowed for extension aggregates, because you can write partial aggregates based on the components you know. That's the same as extension aggregates, so they should be allowed. Randy complains about implementation issues, which doesn't get much sympathy.

4.3.2(4) should say "type extension" rather than "record extension".

4.3.2(5) "…through one or more extensions." Tucker thinks that we might want to say "ancestor", because in a generic the actual can be the parent itself. We should allow it in generics, you don't want to disallow that.

"The type of the extension_aggregate shall be a descendant of the type of ancestor_part." Add an AARM explaining why we're allowing this (zero extensions).

Also, we need to add "If the type of the aggregate is derived from the type of the ancestor_part through one or more private extensions, then the record_component_association_list shall include **others** => <>." to 4.3.2(5).

We need to disallow using positional notation on components that are not visible. It's not worth doing that; just disallow positional notation completely for partial views.

Added positional notation to the bulleted list 4.3.1(17.1). "The record_component_association_list shall not include a positional component association."

We also need a similar rule in extension aggregates.

Approve AI with changes: 5-0-5.

AI-390/01 Defining by renaming an inherited subprogram

Tucker explains that the problem is that when the names match, you can't name the routine that has the same name as one that you have defined.

Pascal thinks it is a minor wart in the language that isn't worth a new feature.

Jean-Pierre says that the feature looks harmless.

Tucker writes the example now found in the AI (this was provided after the deadline).

```
generic
    with type Element is private;
package Sets is
    type Set is private;
    function Union(Set1, Set2 : Set) return Set;
    function Intersection(Set1, Set2 : Set) return Set;
       ...
end Sets;

with Sets;
package Int_Sets is
    type Int_Set is private;
    function Union(L, R : Int_Set) return Int_Set;
    function Intersection(L, R : Int_Set) return Int_Set;
    ...
private
    package Int_Sets is new Sets(Integer);
    type Int_Set is new Int_Sets.Set;
       -- Union and Intersection inherited from Int_Sets.Set,
       -- and then immediately overridden by the explicitly declared
       -- operations from the visible part

    function Union(L, R : Int_Set) return Int_Set renames <>;
    function Intersection(L, R : Int_Set) return Int_Set renames <>;
      -- Define visible operations by renaming the ones they override
end Int_Sets;
```

Erhard wonders why this can't be automatic, but that seems to be dangerous. If a completion was forgotten, the declaration could get completed by something that just happens to be lying around.

Steve Baird notes that you need to specify what happens if the routine is not overriding.

Randy finds this very confusing; only an ARG member can understand what it does. Would real users even use it if they ran into the problem?

Pascal says that introductory programmer will not write this; they would write the wrapper. They wouldn't know enough to use this.

Erhard says that writing the wrapper is hard.

The syntax <> isn't very satisfactory.

We take a straw poll on whether to continue working on this. Keep it alive: 2-3-5.

There doesn't seem to be enough interest to continue.

No action: 9-0-1.

AI-391/01 Functions with controlling results in null extensions

Tucker explains that the compiler can create proper functions of tagged types for null extensions; there is no need to require the programmer to do it.

Pascal says that there are two uses to null extensions: things that happen to be empty, or places where you derive to put it into another scope or to complete a private. The second part is fairly common (it is similar to untagged derivation).

The implementation would have to create wrappers of some sort. Generally, this would have to copy the object. For a limited function, something slightly weird would have to happen (to change the tag in place), but it would be the same as an extension aggregate (F with null record).

What about return access types (**access** T)? Is that handled at all? No. Randy writes an example:

```
type T is tagged...
function F return T;
function F2 return access T; -- Primitive? (Yes.) Controlling? (Yes.)

type T2 is new T with ...
-- F and F2 inherited. We better have these rules (3.9.3)
-- apply to access return types.

type T3 is new T with null record;
-- F2? That would require a copy, and lifetime changes
-- (new (F2 with null record)).
```

Tucker says that return access types shouldn't be primitive and inherited. Egad. Let's leave them the obvious way, controlling, with nice-and-fancy tag-indeterminate calls.

access T can't be automated — it can't work for a limited type; and you can't drop the memory on the floor and allocate some random chunk of memory. We have to leave **access** T out.

Tucker wonders if there is a problem for untagged types (they aren't covered by the 3.9.3 rules). No, for untagged types the representation has to match, so you can just return the access value, suitably converted.

Tucker takes an action item, to write an AI to complement 318-2 to handle primitiveness and controllingness of access result types.

Approve AI with changes: 5-0-5

Detailed Review of Regular AIs

AI-149/04 Miscellaneous confirmations

This was a place that Bob Duff stuffed questions not worth making AIs about. We now have ACs for that purpose, so it can be voted No Action.

No Action: 10-0-0.

AI-158/04 Renamings of primitives of a class-wide generic actual type

Pascal says that the convention of the wrappers needs to be defined; that would be the same as the corresponding primitive of T.

Gary and Randy ask about tag-indeterminate calls. The wrapper model gets different semantics than hand-expanded generic code for such calls. Steve Baird gives an example of the problem, asking what happens with X : T := F; where T is the formal type and F is a function with the result of the formal type. The wrapper for a function with just a controlling result would just be a static call returning T. That's OK in this example, but for P (Obj, F); (where P is a dispatching call with two controlling operands) F should have the tag of Obj, which is not necessarily T. That means that the wrapper model would raise Constraint_Error for a failed tag check.

Tucker suggests that the wrapper could take the tag as a parameter, which would allow tag-indeterminate calls. But that seems like too much work.

There probably should be a note explaining this. Steve Baird wonders how a tag-indeterminate call (which doesn't dispatch in a wrapper) can be a dispatching call. Randy suggests that the wording should say "call on a dispatching operation", so that the rules for tag-indeterminate are triggered if necessary.

Gary comments that there is no wording saying what the parameters or result are for the wrapper. We should use the wording from renames-as-body for this.

Approve intent of the AI: 6-0-4.

Tucker will update the AI by December 10th. We'll have to vote to approve it in Paris. Randy will add this new version to the AARM as if it is approved.

AI-162/05 Anonymous access types and tasks termination/controlled type finalization

Tucker tried to simplify the wording, but didn't make much of a change.

Pascal suggests No Action on this AI. Randy objects, saying that there still is a hole in the language. Ignoring it doesn't seem like an appropriate resolution. (Besides, we've faced this at least once before, defining the handling of renames of functions returning controlled objects). Tucker makes the point that limited aggregates exist now, and that makes this less of a pathology than in Ada 95.

The change in 6.5(18) is for return-by-reference, so we don't care about it.

What does the first sentence of the last paragraph of 7.6.1(13/1) mean? For a delay statement whose expression creates tasks, we wait for the tasks before delaying. For an entry call, this is wrong (we certainly don't want the parameters to go away). What about family indexes? Expressions in timed entry calls? We should change this to say delay_expression.

Tucker has a better idea: "In the case of a potentially blocking operation which is a master, finalization of an anonymous object occurs before blocking if the last use of the object occurs before blocking. In particular, for a delay_statement, any finalization occurs before delaying the task." The group agrees.

Why does the replacement for 7.6.1(13) talk about accessibility? We want to know what the master is, not the accessibility.

Erhard is worried that a common case won't work (this is really an AI-230 issue):

```
type Glob is tagged
   ...
   X : access Y;
end record;

Bar : Glob;

procedure P (A : access Y) is
begin
   Bar.X := A;
end P;

...  P(new Y); -- This will fail accessibility (inside of P).

procedure R (A : Y) is
begin
```

```
      Bar.X := new Y'(A);
   end R;

   ... R ((others => <>)); -- OK (no problem inside of R)
```

This seems like a real problem, but a change would be wildly incompatible (because we'd have to change the accessibility of access parameters).

Steve Baird tries to remember why he wrote this wording in terms of accessibility. He fails.

7.6.1(13) should be phrased in terms of masters.

Tucker claims that "the master of an object is the master enclosing its creation whose accessibility level (see 3.10.2) is equal to that of the object" is sufficient. No one wants to disagree with him.

Approve AI with changes: 7-0-2.

AI-186/04 Range of root integer

"modulartypes" should be "modular types".

Tucker thinks that there isn't any problem here; Constraint_Error should be raised if a *root_integer* value is larger than Max_Int. Pascal asks if the example in the question is going to raise Constraint_Error. Tucker says yes. Pascal says that makes it hard to write a generic that has to handle arbitrary discrete types. For instance, it is very hard to write Ada.Numerics.Discrete_Random without raising Constraint_Error for some legal type.

Randy claims the problem is modular types with a modulus near Max_Binary_Modulus; but that isn't handled here. Pascal just wants to handle up to Max_Nonbinary_Modulus. Simple expressions will work. Otherwise, it will work or raise an exception. He doesn't want to force going double precision.

Randy and Tucker argue that an implementation that makes Max_Nonbinary_Modulus greater than 2**31-1 (for a 32-bit machine) is already asking for trouble, so this doesn't buy anything significant. Pascal says that they support Max_Nonbinary_Modulus as 2**32-1, and have no problems.

Tucker says that root_integer has a range already, thus Constraint_Error can be raised: 3.5.4(14). Pascal doesn't want half of the solution.

No Action: 5-0-4.

AI-188-01/07 Setting a task base priority is vague

This AI was replaced by a different alternative (see next item).

No Action: 10-0-0.

AI-188-02/01 Setting a task base priority is immediate

Alan explains that this alternative requires immediate setting unless the task is on an entry queue, because there isn't an implementation issue and that the alternative is useless (waiting until it's convenient).

Alan is asked about Bob's question. He says that if the task is running, there is no problem changing its priority.

Bob's other question points out that we don't want this to apply to multi-processors. So the wording that is in D.6(2-3) needs to be used here. "On a single processor, ..."; similarly for the documentation requirement.

Alan will provide the wording change to the editor ASAP.

Approve AI with changes: 8-0-2.

AI-226/02 Cyclic Elaboration Dependences

Approve AI: 7-0-0.

AI-237/06 Finalization of task attributes

In C.7.2(15.1/1), make the existing paragraph singular. “An access to a task attribute is erroneous...with another such access or with a call...” The next text should say “An access to a task attribute...”

Alan comments that the wording of C.7.2(30.1) should be “as soon as practical”. This is “should” wording, so no change is needed.

Approve AI with changes: 8-0-1.

AI-291/05 By-reference types and the recommended level of support

The discussion has a sentence that starts with a semicolon.

Tucker explains the differences between his edition and Steve’s revision.

Pascal is concerned that AI-51 is now approved by WG9; and it says that subtype and object sizes are handled the same.

We take a straw poll. The Recommended Level of Support for a size clause should require adding empty space for record/array types: 5-3-2.

Tucker and Randy debate the issues here.

Erhard says that there aren’t a lot of paragraphs here; it’s not a major problem to change this late in the process.

Pascal is uncomfortable changing this without WG9 opinion.

Randy suggests using the WG9 process to determine a solution. Pascal agrees.

Tucker will fix AI-51 to match his model as explained in AI-291; it will be filed as an alternative. Steve’s version of AI-291 will be the other alternative. December 10th deadline for these versions; it will then be sent to WG 9 for a letter ballot.

AI-303/02 Library-level requirement for interrupt handler objects

Randy explains that the library-level requirement is not needed, because existing restrictions, accessibility checks, permissions, and semantics already cover this. The only change an implementation need make is to change its error message to point to C.3.1(17) instead of C.3.1(8).

Pascal “it’s” should be “its” in the !discussion.

Steve Michell suggests changing the subject to “Removal of library-level requirement for interrupt handler objects”.

Approve AI with changes: 10-0-0.

AI-309/01 Pragma Inline issues

Pascal notes that Ada 83 allowed this usage, and we should add an implementation permission to allow it for Ada 2005. Tucker says that their compiler enforces the Ada 95 rule, and has a flag to turn the check off.

Steve Baird notes a typo in the wording: “denotes {a} subprogram_body”.

Why can't this be required rather than just an implementation permission? The last time this was discussed, we said that would require a change to the definition of program unit pragma. Why don't we just say that this is an extension to program unit pragmas just for Inline? That would be better for portability than an Implementation Permission. This is an Implementation Permission for compatibility with Ada 83; the Ada 95 way would be to put the pragma inside the body. There is no additional capability provided by this permission.

AARM note: declarate_part should be declarative_part. Remove the text after the square bracket.

The wording has too many "which"s. Change wording to:

"An implementation may allow a pragma Inline that has an argument which is a direct_name denoting a subprogram_body of the same declarative_part."

Subject should be "pragma Inline compatibility".

Approve AI with changes: 10-0-0.

AI-311/02 Static matching of formal scalar subtypes

4.9(31) should say "scalar formal subtype".

Perhaps the third bullet in 4.9.1(1) should be split, because it is mixing dynamic and static.

- both are nonstatic and result from the same elaboration of a constraint of a subtype_indication or the same evaluation of a range of a discrete_subtype_definition; or
- both are nonstatic and both come from the same formal_type_declaration.

Fix the discussion to reflect the new wording.

Gary notes that the subject should be "scalar formal", not "formal scalar".

Erhard worries that a static range applied to a subtype of a formal type would not be static. After much discussion, we agree on:

"The constraint of the first subtype of a scalar formal type is neither..."

Approve AI with changes: 9-0-1.

AI-312/01 Environment-level visibility rules and generic children

Approve AI: 6-0-1.

AI-330/01 Generic actual parameters are always value conversions

Gary asks that "non-generic" be changed to "non generic". But "non-generic" is used in the normative wording of the standard already (see 6.3(6)). Tucker suggests "or if it appears in a call as an actual parameter..." We agree to this wording.

Delete the second paragraph of the discussion.

Approve AI with changes: 9-0-1.

AI-331/01 10.1.1(19) doesn't handle multiple nesting?

"the visible part" should be dropped from the first sentence of the wording; these generics can appear in the private part.

The wording is considered confusing. We try rewording the second sentence.

“Similarly, for a generic child package there is a corresponding declaration for each child of that generic child package, nested immediately within each instance of such a corresponding declaration.”

“Similarly, for each instance of generic child package there is a corresponding declaration for each child of that generic child package, nested immediately within the instance.”

We could get rid of the first sentence, it is very similar to the second.

“For each instance of a generic package there is a corresponding declaration for each child unit of that generic package, nested immediately within the instance.”

Tucker says that this isn’t quite right, because technically an instance doesn’t have the generic children. Put that in a To Be Honest AARM note.

Dropping the first paragraph drops additional wording. Tucker says that this is covered by the context, the previous paragraphs (especially 10.1.1(17)) in the standard.

The cheat is that we say “child”, we really mean the “child of the original generic”, they’re not of the current generic. Maybe we need to say that.

Add: “For the purposes of this rule, if the child itself has a child unit, the corresponding declaration has a corresponding child.”

Add: “(original)” before child generic unit in the last sentence.

Remove the note about 12.2, it is OK. Or is it? We’re not going to change it.

Approve AI with changes: 8-0-2.

AI-332/03 Resolution of qualified expressions and object renamings

“purpise” should be “purpose”.

Fix the AARM note for 8.6(27.a), as **null** doesn’t require a single type anymore (because of AI-230).

Include Tucker’s e-mail in the discussion explaining how the wording change works.

Remove “like a pain.” (prohibitive covers it).

Add answers to the questions in the question (first two examples should be answered No).

Approve AI with changes: 8-0-2.

AI-337/01 Applicability of C interfacing advice to private types

This is consistent with the treatment of parameter passing in 6.2.

Change “cooresponds” to “corresponds” in the Corrigendum text.

Approve AI with change: 8-0-2.

AI-343/01 C_Pass_By_Copy_Convention is required

Approve AI: 8-0-2.

AI-373/03 Undefined discriminants caused by loose order of init requirements

The first three bullets should end with semicolons, and lower case starting letters for the last three.

Jean-Pierre asks why we only eliminate *some* problems. It's not possible to completely eliminate problems; all of the components can't go last. But at least the order is defined, so the programmer can work around it and the result is portable.

Approve AI with changes: 9-0-1.

AI-377/02 Naming of generic child packages

Change "Furthermore" to "However" in the wording.

Tucker doesn't understand all of this visibility gobbledygook. The with_clauses should be illegal in this case; then we don't need visibility rules. The rule should be in 8.3(26), which is a similar rule on subunits.

Tucker suggests deleting "for a subunit" from 8.3(26); replacing "the place of the corresponding stub" by "the place of the compilation unit". Add an AARM note explaining that this also applies to implicitly declared generic child units.

Pascal doesn't think this wording works. Randy and Steve Baird think this is risky. Tucker convinces the group that the following wording is OK:

"Similarly, a context_clause for a compilation unit is illegal if it mentions (in a with_clause) some library unit, and there is a homograph of the library unit that is visible at the place of the compilation unit, ..."

Approve AI with changes: 6-0-1.

AI-379/01 Static evaluation of numeric attributes

The wording is fine for non-static computations. There are problems for static expressions. But the fix is hard. Pascal doesn't want to work to find fixes. Tucker thought that they had agreed on the needed fixes; Randy notes that only one was explained in a fashion that he could understand well enough to put in the AI. There are two other attributes that may also need fixes.

No action: 7-0-2.

AI-383/00 Unconstrained arrays and C interfacing

A brief discussion concludes that we do want to look at this issue.

Steve Baird is assigned to write up this AI.