

Minutes of the 37th ARG Meeting

20-22 February 2009

Tallahassee, Florida, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Gary Dismukes, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen (via teleconference, Friday and Saturday morning), Ed Schonberg, Tucker Taft, Bill Thomas (except Sunday and periodically on Saturday).

Observers: Ted Baker (Friday morning and late afternoon), Greg Gicca (except Sunday and periodically on Saturday), Matt Heaney (Saturday only), Steve Mitchell (Friday morning), Luke Wong (Friday morning).

Meeting Summary

The meeting convened on 20 February 2009 at 09:10 hours and adjourned at 13:05 hours on 22 February 2009. The meeting was held in room 151 of the Love building on the campus of Florida State University in Tallahassee, Florida. As expected, the meeting covered only about half of the entire agenda, although it did cover all of the ASIS agenda.

AI Summary

The following AIs were approved with editorial changes:

- AI05-0009-1/07 Confirming rep. clauses and independence (6-0-2)
- AI05-0103-1/03 Return statements should require at least static compatibility (8-0-0)
- AI05-0112-1/01 Names for anonymous aspects of representation (8-0-0)
- AI05-0116-1/02 The value of Alignment for a class-wide object (8-0-1)
- AI05-0128-1/01 "/" is a primitive operation (9-0-0)
- AI05-0129-1/02 A limited view does not contain views of incomplete types (7-0-2)
- AI05-0132-1/01 A library unit pragma must be used in a library unit (9-0-0)
- AI05-0133-1/01 Extending a type with a self-referencing discriminant constraint on a component (8-0-1)
- AI05-0134-1/01 Full conformance should include the profiles of anonymous access-to-subprogram parameters (9-0-0)

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

- AI05-0107-1/01 A failed allocator does not leak memory (7-0-2)

The following AIs were discussed and assigned to an editor:

- AI05-0001-1/02 Bounded Containers and other container issues
- AI05-0123-1/03 Composability of equality
- AI05-0135-1/02 An extended-scope use clause
- AI05-0136-1/02 Multiway tree container
- AI05-0138-1/00 Improving accessibility
- AI05-0139-1/01 User-defined iterators
- AI05-0145-1/02 Pre- and Postconditions
- AI05-0146-1/01 Type and Package Invariants

The following AIs were discussed and voted No Action:

- AI05-0140-1/01 Identity functions (8-0-0)

The following SIs were approved with editorial changes:

SI99-0024-1/13 Provide a semantic model in addition to existing syntactic model (10-0-0)
SI99-0037-1/05 Review standard for problematic wording (6-0-3)
SI99-0040-1/01 Primitive operations given by instantiation (10-0-0)
SI99-0042-1/01 Handling generic children in ASIS (7-0-3)
SI99-0043-1/01 Multiple A_Configuration_Compilation_Units are supported (10-0-0)
SI99-0044-1/01 Parent subtype without a declaration (9-0-1)
SI99-0045-1/02 The meaning of Corresponding_Expression_Type (7-0-2)
SI99-0046-1/01 Clarify the Context parameter in ASIS queries (10-0-0)

Detailed Minutes

Meeting Minutes

John will send some typos to Randy for correction. We'll trust them to correct those properly. Minutes were approved unanimously with changes.

Date and Venue of the Next Meeting

The next meeting will be at Ada Europe, Brest, France, June 12-14, 2009 (WG 9 is Friday morning, June 12). Erhard says rooms in and travel to Brest is going to be a problem, there is another conference with the same dates. There is a room block reserved until April 1. The train takes 4 hours from DeGaulle Airport. Planes are likely to be difficult. Ed needs to ask for a meeting room for the ARG ASAP. The conference is at campus (5 miles from town).

The meeting following that will be associated with the SIGAda conference, November 6-8, 2009, St. Petersburg FL. (WG 9 is Thursday afternoon, November 5th).

ASIS document review

How should we review the ASIS Standard document? We have promised to provide it to WG 9 in time for their next meeting (in Brest).

Ed suggests that we delay giving the document to WG 9 until after we have semantic subsystem examples. It would be bad to standardize something that no one has tried. Randy notes that we are running short on the clock for JTC1 procedures; bad things happen if we are late (and JTC1 has been enforcing these time limits recently). Steve Michell: suggests sending ASIS as a CD [Committee Draft]. That is a document that we expect to get comments on. We don't usually work that way because of copyright issues, but in this case there is no reason not to do so (ISO already has the copyright). Submitting a CD would reset the clock. In particular, we don't need to wait until it is done.

We agree to this approach.

Randy and Greg will finish the draft by March 16th. Randy should do the ASIS part of the minutes of first. We should plan on giving Joyce the final CD by May 12th. So ARG comments need to be finished by May 1st. We need to tell Joyce about the impending CD.

The ASIS Editors should assign pieces to review to the ARG members, with comments due by May 1st. They should try to avoid giving people sections that they wrote (the semantic interface should not be reviewed by Tucker, Randy, Steve, or Bill, for instance).

Scope of Amendment 2

Randy describes the situation: he proposed a schedule in November, which mainly drew comments about "requirements analysis". Joyce asked the ARG to provide a proposed scope and plan as soon as possible.

Randy goes on to suggest some goals for this Amendment that he feels cover most of the areas that have been informally discussed:

- Improve use and functionality of the predefined containers;

- Improve the ability to write and enforce contracts for Ada entities (for instance, via preconditions, user-defined constraints);
- Improve the capabilities of Ada on multicore and multithreaded architectures.

Tucker suggests adding two more goals:

- Improving use of access types;
- Language correctness.

Ed wants to show progress to the Ada user community, but doesn't want the language definition to get too far ahead of implementations. (After all, there is only one implementation of Ada 2005 currently available.) But it is important to show changes with "visibility" to the user community. He says that a language revision should be more than fixes and additions to existing libraries, to indicate that language development is ongoing and that the ARG is receptive to change. Otherwise the community will sense that nothing much is happening.

Tucker wonders how much new syntax is allowed. A modest amount is the answer; new syntax is more "visible", but it also causes backwards compatibility issues.

IRTAW should be allowed to look at multicore stuff, but it seems like it would too large for this modest Amendment. Moreover, IRTAW is unlikely to make fully formed proposals at their next meeting, and they only meet every 18 months (which is way too long for our schedule based on the WG 9 request). Perhaps that would be in the next Amendment? Steve Michell asks about the long-term plan, will there be another Amendment following this one? It is way too early to talk about that – we cannot guess the state of the computer business in general and the Ada business specifically in 2017.

Tucker wonders how much IRTAW is tied into the user community (as opposed to academics). Steve Michell says several members are primarily Ada users (including a lot of detail not recorded here).

Jean-Pierre says that many customers hardly use Ada 95, so adding too many fancy features doesn't mean much to those customers. But further examination shows that they do use child packages, access parameters, and the like. And Ed notes that AdaCore has serious customers that use some of the new Ada 2005 features extensively.

We should start with the "Instructions to the ARG" that we followed last time. The last Amendment was about as successful as we could have hoped for, so we would like to repeat that process as much as possible. Randy will draft a new version using the goals previously discussed; we'll try to circulate that and discuss it further during the meeting.

Is the schedule Randy had proposed OK? Randy reads it to the group.

Approve schedule: 10-0-0.

On Saturday morning, we read the draft of the *Instructions to the ARG* that Randy distributed. We make several minor changes to the text. The only significant one was to add "minimally" to the compatibility text.

Randy will correct the formatting of the bullets. The reformatted version can be found at http://www.ada-auth.org/ai-files/grab_bag/Inst2ARG-d3.pdf. This version will be forwarded to Joyce for distribution to WG 9.

Approve draft *Instructions to the ARG*: 10-0-0.

PRG proposals

The PRG again would like to add various functionality to the Ada Standard which they believe is more general than just POSIX. Randy reminds them that additions to Ada are not likely to contain all of the functionality of POSIX (as other operating systems may not have the needed functionality, so doing this is unlikely to save the PRG any work – they still need to make the complete POSIX functionality available.

Steve Michell presents a list of areas that they think should be handled in the Ada Standard:

- Processor affinity (probably a pragma),

- Barriers (improved performance of protected objects),
- Memory model (be able to control memory locations),
- Dynamic libraries,
- Sockets.

It is noted that we had a proposal to create a Sockets interface for Ada 2005, but no interface and standards wording ever materialized. We think sockets should be based on streams in some way.

Tucker suggests that the POSIX interfaces could be designed to support other OSes. For instance, the PRG could define a subset of the POSIX interfaces that could be (and should be) supported on Windows. That would make more sense than trying to bring all of this into the Ada Standard.

In any case, these topics are too vague to be able to make any decisions (recall our Amendment 1 experience with sockets). The PRG needs to provide AIs to discuss (and needs to do so before the appropriate deadlines in our development schedule).

Thanks

Thanks to our host (Ted Baker) for the impeccable meeting conditions and warm hospitality (if not warm exterior temperature).

Old Action Items

Brad did not do an update to AI05-0127-1. Randy did not do an update to AI05-0049-1 (he claims to have run out of time). Nobody did an ASIS example (although Bill Thomas did start working on one). Pascal did not propose new wording.

New Action Items

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

Everyone:

- Review assigned portions of the ASIS Standard document.

Steve Baird:

- AI05-0135-1 (split **use all type** from **use package**)
- Decide on directions for accessibility, equality, and accessors (AI05-0123-1, AI05-0138-1, and AI05-0142-1) with Tucker Taft and Randy Brukardt (see discussion of AI05-0138-1).

Randy Brukardt:

- AI05-0049-1
- AI05-0107-1
- AI05-0136-1
- AI05-0139-1
- Decide on directions for accessibility, equality, and accessors (AI05-0051-1, AI05-0123-1, AI05-0138-1, and AI05-0142-1) with Tucker Taft and Steve Baird (see discussion of AI05-0138-1).

Editorial changes only:

- AI05-0009-1
- AI05-0103-1
- AI05-0112-1
- AI05-0116-1
- AI05-0128-1

- AI05-0129-1
- AI05-0132-1
- AI05-0133-1
- AI05-0134-1
- AI05-0140-1
- SI99-0024-1
- SI99-0037-1
- SI99-0040-1
- SI99-0042-1
- SI99-0043-1
- SI99-0044-1
- SI99-0045-1
- SI99-0046-1

Bob Duff

- Create an ASIS example tool to find return-by-reference incompatibilities (preferably both using and avoiding the new semantic subsystem). Possibly reassigned to Sergey Rybin.

Greg Gicca:

- Add newly ARG approved SIs to the draft ASIS standard.
- Renumber clauses to eliminate so-called hanging paragraphs (at end of standard creation process, not now; see ASIS standard format in the Paris minutes)
- SI99-0048-1 (added after the meeting to create a summary of changes)

Matthew Heaney:

- AI05-0001-1

Pascal Leroy:

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

Brad Moore:

- AI05-0127-1

Ed Schonberg:

- AI05-0145-1

Tucker Taft:

- AI05-0146-1
- Write an implementation permission to allow copying to not actually assign elements in containers (see discussion of AI05-0001-1).
- Decide on directions for accessibility, equality, and accessors (AI05-0123-1, AI05-0138-1, and AI05-0142-1) with Randy Brukardt and Steve Baird (see discussion of AI05-0138-1).
- SI99-0045-1 (as soon as possible, surely before the ASIS review).

Bill Thomas:

- Create an example of the semantic subsystem for adding to Section 2 (see the discussion of SI99-0037-1 from meeting 36, also see discussion of SI99-0024-1 this time).

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2005 AIs. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the Amendment (with a /2 version), the number refers to the text in the Final Consolidated AARM. Paragraph numbers in earlier drafts may vary.

Detailed Review of ASIS Issues

SI99-0024-1/13 Provide a semantic model in addition to existing syntactic model

The introduction to ASIS definitely needs to mention the semantic subsystem. Bill Thomas should add that to his example SI.

We look at the various notes that Randy had left in the SI.

Leave the clause organization as Randy has it (and remove the editor's note).

The introduction "normative text" is actually normative in section 5. So that paragraph should be combined into the note, and referenced to Section 5. Then the complaint disappears (and should be deleted).

xx.1.7: Just have function Expanded_Name return Wide_String; (which is UTF-16). That is more consistent with the rest of ASIS.

Do the same for Static_String_Value return Wide_String; (which is UTF-16). That's in xxx.4.4.

xx.1.12: Jean-Pierre wonders if we should allow adding implemented-defined enumeration literals in the Aspect_Kind. Tucker doesn't think there is any advantage of that. Jean-Pierre says that you can loop over them.

Tucker would prefer having two separate types, Implementation_Defined_Aspect_Kinds (which is totally implementation-defined). That eliminates compatibility problems (especially for case statements) with implementation-defined aspects in the language-defined aspect kinds. Change the existing functions to use Implementation_Defined_Aspect_Kinds where necessary.

X.3.5: Remove note, this is OK.

X.4.3: We agree with Randy's recommendation to split out a package Asis.Object_Views.Access_Views. Tucker wonders if this should have a name about dereferences. It is mostly about dereferences, but the rest of the semantic subsystem names things based on categories of entities. Access seems more appropriate with this model.

XX.6.2 This is fine, delete the note.

Approve SI with changes: 10-0-0.

SI99-0037-1/05 Review standard for problematic wording

1.1.3.1 Tucker says that entities are wrong here. After discussion, we decide this should just say 'declarations'.

"An implementation shall not add any declarations to the visible part of logical packages defined in the following clauses of this International Standard."

Aspect clauses aren't declarations, so we don't have to mention them. It's weird to treat pragmas differently (why is Size ok but not Pack??). So we drop that, too. Thus we end up with a much simpler rule.

1.1.5 "[The following]{These} exceptions..."

10.17: "italize" should "italicize".

12.2: "th{r}oughout" "...no rel[e]ationship..."

13.37: This will be reviewed by the reviewers.

15.44: This is non-normative text; we can't find a good place for it. Perhaps we should get rid of it?

Tucker suggests making a new section 2.4.5.3 and putting this text there. (There probably are other such digressions that ought to be moved similarly, reviewers ought to suggest that when they encounter them.)

16.12: OK.

17.8: OK.

17.17:

It is thought that the wording could be better.

Returns a list of the `Array_Component_Associations` in an array aggregate. {If the aggregate is a positional array aggregate, the `Array_Component_Associations` consist of an expression of the aggregate {with}[, and] `Array_Component_Choices` that are {each} a `Nil_Element_List` for all positional expressions except {for the}[an] others choice{, if any}.

20.7: Fix the "..." character.

21.2: `Is_Equal` should be `Is_Idential`.

21.3: Just say that "<" is implementation-defined, we don't care what this means. Same for 21.4.

21.6: We'll leave the term "same physical compilation unit", alternatives aren't better. So delete all of the changes for this subclause.

22: OK.

D.2.3: OK.

3.12.3: We definitely don't want `A_Unit_Origin` to change.

"Units with an Origin of other than `An_Application_Unit` are implementation-defined. This will affect other ASIS return values for other than `An_Application_Unit`."

Greg will write a new paragraph to be discussed tomorrow.

We'll postpone voting until then.

On Saturday morning, Greg proposes the following change:

Suggested updated text for section 3.12.3

Old text:

Units with an Origin of other than `An_Application_Unit` are implementation-defined. This will effect the return value of function `Asis.Compilation_Units.Unit_Origin` directly and all other ASIS return values for other than `An_Application_Unit`.

New text:

--Implementation Permission--

The meaning of `Unit_Origins` values other than `An_Application_Unit` are implementation-defined. Thus the return value for `Asis.Compilation_Units.Unit_Origin` is also implementation-defined if the return value is not `An_Application_Unit`.

This is wrong; the return value of `Asis.Compilation_Unit.Unit_Origin` is just about the only thing that is *not* implementation-defined. We need to make it clear that some parts of language-defined and implementation-defined units may not be available.

Greg will try again, and he does on Saturday afternoon:

3.12.3:

Values other than `An_Application_Unit` or `A_Predefined_Unit` are implementation-defined.

[Notes/states:]

[Values of `An_Implementation_Unit` are suggested but not a required origin.]

[Values of `An_Unknown_Unit` can be any origin.]

10.3:

Current PP2 says:

Returns the `Unit_Origins` value of the unit. Returns `Not_An_Origin` for a `Compilation_Unit` whose `Unit_Kind` is `Not_A_Unit`, `An_Unknown_Unit`, `A_Nonexistent_Declaration`, or `A_Nonexistent_Body`.

Delete "`An_Unknown_Unit`" from above list.

Add:

ASIS queries on `Compilation_Unit`'s whose return value is other than `An_Application_Unit` or `A_Predefined_Unit` may have limitations that are implementation-defined.

21.3

Less than ("`<`") is defined in terms of the corresponding mathematical operation on the `Ids` based on the positional order of `Asis.Asis_Integer` that would be returned by the function `Hash` for each `Id`.

21.4

Greater than ("`>`") is defined in terms of the corresponding mathematical operation on the `Ids` based on the positional order of `Asis.Asis_Integer` that would be returned by the function `Hash` for each `Id`.

This wording still isn't quite right. Let's start by describing what we need.

The categories seem OK as described. For implement-defined units, nothing *else* in ASIS is required to work. For predefined units, we want visible declarations to work, but we don't want to expect that source code exists, that comments can be read, can get to bodies, and so on.

3.12.3 doesn't need a change.

10.3:

"ASIS queries on `Compilation_Units` whose return value is other than `An_Application_Unit` may have limitations that are implementation-defined."

This is too broad. For predefined units, the private part and bodies may have implementation-defined limitations, the package text is not expected to work anywhere. But the declarations on the specs of predefined units must work. (This has to be worded further.)

Leave the sentence on 10.3:

"ASIS queries on the private part and bodies of Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined. Operations of package Asis.Text on Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined. "

Tucker, Greg, and Randy will take this off-line at the break to polish this wording.

The 21.3 and 21.4 routines still seem weird. Should these be obsolete? We need to find out what Rational does, Steve Baird will ask Gary Barnes.

After the Saturday afternoon break, we try again with this wording developed by our draft (daft?) subgroup:

10.3:

Change paragraph 2: "Returns the Unit_Origins value of the unit. Returns Not_An_Origin for a compilation_unit whose Unit_Kind is Not_A_Unit, A_Nonexistent_Declaration, or A_Nonexistent_Body or possibly An_Unknown_Unit."

Implementation permission:

"ASIS queries on the private part and bodies of Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined. Operations of package Asis.Text on Compilation_Units whose Unit_Origin is other than An_Application_Unit may have limitations that are implementation-defined. "

21.3:

"Defines a total ordering on Ids."

21.4:

Equivalent to Right < Left.

This looks good to the group.

Approve SI with changes: 6-0-3.

SI99-0040-1/01 Primitive operations given by instantiation

"It is possible..." in the question.

The header says "SI99-0045-1" instead of SI99-0040-1, fix that.

Add an indication of the clause that is being modified to the !wording.

Approve (lunch and) SI with changes: 10-0-0.

SI99-0042-1/01 Handling generic children in ASIS

When does a "sprouted" generic exist? Steve Baird says when the **with** exists. But as a practical matter it only matters when the instance is created.

Jean-Pierre notes that the template that is the parent unit of the sprouted unit does not include the child (even though it is the "enclosing unit"). That's true of all child units; it is a bit weird in this case as it could be a non-library unit. But that seems OK (in the sense that it can be handled and doesn't introduce any new problems).

In particular, you won't find the child if you scan through the parent instance. Jean-Pierre notes that the second paragraph of the !discussion implies a different model. It should say that the relationship between child and nested generics should be the same as the relationship between child and nested packages.

We definitely think that these elements must exist, and that may be hard for some implementations. That is true for lots of things in ASIS for some implementations. Moreover, the alternative is to make ASIS users work harder (by requiring special code for child generic units, which would be difficult to identify at the point of the instantiation, and easy to get wrong), and we would need a better justification for that than to save some implementors effort. Thus, we have no interest in making a special case here.

Whether this particular ASIS feature (really a bug) in this particular implementation is implemented/fixed is an issue for that implementation to figure out (weighing all of the usual factors: the importance of the feature, the number of customers affected, the cost of fixing it, etc.); there is no reason for the Standard to be involved.

The SI should be structured and classified as a ramification.

Approve SI with changes: 7-0-3.

SI99-0043-1/01 Multiple A_Configuration_Compilation_Units are supported

Clause 3.12.1 says that multiple units are OK. So there is no problem. Jean-Pierre notes that the wording in the GNAT spec says what Sergey reported, we verify that ASIS 99 has different wording and that Sergey's wording does not appear in ASIS 99. Apparently the GNAT specs are based on some draft version of ASIS, and Sergey based his question on that.

There is some gibberish markup in the text of 3.12.1 in the draft Standard; it should be fixed.

Approve SI: 8-0-2.

The two people who abstained tell us that they don't quite understand the statement in 3.12.1; it is confusingly worded. Change it to "A context may have any number of units of A_Configuration_Compilation kind."

Rewrite as a Ramification (confirmations can't have wording).

Approve SI with changes: 10-0-0.

SI99-0044-1/01 Parent subtype without a declaration

This interface is totally broken, it doesn't make any sense even for a type with a constraint:

```
type T is new Integer range 1..10;
```

Definitely make this obsolescent, new code should use Parent_Subtype_Indication instead.

Approve SI with changes: 9-0-1.

SI99-0045-1/01 The meaning of Corresponding_Expression_Type

Someone jokes that this should be named Corresponding_or_Convenient_Expression_First_Named_Subtype.

We want to combine this so that all of the Nil_Element cases are together. We should simply say that anonymous types and classwide types return Nil_Element.

Take the entire list of existing bullets and put it into the AASIS. (Steve notes that an exception name might be an "expression" in the ASIS sense.)

Tucker will write it up and present it tomorrow.

Fix spelling of "litteral", get rid of the question marks in the question.

SI99-0045-1/02 The meaning of Corresponding_Expression_Type

Aggregates and attribute references should be implementation-defined.

So that has to be a separate sentence.

Approve SI with changes: 7-0-2.

On Sunday, Jean-Pierre sends comments about the approved wording. Tucker is assigned to update the SI to reflect Jean-Pierre's comments.

SI99-0046-1/01 Clarify the Context parameter in ASIS queries

Subject should be "ASIS", not "Asis".

Should this be obsolescent? The definition allows elements from different contexts to be equal only if they come from the same version unit (as described in the Ada Standard); that's pretty useless as that means that nothing (including dependent units) can change in any way. Most people think this could be used to compare two versions of the same unit, but that is not supported.

However, if someone does find a use for this, we don't have an alternative way to accomplish these tasks. That argues that these operations should not be obsolescent. Usually, we have a better way to do something that we make obsolescent. If an implementation doesn't support multiple contexts, they don't do anything interesting anyway, so it seems harmless. The discussion ought to be changed to reflect this.

The wording is OK.

Randy notes that 10.11 through 10.14 now specify the context of the returned units three times; get rid of the second one (the paragraph immediately following the example calls in 10.11 and 10.12, the one starting Implementation Requirements in 10.13 and 10.14) and add "Is_Identical" to the third one (the only useful thing from the second occurrence).

Approve SI with changes: 10-0-0.

Detailed Review of Ada 2005 AIs

AI05-0001-1/02 Bounded Containers and other container issues

Bounded Vectors

"It provides the same operations as the package Containers.Vectors (see A.18.2), with the difference that internal storage is allocated statically."

Is this OK? No.

"It provides the same operations as the package Containers.Vectors (see A.18.2), with the difference that there is no usage of dynamic storage allocation."

We don't like this, either, it's not clear what "dynamic storage allocation" means (we don't mean to disallow stack allocation, but surely that happens dynamically).

"It provides the same operations as the package Containers.Vectors (see A.18.2), with the difference that the maximum storage is bounded."

There is an organization problem: Make the changes for the bounded form into bullets, move the changes to other units to a separate place. And put all of the changes into clause order (that means that bounded forms will go after all of the changes to the existing sections).

Change "The package has Pure categorization." to "The package has a pragma Pure instead of pragma Preelaborate."

We need to say that these do not "need finalization", because that is an intended difference from the unbounded forms. Ed notes that if the element type needs finalization, the container type had better as well. So to the next bullet, add "The container type needs finalization if and only if the element type needs finalization."

Copy: Should it be possible allow a larger capacity? Unbounded typically does allow that (for instance, an implementation might chose that all actual capacities are powers of two). The bounded case should require the capacity to be exact.

Is "whose elements match the active elements of Source." OK? No, various ideas are tried: "whose elements correspond to the elements of Source." "whose elements are copies of the elements of Source." This needs to say that the length is the same.

"Returns a vector of the same length as Source, whose elements are copies of the corresponding elements of Source."

Erhard objects, he mentions that we need to say "assign" to ensure the use of Adjust. But this is a new object, it should be "initialized".

"Returns a vector of the same length as Source, whose elements are initialized from the corresponding elements of Source."

Capacity_Subtype: this doesn't seem to work. Is it OK to have a capacity greater than the maximum available? Sure, that might be stupid but harmless. But a vector with more real elements than can be indexed is a real problem. You could get that by continual appending, for instance.

So, we should add wording so that you get a Constraint_Error if the position of the last element would be greater than Index_Type'Last. That happens in any operation of type Vector.

[Editor's note: If you have conditional expressions, as proposed after the meeting in AI05-0147-1, writing the subtype becomes possible:

```

subtype Capacity_Subtype is Count_Type range 0 ..
(if Count_Type'Pos(Count_Type'Last) <=
  Index_Type'Pos(Index_Type'Last) then Count_Type'Last
elsif Index_Type'First >= 0 then
  Count_Type(Index_Type'Last) - Count_Type(Index_Type'First)
elsif -Count_Type'Pos(Count_Type'Last) >=
  Index_Type'Pos(Index_Type'First) then Count_Type'Last
elsif Count_Type'Last - Count_Type(-Index_Type'First) <=
  Count_Type'(Count_Type'Last) then Count_Type'Last
else Count_Type'Last - Count_Type(-Index_Type'First) +
  Count_Type'(Count_Type'Last));

```

The short-circuit behavior eliminates many problems, as the cases which potentially overflow are not executed.]

Describe wording changes to Move:

Reserve_Capacity shouldn't be required to copy anything, especially for the unbounded forms. (Pascal wanted to use a skip list, for instance.)

We need an implementation permission to say that when we say "copy", we don't necessarily actually require copying in the sense of calling Adjust, etc, so long as the effect is preserved. Tucker will take a stab at writing such a permission.

That permission means that Reserve_Capacity doesn't have to copy anything if it wants. So with that we have no issue with Matt's description.

One of the differences between the unbounded and bounded form is that the Implementation Advice for Move does not apply. It surely should not be deleted from A.18.2(261/2). We should simply say that it "does not apply" in the bulleted list of differences between the forms.

In this list of differences, you should say "replace the description of Reserve_Capacity by <text in the AI>".

Change the introductory paragraph to say "...change the contents {and semantics} of the package...". That makes it clear that the semantics also are changed. Really should make that change to all of the indefinite forms as well.

Matt did not do the following things from the minutes of the Portland meeting; these still need to be done (for all of the [bounded] containers):

These containers cannot be controlled, because Ada.Finalization is Preelaborated and these packages are Pure. There should be an AARM implementation note to that effect. (Put it immediately after the Pure categorization statement.)

'Read and 'Write should only stream Length elements, not the capacity elements (which is what the default implementation would do). Matt will need to create wording for that effect for only the bounded forms. ('Read and 'Write do not include the discriminants.)

Assign should tamper with cursors. We're adding Assign and Copy to the unbounded forms, so we should add this bullet to the tampering definition in A.18.2 (after paragraph 93):

- it calls Assign with V as the Target parameter;

And similarly for the other bounded containers.

Bounded Linked List

Tucker suggests saying that Reserve_Capacity is called to allocate internal storage. Then there exists wording saying that any exception is propagated. But we don't need to add Reserve_Capacity to the unbounded form of lists, as it is useless for them. So instead for the bounded list form:

- The allocation of internal storage includes a check that the capacity is not exceeded, and Capacity_Error is raised if this check fails.

Tucker says that the wording about "allocation of internal storage" is only used in lists. Darn.

Just use this description for Move for the bounded form only. This description doesn't work if Source and Target are the same. Tucker suggests leaving the wording alone and add a check at the beginning: "For both Move and Assign, precede the operation with a check: if Source length is greater than Target capacity, then the operation raises Capacity_Error, and Target is not modified." Remove the sentence about capacity from the Assign definition.

For the unbounded form, use a single parameter for Copy:

```
function Copy (Source : List) return List;
```

Returns a list whose element match the elements of Source.

Then use the existing description as a replacement in the list of differences.

Splice: The existing wording is fine, just add an extra check as described above.

Bounded Maps

Tucker says that for bounded Maps, we can say that "allocation of a node as part of Insert includes a check that the capacity is not exceeded, and Capacity_Error is raised if this check fails. All of the inserting operations work through Insert.

Again, don't mention capacity in the general wording for Assign. Then use similar wording for Move and Assign as given for lists previously.

Assign should be shared for all maps, with added text for each kind if needed. Copy should be defined separately everywhere, because the parameter lists are different (hashed forms have capacity, order forms don't).

Fix "modulous" in the start of the hashed maps and hashed sets.

Drop the discussion about Golden Ratio.

Replace "buckets array" with "number of hash values". Implementation-defined is OK.

"Default_Modulus returns an implementation-defined value for number of distinct hash values to be used for the given capacity (maximum number of elements)."

Bounded Sets

Reserve_Capacity under sets has a parameter of type Map! Better fix that.

Protected Queues

Matt wonders if we ought to have a priority queue. You would need a "<" operator on elements to provide the priority.

Should a priority queue have a different name than the current proposal? There is support from some people to do that, but it seems weird. Tucker says that the language talks about queues, some of which are ordered by priority. So it would be really strange to call them something else.

Perhaps we ought to have a separate interface for priority queues. We discuss structuring for a while; we eventually decide to let Matt design something.

Sorting

We should say "indexable structure" here – there is no container in sight. "Less" is used because this isn't comparing the indexes, it's comparing the "elements". Using the operator "<" would almost certainly compare the wrong thing. Maybe it should have a different name. "Element_Less" and "Element_Swap"? "Element_Before"? Jean-Pierre suggests "Is_Sorted"? Tucker suggests "Before" and that was chosen.

Should these have box defaults on the formal subprograms? No, it is unlikely that there will be any routines around that would actually have the correct parameters and name. (Swap for containers doesn't work here as there is no container parameter.) If there is, it probably won't be the ones intended.

We're getting hungry enough that we leave immediately for lunch and forget to vote.

AI05-0009-1/07 Confirming rep. clauses and independence

Randy explains that Pascal disliked the fact that there is no semantics for the pragmas defined in C.6. That is pretty weird. But moving the main semantics out of 9.10 would be wrong: independence is a core language concept. So he added a sentence in C.6 to point at 9.10.

Tucker suggests defining "specified independent addressability" in C.6(14):

Pragmas Independent and Independent_Components *specify independent addressability* for the named object or component(s), or in the case of a type, for objects of that type.

Then in 9.10(1), use "specify independent addressability":

Any two nonoverlapping objects are independently addressable if either object is atomic (see C.6) or if either object is specified as independently addressable (see C.6)."

Move the first AARM ramification paragraph of 9.10(1) after C.6(14). Drop "Similarly" from the second AARM ramification in 9.10(1).

[Editor's note: After the meeting, it was noticed that the terms are different for the definition ("independent addressability") and use ("independently addressable"). We also noticed that saying that atomic objects are independently addressable simplified the wording further. So the wording for C.6(14) was changed to:

Pragmas Independent and Independent_Components *specify as independently addressable* the named object or component(s), or in the case of a type, all objects of that type. All atomic objects are specified as independently addressable.

And the wording for 9.10(1) was simplified to:

Any two nonoverlapping objects are independently addressable if either object is specified as independently addressable (see C.6).

This also means that both AARM notes move from 9.10(1) to after C.6(14). Consider this the editor's editorial review of the AI.]

Approve AI with changes: 6-0-2.

AI05-0103-1/03 Return statements should require at least static compatibility

Replace "Ouch." with "Is this intended? (No.)" in the question.

We discuss the static matching rules for access types for a while, and eventually decide that it would be more work for compilers to relax these rules, and there would be no reason for that work.

Erhard notes a typo in the last paragraph of discussion. "... the [the] ..."

Approve AI with changes: 8-0-0.

AI05-0107-1/01 A failed allocator does not leak memory

Randy starts explaining the problems that this AI needs to solve.

Tucker asks to change the requirement on finalization to start not later than the point of the innermost master, as it might be possible to do it immediately, and that should be safe.

A trio of typos are noted:

Erhard: "...the [the]..." in the wording, second paragraph.

John: second paragraph of the summary: "...call [at] during..."

Brad: first paragraph of summary: "...and then [to] call..."

The dynamic semantics header should be inserted before paragraph 16, then put the new text immediately after that.

These bullets probably ought be implementation requirements; this defines the only places where an implementation can make an *implicit* call to Allocate.

Tucker says that the first three bullets are essentially the same. They probably should be combined. "During the execution of an assignment operation of the target of an allocated object of type T with a part that has an unconstrained discriminated subtype with defaults."

[Editor's note: Upon reflection, that's not quite right; we need to know where exceptions can be propagated from Allocate to a local handler; that includes allocators, assignment statements, and return statements (for build in place). That means the third bullet is **not** equivalent to the other two. We surely do not want to have to figure out how to "unhandle" an exception that comes from a call to Allocate in an extended return statement so that it is handled in the allocator and not the return statement.]

The Deallocate calls need similar changes.

Randy will try to rework the wording to make those changes.

Gary asks that the hyphens be removed from "built-in-place".

Approve intent: 7-0-2.

AI05-0112-1/01 Names for anonymous aspects of representation

Typos: in the discussion "What is {the} name..."

13.1(9){ }states [there is a space missing]

In the summary "specif{i}ed".

Gary points out that there is no !question. Use the first paragraph of the discussion as the question.

Erhard insists on deleting the first line of the summary.

Approve AI with changes: 8-0-0.

AI05-0116-1/02 The value of Alignment for a class-wide object

AARM note 13.3(32.a) "...need to allow[ed] inherited..."

Move the otherwise in 13.11(16): {if D is a specific type, and otherwise is the Alignment of the specific type identified by the tag of the object being created}.

Correct second paragraph of the summary: "...{if}[of] this is for..."

First AARM note in wording: "A tagged object should never be less aligned than the alignment of the type of its view, so for a..." also: "...then that {of} T." "suggest that" should be "recommend this".

Answer the question: "(The alignment depends on the tag.)"

Approve AI with changes: 8-0-1.

AI05-0123-1/03 Composability of equality

We look at this AI for a few minutes, then decide that the time allotted for the meeting has expired. We decide to have Randy, Steve, and Tucker add this to their discussion for accessibility, and then we'll provide a new write-up.

AI05-0128-1/01 "/=" is a primitive operation

Eliminate hyphen in "explicitly-declared".

Correct the summary: drop "for the purposes of use type clause", because the change is general and this implies that it only apply for use type. Change it to: "The implicitly declared "/=" operator is primitive if "=" is explicitly declared."

Add the missing !wording in front of the wording.

6.6(6) appears to be wrong with this, we'd get two "/=" operators in some cases. It should say: "An explicit declaration of "=" whose result type is Boolean implicitly declares an operator "/=" that gives the complementary result."

Approve AI with changes: 9-0-0.

AI05-0129-1/02 A limited view does not contain views of incomplete types

Change 10.1.1(12.3) wording to:

- For each type_declaration in the visible part {that is not an incomplete_type_declaration}, an incomplete view of the type; if the type_declaration is tagged, then the view is a tagged incomplete view.

In the discussion, quote "hidden from all visibility" to make this more readable.

In the question, in the paragraph starting "The question...", "chose" should be "choose".

After some reading, we realize that this wording is already recursive, so the question is wrong. Delete the penultimate sentence of the question, since it isn't true. Similarly, drop "but not the associated type" from the discussion.

Add "{directly} in the visible part" to both paragraphs. We don't want to consider types or packages in anything nested, so make it crystal clear.

Approve AI with changes: 7-0-2.

AI05-0132-1/01 A library unit pragma must be used in a library unit

Change subject and summary "must be used in" should be "must apply to".

Approve AI with changes: 9-0-0.

AI05-0133-1/01 Extending a type with a self-referencing discriminant constraint on a component

Discussion: "to an an" should be "to be an"

Last paragraph, drop second 'u' from curiosity.

Answer question with "(See discussion.)"

Tucker notes that the access type is looking at a view that is of type Root. So it is sort of an implicit type conversion.

Approve AI with changes: 8-0-1.

AI05-0134-1/01 Full conformance should include the profiles of anonymous access-to-subprogram parameters

The question got chopped off: "...requiring the profiles for access-to-subprogram parameters to be fully conformant."

Note that if this was allowed, we wouldn't know which of the two different

default expressions would be used in a call of the example. So it is important to fix this.

Wording: "access-to-subprogram result{s} [types]"

Approve AI with changes: 9-0-0.

AI05-0135-1/02 An extended-scope use clause

Steve explains how this works. There is some discomfort with the idea that this essentially forces a use clause into clients of a package. Use-adverse programmers will not be happy. It seems especially weird when it is used on a package that is defined outside of the current unit.

Tucker suggests that perhaps we need anonymous packages (he suggests using the getting-to-be-overused $\langle \rangle$):

```
package <> is
  ...
end <>;
```

These would then have effectively a use all (as defined in this proposal). Then we don't need the more general idea.

Ed asks if there are any examples where this would not work? No one can think of a real case where we needed the more general **use all**.

The **use all type** to get all primitives is liked (and totally unrelated to the other ideas).

Erhard notes that the implicit use of an anonymous package makes it harder to find where objects come from. The Ada 83 rule is that everything has an expanded name, and **use** clauses are sort of a fix for verbosity that results.

Erhard thinks that adding a new item in an inner package could make accessing something impossible. He gives an example:

```
package G1 is
  procedure Foo;
end G1;

package G2 is
  --procedure Foo; -- (1)
end G2;

package P is
  package <> is new G1;
  package <> is new G2;
end P;

with P; use P;
procedure Proc is
begin
  Foo; -- (2)
end Proc;
```

If (1) is uncommented, (2) disappears, and it becomes illegal. And there is no way to disambiguate because the packages have no name. This is clearly bad.

Tucker argues that Foo doesn't make sense; real subprograms have parameters and they can be used to disambiguate the call. But say the declaration in question is an exception (say "Not_Valid_Error"); that seems like it could happen.

Perhaps we need to give this "anonymous" package a name, then the full name is possible if needed for disambiguation. So the suggestion is to add the keyword **use**.

```
package P is
  use package I1 is new G1;
  use package I2 is new G2;
end P;
```

Then you can reference Foo as I1.Foo and I2.Foo.

Steve will rewrite the AI to define **use package**.

Split the **use all type** into a separate new AI.

Keep alive - **use all type**: 8-0-0.

AI05-0136-1/02 Multiway tree container

Randy explains the reasoning behind the tree container: it essentially is that each element has a list of children.

Tucker asks why this is better than a list of lists. Randy notes that you want each element to be able to point at a list that contains the element type itself. There isn't any way to do that without using an access type, and that prevents the container from doing all of the storage management (which is the best reason to use a container like this).

There is a lot of objection to having this to be a forest (multi-rooted). Several people think that the root is special. Randy and Tucker argue that you want to create partial trees, and the root isn't special in this case.

Erhard notes that you can't use a single container to build bottom up unless you have multiple trees. The problem is that if you don't have a forest, then you also need auxiliary container objects. Those aren't a logical problem, but can be a performance one, especially for the bounded forms, where linking subtrees into larger (sub)trees would require copying the subtrees (a horrible cost if the elements or subtrees are large and many steps are needed).

Ed comments that this would perform a bit worse than writing this yourself, as you probably wouldn't use all of the links (up and down, back and forward) in a custom design. Randy notes that this design lets you use operations that would have been too hard to implement in a custom design without all of the links (for instance, deleting an element, or looking at the previous element), and it ensures that all of the links are correct for any operation. In any case, the performance hit is small (especially if the elements are large). If you really need maximum performance (time or space), you'll surely use a custom data structure. But 95% of the time you don't need maximum performance, just decent performance.

While Ed and Randy were discussing performance, Tucker has been looking at how the Dom for XML (which has a very similar structure) handles roots and unattached subtrees. He says that the Dom allows inserting an element without putting it into the tree in a particular place. It is part of the document, but not necessarily linked in. You could access them by an iterator (or by holding onto a cursor). Randy notes that this is essentially the same as the forest approach, except that it designates a particular node as *the* root. There is more comfort with this approach; it seems to allow the best of both possibilities.

So we probably need a Root (to retrieve the root) and Set_Root operation. We also need to have an iterator that visits all elements linked to the root, and one that visits all of the unattached elements and their children.

Randy will do an updated version along these lines.

AI05-0138-1/00 Improving accessibility

Tucker, Randy, and Steve will try to make sense of these proposals. Try to involve Bob Duff as well. They will set up a teleconference to discuss these issues and hopefully come to some conclusions. Tucker suggests March 19th, 12 noon CST (1 pm EST, 10 am PST). Steve will set up the telecom. (He will be gone until March 8th).

AI05-0139-1/01 User-defined iterators

Randy explains that he is proposing that objects that have an appropriate "magic" interface can be used in for loop syntax.

The resolution for the object should be defined as "A type covered by Basic_Iterator'Class".

Tucker suggests that this should be a single type context, where there is no context. We really don't want to be using the kind of interface needed here (that would be a new kind of resolution). The requirement to be a type covered by Basic_Iterator'Class would have to be a legality rule.

The AI should say that this is a place where a limited function can be called, rather than a "built in place context". (It would be added to the list in 7.5.)

Randy screwed up the syntax for a derived interface. It should be:

```
type Reversible_Iterator is limited interface and Basic_Iterator;
```

The operations of the interfaces need to be abstract, of course.

Ed wants to be able to directly use containers in the iterator. Randy explains that is possible; he didn't use it in the example because of the need to set/clear a tampering context (in the predefined containers). Several people are confused by the model that Randy is proposing.

There would need to be a legality rule that exactly one Basic_Iterator interface is implemented for the object passed.

Randy notes that Bob asked for the type name of the cursor to be provided in the syntax (rather than being just implied). No one thinks that is a good idea.

Tucker suggests that we create a separate AI to add these to the containers. We'll need to see how that works in order to make a decision as to whether to add this to the language.

Do we want to do this? Yes, Tucker notes that he's wanted this for a long time.

Randy will draft a new version with wording.

John asks what other languages do. Tucker says that Java does something that is almost exactly this.

Keep alive: 7-0-1.

AI05-0140-1/01 Identity functions

Randy explains the purpose: to provide a default implementation for interface routines that are functions. It is odd that these exist for procedures but not for functions. That forces designs that have rarely overridden operations to use procedures when functions would be more appropriate.

We definitely would want to use the parameter name (as opposed to the number as in the proposal). We need something to make it look different than just a name, since we would want to use this as a kind of default for generic formal functions.

Parenthesizing the name is suggested:

```
function Dereference (Pool : in out Root_Storage_Pool;  
                    Access_Value : in Address)  
return Address is (Access_Value);
```

Steve Baird wonders how we would deal with the fact that we could inherit two of these functions that specified different parameters from different interfaces. He reminds us that we allow inheriting two conforming null procedures from different interfaces without requiring overriding them; but they are identical in every way. Someone notes that two functions that specify different parameters would not fully conform, and we have rules if the subprograms don't fully conform (the subprogram must be overridden).

No one is strongly in favor of including this idea in the language. Randy notes that he mentioned it as an aside in another proposal; he was not intending to seriously propose it.

No action: 8-0-0.

Mention in the AI that the recommended syntax would be the parenthesized name of the parameter, in case it is ever resurrected.

AI05-0145-1/02 Pre- and Postconditions

Tucker explains how he came up with this version. He started with that last version of AI95-0288-1, and then added the experience with such pragmas in GNAT.

Note the deferred resolution to end of the visible part. This makes it easier to write expressions. But this only allows using *visible* entities; you don't want to allow private declarations to be used in these visible expressions.

The question of why we are using pragmas here. In the past (such as overriding indicators), pragma solutions have proved less than ideal, and we ended up inventing new syntax. Isn't that true here? Tucker thinks that is it easier this way, it is less of a change, and it is backwards compatible work with older compilers (new syntax that wasn't recognized is a dead end with an old compiler).

The pragmas are intended to be used in the visible part of a package specification. These shouldn't be allowed on bodies.

What about null procedures? Steve Baird thinks they shouldn't allow these pragmas on them. Regular ones surely shouldn't be allowed on abstract subprograms (they could never be used). But the classwide ones could make sense even for abstract subprograms. Both kinds could make sense on a null procedure (as it can be called). Steve Baird wonders what happens when two null procedures are selected for different interfaces? We allow that because they're all the same, but that isn't the case if the preconditions aren't the same. Humm.

What about renames? Ada doesn't allow changing the contract on a renames, so no preconditions on renames. (Although you can *write* a different contract, it is ignored; we sure don't want any more of that in Ada!!) No preconditions on formal subprograms (they work like renames).

Randy notes that if we had syntax for them, we could allow matching pre and postconditions for a generic formal.

Steve claims that the caller in general doesn't have enough information to ensure that the precondition is met, for a dispatching call. This argues to always have the callee evaluate. Tucker would like to allow the caller do it. Ted says that the caller doesn't care about the additional information that is checked. Note that it matters which side does it if one or the other side suppresses evaluation of the conditions.

Randy objects. The programmer could have a bug that the postcondition would have caught, and they would never understand why it did not catch the problem. That would be horrible. Tucker changes his mind and agrees with Randy. The callee has to evaluate these.

Remove the subprogram name from the pragma, as it must directly follow the subprogram declaration. It's just noise here.

Randy is dubious about removing the name; it could cause issues with cut-and-paste. It is felt that isn't a major problem because of the references to subprogram parameters. They usually would cause an error if the precondition was on the wrong subprogram. (But see below.)

Erhard asks why the rules are different for the string expressions. He would like to refer to the parameters: Integer'Image(P) & " is out of range". Good idea, the two arguments to the parameters should be the same.

There needs to be a legality rule preventing multiple pragmas of a single type on a single subprogram. Such a rule would also would help mitigate Randy's concern.

Exceptions in postconditions can't be handled by the called subprogram. Their failure represents a bug. Steve Baird says that assigning -1 to an out parameter of Natural can be handled by the called subprogram; that's just a violation of a postcondition. He claims that you could have a global handler to catch all bugs. That still seems weird and not worth changing the model for.

For interfaces, abstract and null subprograms can't have non-inherited pre- & postcondition. They could never be used. To make it simpler, do that for all null subprograms. (That answers most of Steve's earlier concern.)

Brad wonders about using these pragmas on pragma Import. That's OK, it's the callee's responsibility to do the right thing. If they don't, tough – it is then erroneous by B.1(38.1/2).

Ted says that the callee should check preconditions, and the caller should check the postconditions, because that is the safest case. If you are being called by an untrusted caller (or returning from an untrusted call), this would make things safe.

What about a unit where checks are off called from a unit where checks are on? It seems unsafe to let the postconditions to be suppressed in that case.

So that suggests we really need to do these checks on both sides (at least logically). Compilers can eliminate either side based on whether enough is known about the other side's checking.

Are these allowed on entries? Ed wonders what happens if a task entry fails a postcondition, the caller must get an exception, but does the task itself get the exception or continue along? Both get an exception in this case.

If the caller of an entry fails a precondition, then you don't want the callee to get an exception. So in that case no rendezvous is started.

John is confused by the example in the AI. But it is right; he was confused by the "condition". We think about this some and decide that "condition" grabs the eye too much, and this is the unimportant part. So just call these "pre" and "post".

Randy notes that these preconditions and postconditions all would be legal if accidentally attached to the wrong subprogram; that could not happen with syntax or with the subprogram name that has been dropped. (That happens because the stack parameter has the same name in all of the subprograms; something similar happens in the predefined containers, and is probably common in abstract data type packages.)

For protected objects, the precondition should not lock the object. Randy wonders about a modification; Tucker claims that can't happen. Jean-Pierre notes that evaluating the precondition could involve blocking, that couldn't work inside. Randy thinks it still sounds dangerous.

Steve says that we have to say where this happens relative to the hand-off and other operations defined in 6.5.

Entries seem pretty complex, we could just not allow pre and post conditions on them.

Note that would it be strange to disallow protected subprograms, they're subprograms. So it's not clear that this helps much (and would be really strange when implementing an interface with an entry).

Steve wonders about Initialize and Finalize. (And other compiler generated calls like pools and streams.) So we need to do the evaluation inside the subprogram.

Ed will write the next draft of this AI. That will include some wording.

Tucker wonders if we need the message part of these parameters? Erhard says that pre and postconditions shouldn't be for debugging; they are logically part of the contract. But messages are primarily for debugging, so these probably shouldn't have messages. (Use pragma Assert for that.) Besides, which message do you get if there are multiple preconditions that fail?

Can there be more than one precondition and postcondition? Tucker worries that it would be very confusing as to how these are combined, it's much better if the user writes what they want. Then they won't be confused as to whether it is "**and**" or "**or**". So only allow one of each. But remember that there still can be classwide and normal preconditions on the same subprogram (so there will be some combining going on).

Tucker asks if using the Assertion_Policy at the position of the pragma as the important thing causes any issues. We didn't find any (after discussion). Note that it could be different for different inherited conditions.

Steve asks what happens when 'Old is used on an object with controlled component. That implies a copy, with the entire mess of Adjust, etc. That needs to be clearly stated.

John asks if Pre_Condition and Post_Condition would be better. Perhaps, we should try that.

Randy asks which parameters are copied. Only those that are used for 'Old. And they are always copied in that case, the compiler can optimize using an as-if optimization. 'Old could be allowed on anything, but that can get really confusing. Some ways to describe that could require a time-machine to predict the future. For instance, X.all'Old could be asking for the old value of a new pointer.

Tucker thinks you do need to be able to ask about things that are global in postconditions. 'Old really should apply to any name (all evaluated before the call, no time machine needed).

Approve intent of AI: 6-0-4. Tucker surprisingly abstains, he is asked why. He thinks it changed a lot, and there are new ideas raised that should be considered. Keep it alive: 10-0-0.

Tucker woke up in the middle of the night on Sunday (4:30 am) and came up with syntax for these conditions.

Tucker sent us the following note:

I'll admit I have been bitten by the syntax bug (thanks, Randy!) for pre- and postconditions, and with invariants coming up I suspect it will continue. Here is a general proposal that avoids introducing new reserved words, but is quite flexible:

To specify aspects of an entity (at least "operational" ones) and link them directly to the associated declaration, consider a general syntax of:

```
<declaration>
  with
    <aspect_name> [=] <aspect_value>,
    <aspect_name> [=] <aspect_value>,
    ... ;
```

E.g.:

```
function Pop(S : in out Stack) return Elem
  with
    Pre => not Is_Empty(S),
    Post => not Is_Full(S);

type Even is new Integer
  with
    Invariant => Even mod 2 = 0;

type Atomic_Array is
  array(Positive range <>) of Natural
  with
    Atomic_Components;

type Set is interface
  with
    Invariant'Class =>
      (Is_Empty(X)) = (Count(X) = 0);

function Union(X, Y : Set) return Set
  is abstract with
    Post'Class =>
      Count(Union'Result) = Count(X) + Count(Y);

type R is record
  X : Positive := 0
    with Independent;
  Y : Natural := 77
    with Atomic;
end record;

type Shared_Bit_Vector is array(0..15) of Boolean
  with Packing, Independent_Components;

type Bit_Vector is array(0..15) of Boolean
  with Component_Size => 1;
```

The notation `<aspect_name>'Class` was intended to imply an inherited, non-overridable aspect. The `'Class` might be implicit for abstract tagged types and their primitive subprograms.

Somewhat independent suggestion:

Add "**X implies Y**" as a new short-circuit operation meaning "**(not X) or else Y**".

By making it a short-circuit operation, we avoid the burden of worrying about user-defined "**implies**" operators (which then might no longer mean "**not X or Y**"), boolean array "implies" operators, etc., and the compiler can trivially transform them to something it already knows how to compile.

I suspect **implies** will be used almost exclusively in Assert and pre/post conditions.

Steve says "how about conditional expressions?" That's surely more complex than this proposal.

Ed: This looks appealing. Randy: I agree.

Gary: This doesn't work for aspects that need more than one parameter.

Erhard notes that some of these read funny. These are the aspect names, which we've never used before. But we could rename some of the aspects to make their names read better – no existing code can depend on those names.

Split these two ideas.

Keep **implies** alive: 7-0-1.

Keep syntax alive: 7-0-1.

[Editor's note: After the meeting, further discussion has directed the **implies** proposal back to the conditional expression idea mentioned by Steve, in large part because it is both more generally useful and more understandable. A proposal for that has been created as AI05-0147-1 and the **implies** action item canceled.]

AI05-0146-1/01 Type and Package Invariants

Tucker explains the proposals.

Invariants are enforced on exit from visible subprograms in the package, so they're essentially implied postconditions.

Randy asks whether the user-defined constraint idea is worth looking at (either as an alternative or replacement). After discussion, we decide that it seems to solve a somewhat different problem - it allows adding contracts to particular parameters, objects, etc. User-defined constraints would be a way to deal with non-contiguous sets of discriminants, one-sided array constraints, and so on. There is sufficient interest to have that written up (it previously was discussed on Ada-Comment and was filed as AC-0157). It's not very necessary on scalar types, so if the rules get too messy for them, don't allow them. (Randy notes when writing up these minutes that that would probably be a generic contract problem.) Steve notes that it would need a bounded error if the expression does not return the same value when called with the same value (we would want to be able to eliminate duplicate checks) -- the bounds are that the check is either made or not. Randy will write this up.

Returning to invariants: Erhard notes that Invariants really only work on a private type. They'd have to be checked on every modification for a non-private type, which is impractical. But he worries that children can mess around with the internal state. Tucker notes that the check is enforced only on visible procedures of the subsystem (that could include visible children). So there is no check on internal-only children and subprograms only defined in the body.

The function result case has the same rules; it's only enforced on return from visible functions. That's not clear in the write-up.

Gary wonders if you can say `type_name.discriminant` in the invariant expression. Only if the discriminant is visible.

A package invariant would be most useful for a package with a lot of state: an abstract state machine.

Split out the package invariants into a separate AI.

Tucker notes that package invariants don't work very well with the new syntax – where do they go?? And they seem redundant with postconditions. They would be best put syntactically at the end of the visible part. That sounds ugly. So just drop package invariants completely.

Use the new syntax on the next round.

Keep alive (type invariants only): 8-0-0.