Minutes of the 24th ARG Meeting

17-19 September 2004

Madison, Wisconsin, USA

Attendees: Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Gary Dismukes, Kiyoshi Ishihata, Pascal Leroy, Erhard Ploedereder, Tucker Taft.

Observers: Matthew Heaney (Saturday only).

Meeting Summary

The meeting convened on 17 September 2004 at 09:15 hours and adjourned at 13:00 hours on 19 September 2004. The meeting was held in room 832 at the Best Western Inn on the Park, in Madison, Wisconsin, USA. The meeting was hosted by Randy Brukardt of AXE Consultants. The meeting covered most of the agenda, discussing all amendment AIs on the agenda, as well as the majority of normal AIs.

AI Summary

The following AIs were approved:

AI-334/02 Is overriding of abstract equality required? (8-0-0)
AI-384/01 Conversion rules should be symmetric (9-0-0)

The following AIs were approved with editorial changes:

AI-51/12 Size and Alignment clauses (8-0-1)
AI-133/04 Controlling bit order (8-0-1)
AI-279/05 Tag read by T'Class'Input (7-0-2)
AI-284-02/01 New reserved words (9-0-0)
AI-285/09 Support for 16-bit and 32-bit characters (8-0-1)
AI-318-02/05 Limited and anonymous access types (5-0-4)
AI-333/04 Other locking policies can be used with FIFO_Within_Priorities (8-0-1)
AI-355/05 Priority Specific Dispatching including Round Robin (9-0-0)

The corrections to the following approved AIs were approved:

AI-212/04 Restrictions on configuration pragmas (9-0-0)
AI-217-06/10 Limited With Clauses (9-0-0)
AI-224/10 pragma Unsuppress (9-0-0)
AI-231/10 Access-to-constant parameters and null-excluding subtypes (8-0-1)
AI-252/08 Object.Operation notation (9-0-0)
AI-254/09 Anonymous access to subprogram types (9-0-0)
AI-287/07 Limited aggregates allowed (8-0-1)
AI-344/05 Allow nested type extensions (9-0-0)

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

AI-260-02/01 How to control the tag representation in a stream (9-0-0)
AI-302-03/06 Container library (7-0-1)
AI-357/05 Support for Deadlines and Earliest Deadline First Scheduling (7-0-2)
AI-359-04/01 Abstract generic instantiations (8-0-1)
AI-370/03 Environment variables (8-0-1)
AI-377/01 Naming of child packages (8-0-0)
AI-382/01 Current instance rule and anonymous access types (9-0-0)

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

AI-158/03 Renamings of primitive of a class-wide generic actual type
AI-291/03 By-reference types and the recommended level of support for representation clauses
AI-366/03 More liberal rule for Pure units

The following AIs were voted No Action:

AI-260-01/04 How to control the tag representation in a stream (9-0-0)
AI-284/04 Nonreserved keywords (9-0-0)
AI-315/02 Full support for IEC 559:1989 (8-0-1)
AI-318/03 Returning {limited} objects without copying (9-0-0)
AI-359-01/03 Deferring Freezing of a Generic Instantiation (9-0-0)
AI-359-02/02 Deferring Freezing of a Generic Instantiation (9-0-0)
AI-359-03/01 Delayed completion of a generic instantiation (9-0-0)

Detailed Minutes

Meeting Minutes

The minutes of the 23rd ARG meeting were approved by acclamation.

Next Meeting

The next meeting (25th) will be held following the WG9 meeting at SIGAda '04, in Atlanta, Georgia. The dates are November 19-21. The panel session on Ada 2005 is to be the (entire) morning of November 18th; WG9 will be the afternoon of the 18th.

As discussed last time, Jean-Pierre will host the following meeting in Paris, February 12-14, 2005.

We look at our schedule. Most AIs need to be done by Atlanta; the last few will be completed in Paris. We need to deliver a completed document to WG9 well before the meeting at Ada Europe (the week of 20-24 June, 2005). We may need a meeting for line-by-line review of the completed drafts. That should be in late April.

Perhaps have a meeting around April 17 or 24 for line-by-line review. Not sure if we will meet in York, depending on the work needed. Such a meeting should be in the US (because the other 2005 meetings are in Europe). An East coast location would be best; Tucker volunteers to host a meeting if needed. ARG members should reserve time for this review.

If we have this review, we won’t need to meet in York (or, at least our meeting could be short, since many of us will be there anyway).

Corrections Procedure

We’re starting to generate quite a few corrections to the completed AIs to fix issues found during integration and also because several implementers have started work on parts of the Amendment. Formally, we ought to reopen the AI or create a new AI to fix problems. But that is too heavyweight; we’d be bogged down in administrative tasks. Moreover, WG9 will have another chance to review the wording anyway. So we’re proposing to prepare needed wording changes before the meeting, review the change at the meeting, and take an approval vote. If there is a consensus, then we’ll use the change. If not, we’ll open or reopen an AI, and go through the normal process. Someone asks if we’re required to provide WG9 AIs. Not really, we’re only required to provide WG9 finished documents; giving them a sneak preview of the AIs is more of a courtesy than a requirement. The Corrections procedure is approved by acclamation.

SC22 action on AI-285

We asked SC22 to approve our approach to character set issues. Most countries voted Yes on the proposal, however we got a No from UK without comments, and a No from Germany with comments. During the SC22 meeting, a resolution was approved (4-0-1) which asked us to look at TR 10176. We have responded to the comment about TR 10176. So, our approach was approved. We need to avoid changing the AI much so as not to invalidate that approval.

WG9 action on reserved words (AI-284)

WG9's vote was all for new reserved words or to abstain. Thus, non-reserved words are dead; we'll handle that later in the meeting.

Thanks

Although there was no formal motion to that effect, the ARG thanks our host, Randy Brukardt, for the fine facilities and choice of restaurants, which will remain in the annals of the ARG as rivaling those of Padua.

Old Action Items

For the first time in memory, 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 and new action items from the meeting are shown below.

Everyone:

- Read sections 1, 10, and 11 of the early draft AARM and send comments to Randy and Pascal.

Steve Baird:

- AI-291 (with Tucker Taft)
- AI-377

John Barnes

- Update the Introduction of the Standard to reflect the updates of the Amendment. Redo the list of "Language Changes" to be more like the one in the Ada 95 Standard.

Randy Brukardt:

- AI-260-02
- AI-302-03

Editorial changes only:

- AI-51
- AI-217-6
- AI-231
- AI-252
- AI-279
- AI-284-02
- AI-318-02
- AI-333
- AI-344
- AI-355
- AI-384

Alan Burns

- AI-357

Pascal Leroy:

- AI-133 (before editorial review)
- AI-285 (text of notes)

- AI-302-3 (attempt to create wording for Ordered_Maps and Hashed_Sets)
- AI-370
- AI-382

Erhard Ploedereder:

- AI-366

Tucker Taft:

- AI-158
- AI-291 (with Steve Baird)
- AI-359-04

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 explains that only the Foreword, Introduction, and Sections 1, 10, and 11 of the draft AARM that he handed out were updated. Those are the only sections that should be reviewed at this time.

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

Q) Should we refer to this as Ada 0y, Ada 05, or Ada 2005 in annotations? (This shouldn’t occur in normative text.)

A) Ada 2005 is preferred by a wide margin.

Q) Should paragraph numbers in added subclauses be normal, or should they work like other insertions? (See 11.4.2 for an example.)

A) Use regular paragraph numbers. 0.1 is too goofy. If insertions at the end of a subclause, do the same. For something like 10.1.2, ask the ARG if it is OK to change the paragraph number of the note. (As long as the version number (“/2”) is included, there cannot be any confusion.)

Q) Should we use the (directional) left and right quotes as intended in the generated HTML? This doesn’t work on some older browsers, and perhaps not on non-Windows systems without Unicode fonts. We could replace both with straight quotes, or use back quote for left and quote for right (the latter is what the current Standard does).

A) Tucker’s Macintosh displays the characters correctly. We decided to use the modern characters. Where pairs of double single quotes are used (certain index entries), change these to use double quotes. (The double single quotes were used to match the original Ada 95 standard.)

Q) The Ada Europe copyright statement leaves out the express permission to use compiled copies of the library units without containing a copyright notice. Should this be changed?

A) No, the corrigendum copyright is already missing this permission. (Probably drop it from the AXE one as well, for the same reason.)

Q) Do we need to show changes in cross-reference text? (This is not done. See 1.1.2(13) for an example of where it would matter.)

A) Don’t bother tracking this change.

Q) Do we need a separate search facility? (The current button searches the existing consolidated Standard, not this draft.)

A) Randy is asked how difficult this is. Not very; a few hours at most. Therefore, change the search facility before releasing to public. But it's not necessary for versions that are only internal to the ARG.

Q) Should the AI numbers be complete, including the "95", the extra zeros, and the -01 for the alternative number?

A) Use the "official" full number.

Q) How should the Foreword and Introduction be handled? This is ISO required material, we probably ought to update them, and include the changes in the Amendment document.

Randy explains the proposed changes. "Foreword to this version" would be deleted; the Acknowledgement that it currently has would be moved to follow the Ada 95 one. (The Ada 2005 Acknowledgement would follow this.)

The Introduction text should be updated to reflect the changes to the language. Action Item to John to update the Introduction — especially Design Goals.

The list of Language Changes is more specific than the Ada 95 one. It should be redone. John Barnes is drafted to do that as well.

What should be reviewed? Ultimately, we need to review the Amendment document for typos (as the text is generated separately from the RM/AARM), but it is unlikely that many important errors (like conflicts with other text) will be found that way — it's just a list of paragraphs. For now, concentrate review on the AARM, which allows reviewing index entries, glossary entries, and the existing AARM notes for needed updates along with the actual text. Use the RM PDF to see how the changes look without all of the notes.

Action item on everyone: read sections 1, 10, and 11 and send comments to Randy and Pascal.

Randy asks about AARM notes. Pascal comments that the references to the AIs provide detailed explanations of the rules and design; we don't need to repeat that. Randy counters that it is often obvious that a brief note will help understanding the rules. He's looking at this in more detail than anyone else will, and it pays to let him help out future readers. Moreover, forcing readers to dig around in 20-page AIs to answer simple "why?" questions is not going to help. Randy also points out that many AIs (especially earlier ones) did not make an attempt to include needed AARM notes in the !wording (limited with, for example, has no notes).

But Randy should not spend his time condensing 20-page AIs into short notes. He should be conservative about putting in AARM notes other than those already included in AIs.

Corrections to Approved AIs

AI-212/04 Restrictions on configuration pragmas

Randy explains that "must" was changed to "shall" in the Implementation Permission, as this refers to legal code that has to be supported.

Tucker would like to drop "still" from this wording. Steve would prefer to leave the emphasis. Pascal notes that this is more informal than legality rules. Leave "still".

Approve change: 9-0-0.

AI-217-06/10 Limited With Clauses

Pascal noted that the rules didn't prohibit limited withing yourself. Added a bullet to 10.1.2(8) to make that illegal.

Randy further notes that “within an environment” was changed to “to an environment” in the added sentence of 10.1.4(3). He also moved the description of elaboration of limited views to dynamic semantics as all elaboration rules are dynamic semantics.

The last change is not reflected in the !wording section, only in the !corrigendum section. Fix that. Gary asks if the italics on “not” in 10.1.1(12) are appropriate? No: this is not a definition of “not”, and we don’t put emphasis in the text of the RM; remove those italics.

Approve changes: 9-0-0.

AI-224/10 pragma Unsuppress

Randy explains that the example of 11.5(32) was corrected not to use the “On” parameter (which was moved to Annex J) and an Unsuppress example was added.

Approve change: 9-0-0.

AI-231/10 Access-to-constant parameters and null-excluding subtypes

Randy says that a change to 11.5(12/1) was added to reflect Access_Check of null-excluding subtypes (and to remove the old text specifically talking about access parameters). This change is missing from the !wording. The paragraph number is 11.5(11/1). Randy is asked if there are other such changes needed; he thinks there may be, this was the only one that was obviously wrong.

Approve change: 8-0-1.

AI-252/08 Object.Operation notation

Randy explains that a legality rule was added, stating that a prefix used as an access parameter has to be aliased — 4.1.3(13.1/2). We want this to be a purely syntactic translation. Tucker notes that the new wording should be indented to match the rest.

Approve change: 9-0-0.

AI-254/09 Anonymous access to subprogram types

Gary had noted that the rules of 3.10.2(32) didn’t allow passing subprograms from a generic body as anonymous access-to-subprogram. This rule essentially stands in for a dynamic accessibility check, so we have to be careful. Certainly, an anonymous access-to-subprogram used as an access parameter is OK (the accessibility level is infinite); but when used as a component or a discriminant, its level is finite and we shouldn’t allow it. The wording was borrowed from another part of the standard.

Approve change: 9-0-0.

AI-287/07 Limited aggregates allowed

Randy explains that the rule preventing named objects from being returned does not belong here. In the absence of AI-318, the return-by-reference rules mandate a run-time check. And the change to those rules belong in AI-318, so it was moved there (along with a note).

Approve change: 8-0-1.

AI-344/05 Allow nested type extensions

Pascal noted that the prohibition against deriving from a generic formal type needs to include tagged types and tagged formals from generic packages. Otherwise, a formal package could be used to sidestep the restriction. The wording of 3.9.1(4/2) was changed to accomplish this.

“convuluted” should be spelled “convoluted”

Someone notes that the !wording section has both “replace 3.9(4)” and “modify 3.9(4)”. The latter is really only fixing an AARM note, but the reference is misleading. It should really say “modify 3.9(4.a-b)”.

Approve change: 9-0-0.

Detailed Review of Amendment AIs

AI-260-01/04 How to control the tag representation in a stream
AI-260-02/01 How to control the tag representation in a stream

Tucker: Add pragma Intrinsic to the generic. He would prefer that we define the dispatching generic formal, and suggests **is abstract** as the syntax. That would require being able to figure out the dispatching type from the signature; he recommends a rule like 3.9.2(12); that is, multiple specific tagged types would be disallowed in such a subprogram (the only remaining ones would be the controlling operands/result).

Should this generic be Pure? The instance isn’t automatically Pure, but this would allow Pure instances. That seems harmless.

John suggests making this a child of Ada.Tags. Should the parameter be named Tag? Tuck suggests Creation_Tag or New_Object_Tag. John would prefer The_Tag. No consensus for a change emerges.

Pascal notes that Tag_Read was a rather uncomfortable kludge; it feels good to replace it by something more general.

John notes a typo in the first paragraph of the discussion: a space in “scenar ios”.

The description of the operation of Generic_Dispatching_Constructor should be under the subheading Dynamic Semantics.

For the dispatching formal subprogram, **is abstract** replaces **is** Default, meaning you can’t specify a default for such a subprogram. Not a big deal. A dispatching formal subprogram is abstract in the generic body; thus it can only be used in a dispatching call (following from the existing rules). The actual subprogram has to be a primitive operation of the corresponding actual type, but it could be abstract.

Randy notes that the approval of AI-318 without restrictions means that we should have a limited version of the constructor generic (similar to T'Class'Input, which is defined for some limited types).

For alternative 1, No Action: 9-0-0.

For alternative 2, Approve intent of AI: 9-0-0.

Randy wonders if a Parent operation for Ada.Tags would be a good idea. There is no problem with the semantics (as the parent is unambiguously defined by the language) or the implementation (as all implementations already need the information to implement membership checks and conversion checks). The group agrees it would be worth looking at this operation. A No_Tag constant or some other way to identify a root tag would be needed.

AI-284/04 Nonreserved keywords

WG9 rejected this proposal in its recent vote.

No Action: 9-0-0.

AI-284-02/01 New reserved words

The only issue for us to decide on this AI is what to do with pragma Interface. WG9 has decided on the other issues.

We enumerate the choices on the whiteboard:

1) Disallow pragma Interface.
2) Put pragma Interface into:
    a) 2.x
    b) J.x
3) Allow any reserved word as a pragma.
4) Language-defined pragma may use reserved words
    a) Define Interface in Annex J.

One suggestion for wording for choice 2b is "Implementations shall support the use of interface as a pragma identifier." We don't want to define the semantics of this pragma, just require all implementations to at least recognize (and possibly ignore) it.

The subclause would be called "Pragma Interface" in the obsolescent features Annex.

Tucker notes that choice 4 would be weird as Interface would be the only one. And it certainly wouldn't save any wording (we'd still need a subclause in Annex J). No one is interested in choice 1 (too incompatible) or choice 3 (too weird — "pragma in", "pragma pragma", etc.)

Choice 2b is selected by a 9-0-0 vote.

Tucker says that this has to be written as a syntax rule, in English. Tucker will provide this wording.

Later, Tucker provides: "In addition to an identifier, the reserved word **interface** is allowed as a pragma name, to provide compatibility with a prior edition of the International Standard."

John wonders if the !discussion is too much of a case for reserved words. Pascal replies that it is good to show that the incompatibility was considered. John withdraws his comment.

Approve AI with changes: 9-0-0.

AI-285/09 Support for 16-bit and 32-bit characters

Pascal fixed various minor errors in the AI.

Remove the Open Issue from the AI. (Pascal just forgot to remove it); replace it by a citation of the SC22 resolution. Also should be an AARM note covering the response to the TR 10176 question. Pascal will write that.

John would like to thank to Terry Froggatt for his helpful comments on the AI.

Approve AI with changes: 8-0-1.

AI-302-03/06 Container library

Randy reads through the list of post-Palma changes to the proposal, asking for comments on each one. (Items not mentioned here had no discussion.) The list of post-Palma changes can be found in the !appendix to the AI; the numbers refer to the items in that list, which won't be repeated here.

17) Correct the spelling of "communitive" to "commutative" in AARM notes.

26) Checked_Updated_Element should raise Program_Error to make it clear that there is a bug. Program_Error means there is a bug; Constraint_Error means more of a limit error.

27) Pascal thinks that the important semantics for Swap is the same for each version (that is, the elements have swapped locations afterwards); the cursor results are a very minor difference. Matt argues that moving between containers is a problem if Swap has slightly different semantics. Pascal and Tucker suggest that cursors denote the containers; and we need consistency checks in the containers (make sure that cursors belong to the same container). Insert, Delete, etc. all have the same problem.

Tucker suggests that Swap could change the cursors so that it follows the elements (that is, they are in out). That seems like it would be harder to use (you couldn't use a function call or an in parameter in Swap).

Much more discussion ensues. Tucker claims that copying the elements is likely to be faster than relinking for indefinite list Swap; his position is that only definite lists would be helped by swapping the nodes. Randy thinks they'd be about the same speed at worst, but Matt agrees with Tucker. Erhard complains that swapping links makes writing code harder, but no one understands his concern.

We eventually conclude that we should go back to the Palma semantics for Swap (all containers), and then add Swap_Links to lists to give the fast version for definite lists. Swap does not pass the container; Swap_Links does.

Q1) Pascal suggests Find returns Last+1 and Reverse_Find returns First-1, because then the order of return is going in the correct direction.

We discuss the No_Index suggestion. A special value for Find failure feels better; and it is similar to the cursor version.

So we decided to add the following:

```
subtype Extended_Index is
   Index_Type'Base range Index_Type'First-1 .. Index_Type'Last;

No_Index : constant Extended_Index := Extended_Index'First;
```

Get rid of the Assert (the subtype now makes the check), change all uses of Index_Type'Base to Extended_Index. To_Index returns No_Index. Check all of the semantics of every operation to make sure that this doesn't break anything. Find_Index returns No_Index, making the default No_Index. Pascal would prefer that the parameter type remain Index_Type, and leave the defaults as is. Tucker agrees.

Q2) Pascal says Merge and Sort should have the same parameter profiles. There is general agreement on that. Pascal rather prefers option 3 (the formal type option), but he worries that if the user overrides some of the operations, invariants might not be maintained. Tucker thinks its better to write the explicit conversion, and then they are required to maintain the invariants. So drop the class-wide from Generic_Sort, and leave Generic_Merge (and Generic_Keys) alone.

Q3) The function should be Equivalent_Keys (note plural). Tucker says that we need to pass in the overriding "=" to avoid re-emergence. Matt asks what to do with the other functions in the specifications that are named Is_Equal_Key. They must change, too. This topic is not resolved now, we'll have to talk about Q6 first.

"Is_Disjoint" should be "Are_Disjoint" as it too is talking about two items. Tucker had suggested "Overlap"; Pascal and Gary agree. Change to "Overlap".

Q4) Tucker agrees with Matt's opinion (no exception). Pascal says Tucker is confused; it seems weird to allow setting the Capacity to a smaller value. Nick had suggested changing the name to Set_Minimum_Capacity. That seems to match the semantics better, and then removing the check is OK. Steve suggests “Ensure_Capacity”. That seems OK to most (Matt is uncomfortable, he prefers using Set_Something consistently).

Q5) Tucker insists on talking about Q6 first.

Q6) Tucker explains that Keys can be unequal and equivalent. A mapping is a set of Key→Element pairs. If you insert KeyX→ElementX, and there is an equivalent key in the map, ending up with KeyX→ElementY or KeyY→ElementX seems weird to him.

Tucker wants the key to be equal (in the sense of "=") after the insertion operation.

Pascal wonders why we need "=" to be different from Equivalent. It’s only used in "=" for maps. Why do we need "=" for comparing maps? Shouldn’t we use Equivalent for comparing maps? Steve gives a map of “A” vs. “a” — is that really different if Equivalent is case insensitive?

Tucker would like to look at things that are not part of the equivalent relationship. He wants "=" to tell you the same, which is stronger than Equivalent.

Matt notes that the Hashed_Set container (now dropped) was intended to fulfill this function. Tucker says that the key could be separate from the element and still have extra stuff.

Pascal thinks that it simplifies the model for the user to use key equivalence (i.e., Equivalent_Keys) and element equality (i.e., “=”) for map equality. Then we would drop the "=" on keys, and the default for Equivalent_Keys. There is general agreement on this change.

Q5) Tucker suggests that when the key and element are changed, both should be updated. The only time the implementation could avoid doing it is if the keys are equal (using the predefined "="). It does seem weird to not do this (imagine inserting “able” and getting “AbLe” back). We agree with this requirement.

We list the possible behaviors of Insert and Replace:

1) Insert raises exception if equivalent already in map;

2) Insert silently returns if equivalent already in map;

3) Insert replaces the entire pair if in map;

4) Replace if not there raises exception else replace key and element;

5) Replace key and element if there, otherwise do nothing;

6) Replace the pair, inserting if not there (same as #3)

We briefly discuss the five-parameter Insert that returns a cursor and inserted flag. It’s similar to a test-and-set; changes that happen after it is called don’t matter because the cursor is returned. It may be used to build all the others, but it’s sufficiently cumbersome that we want to provide convenience operations for the most frequent usages.

Tucker doesn’t think anyone is interested in #2 and #5. No one disagrees (or perhaps they’ve all dozed off after lunch).

He then goes on to claim that similar behavior needs to exist for Delete (either exception or ignore). He proposes that Insert and Delete raise exceptions (case #1 above); then we would add two new operations, Include and Exclude, which would not raise exceptions (case #3 above). Randy suggests that we probably need a Replace (case #4 above) for the first pair; otherwise there wouldn’t be a way to do a key replacement. Key and element are always set together unless the operation is clearly only on the element (like Replace_Element). The exception for Insert, Replace, and Delete is Constraint_Error.

We’re finally done with Randy’s list. We turn to other questions.

Matt asks about Insert for list. He thinks that we want to allow partial success if any exception is raised (that is, if multiple objects are inserted). The group agrees. Strike the last sentence of the wording. Add an AARM note that you have to leave the container in a consistent state when an exception is raised (if there isn't one already). That is a general requirement.

Matt asks whether map Update_Element and Query_Element routines pass the key to the Process routine. Yes, that should be done.

Pascal wonders what happens if you have two cursors from different containers in any operation. Someone says that it should be a bounded error; Pascal thinks that it should be detected, it is a safety issue. The same issue applies to a cursor and a container; they should be checked that they are the same when that matters.

Making this check implies that all cursors include some reference to their container (that's not currently true for List and Set). No one is worried about the size of a cursor; it's still going to be small.

Pascal wonders if we could get rid of the "extra" container parameters which require such checks. Matt explains that the model is that we have container parameters when the cardinality of the container changes, and otherwise we don't. It seems too late to change that decision, and changing it piecemeal is unlikely to give users a consistent experience. Tucker suggests that it would be possible to create an "unchecked" container, with the current interfaces, if the cost of the check becomes a real issue for users.

We agree that the check should raise Program_Error, and it is required anywhere it is needed.

Pascal notes that we have Hashed_Maps and Ordered_Sets. We are obviously missing two of them (Hashed_Sets and Ordered_Maps). Perhaps this is now stable enough to build these other two on top of what we have. These are useful (it's not easy to build a decent hash function in general, ordering is much easier). Moreover, it is currently hard to move from an Ordered implementation (using Ordered_Set) to a hashed one (using Hashed_Map) if performance concerns require that. Improving performance shouldn't require a complete rewrite.

Matt has created specifications for these two packages, but we'll need words. It would be nice if we could share a lot of the words between the related containers. Pascal will take a stab at defining the words needed.

Tucker wants an extra parameter to Hashed_Sets to define equivalent elements. (e.g., Equivalent); this often will not be "=".

Approve Intent of AI: 7-0-1.

AI-315/02 Full support for IEC 559:1989

Pascal (the AI's author) proposes No Action. There is a big implementation impact, and it's unclear that there is sufficient demand. Tucker wonders if we should encourage implementations to follow this model. Pascal says that there are problems with it; being able to query flags has a significant negative impact on instruction scheduling. So you'd have to do operations in order, which would negate 11.6. Tucker says this still a good place to start. Steve says that doing this would require a non-standard mode for an implementation.

Erhard would like to leave it open for the future, as this is a good idea that we couldn't finish satisfactorily. Some AIs have been around for a long time (e.g. AI-51). Pascal would not like to leave it open; he prefers a clean list of AIs. Other AIs (like preconditions) would be also fall into the category of good ideas that we didn't have sufficient time/effort/experience with.

John volunteers to put information about AIs that might be good ideas but we couldn't finish into the Rationale.

No Action: 8-0-1.

AI-318/03 Returning {limited} objects without copying

We decided last time to scrap this alternative in favor of alternative 2.

No Action: 9-0-0

AI-318-02/05 Limited and anonymous access types

This AI was reopened because there are additional compatibility problems with the AI that were not previously considered. In particular, there is a problem with stream attributes. Redefining T'Input for unconstrained limited would be illegal, but we still have calls to the predefined ones (which could exist for extensions where the base type did not have discriminants, while the extensions did have discriminants). T'Class'Input is always unconstrained, and thus always would be affected. We need to address this somehow.

Pascal and Randy suggest dropping the restriction to constrained returns, because users are going to complain about it (and doing so would eliminate the stream attribute problem). There is a discussion about the implementation cost and potential difficulties with this.

Tucker suggests adding an attribute Limited_Input that returned an access to item. That would have to be for all types to avoid contract problems. And it doesn't make much sense: T'Input is the best example of a constructor function in the language.

We enumerate the possible solutions to both problems (the second problem will be discussed in a moment):

1) Blow away 318.

2) Keep 318 with the restriction; would have to change the resolution of 195 to avoid interactions with streaming.

3) Keep the restriction, and allow return-by-reference of constrained and unconstrained. Probably a runtime check if a constrained object is initialized by a function call (or dynamic renaming). A function call as a component of an aggregate would have to involve a check.

4) Remove restriction:

    a) test in ACATS.

    b) don't test in ACATS, let market decide if/when implementers support it.

5) aliased return for return-by-reference.

6) implicit dereference of call.

7) "constructor" function.

Pascal would like to discuss the incompatibility problem with the rule that a return statement can only be an aggregate or function. Pascal says that this causes 50-60 ACATS tests to fail. That means there is a lot of code that would have to be changed. But how typical is the ACATS?

To address this incompatibility, we'd need to use (1), (3), (5), (6), or (7) above. Also, there is an interaction with extending limited vs. non-limited for interfaces (we dropped restrictions on interfaces based on the direction of this AI).

One opinion expressed is that the incompatibility isn't that bad, because limited as it is defined is rather useless. But if we're making it more useful, it would be best to get it right this time. We don't want to make arbitrary rules that people keep bumping into. Many people agree with this.

We agree to vote on (4a). Drop the restriction to constrained results.

Approve AI with changes: 5-0-4.

AI-355/05 Priority Specific Dispatching including Round Robin

Alan explains the changes.

Tucker points out various wording problems: the first sentence of the pragma description goes into static semantics; the part after the semicolon goes into legality rules. The first two paragraphs of the legality rules should be replaced by these simpler ones. Add an exception not allowing Non_Preemptive to be used in Priority_Specific; put that in the definition of Non_Preemptive. Post Compilation Rules is out of order, it goes just before Implementation Permissions. [Editor's note: The last change is wrong; clause 1.1.2 (which is definitive) shows Post Compilation Rules coming just before Dynamic Semantics. Thus, no change will be made.]

“Designated” implies access types. “Designates a task dispatching policy” should be “is a task dispatching policy”. “designated range” should be “specified range”. “The policy_identifier shall designate a task dispatching policy.” should be “The policy_identifier shall be the name of a task dispatching policy”.

The syntax of the declaration of the exception in Ada.Dispatching is incorrect.

The last sentence of !problem needs a verb. “This is a need…”.

D.2.4 should be D.2.5 (AI-298 defines D.2.4).

In (old) D.2.4, first sentence, add “the” in front of “package”.

In the last sentence of Static Semantics of D.2.4: “tasks with priority…” should be “each task with priority…”.

The 4th paragraph of Dynamic Semantics doesn’t parse. Drop the “either”. Same is true in the first paragraph of Dynamic Semantics. We really shouldn’t be talking about calls here, either. “A call of either Set_Quantum procedure” should be “The procedure Set_Quantum..”

Paragraph 5, Dynamic Semantics of D.2.4 should end with a colon.

In Dynamic Semantics for D.2.2, paragraph 4, 1st sentence; Tucker says that commas make it hard to read. Remove the two commas and the stuff between them. Same in D.2.4, 4th bullet of Dynamic Semantics.

D.2.4 Note, “may not” is bad news for ISO, use “might not”.

D.2.4, Dynamic Semantics, first bullet: Remove “set” from “set equal”.

3rd bullet: Change “follows that for” to “is the same as that for”. “When a task” should read “While a task”.

5th bullet: Change “when it executes” to “while it executes”. Actually, this just repeats the 6th bullet (and the 3rd bullet); get rid of it completely.

6th bullet: Remove first comma.

Bullets 4 and 6 repeat the text of about “given a budget…” from bullet 1. There is no need to repeat this, drop it from these bullets, just say that it goes to the “tail of the ready queue for its new priority level” and then bullet 1 kicks in. (An AARM note explaining this ramification would help.) Change first bullet to “added or moved to the tail”.

Documentation Requirement “range of quantums” should be “range of quanta”. In the second paragraph, replace “task’s budget” with “the budget of a task.”

D.2.4 Notes: delete “a call of” (two places).

In the example, add a hyphen so it reads”32-priority”.

Approve AI with changes: 9-0-0.

AI-357/05 Support for Deadlines and Earliest Deadline First Scheduling

Should this (and AI-355) be explicitly optional (as in D.11(10))? Pascal argues that 1.1.3(17) covers this, but that is rather dubious. Randy says that we want to warn users that this might not be available.

Add that text for EDF and for Priority_Specific_Dispatching in AI-355.

Since AI-355 is D.2.5, this should be D.2.6.

First line of the wording should say "the absolute deadline of a task". Change the second sentence to "A pragma is defined to assign an initial relative deadline to a task." Add an AARM note to point out that this is the only way to set the deadline during activation. The text should start with a description of what a deadline is, along with other introductory information. Probably the text should start with the second paragraph.

"designates" should be "is the name of".

First paragraph of Static Semantics, the policy name should be plural.

Paragraph immediately after the package: EDF does require Ceiling_Locking; this should remain no matter what the result of AI-333. This is a Post Compilation rule.

The next paragraph after the package is Dynamic Semantics. It should say "initial deadline Default_Deadline", and drop the mention of Set_Deadline (this can't affect the initial deadline).

Second legality rule cannot be a legality rule, as it could be in a separate unit. A post compilation check seems awful. So it needs to be a dynamic check. The range needs to be the specified range of priorities, not some subrange of it. (Otherwise, Low could be any priority.) It seems like a bug to set the priority that way; therefore it should raise Program_Error when the object is elaborated, or when an assignment to the Priority attribute of the object occurs.

First paragraph of Dynamic Semantics, the last sentence seems to repeat the text that we just moved here. Delete the extra text (the previous text that we were going to move). The second sentence has no verb; merge the first two sentences by writing ", where the call of … is made…".

In the second paragraph, change to "The function Get_Deadline…" and "The procedure Set_Deadline…".

In the third paragraph, drop "task's".

In the fourth paragraph, change to "The procedure Delay_Until…".

The fifth paragraph is another place where the ranges need "is specified".

The first bullet doesn't talk about modifications. This should be part of fifth paragraph. The last bullet also isn't about modifications, and should be a separate (non-bulleted) paragraph.

The second bullet should use nested bullets, not numbers. The sub-bullets should be separated by semicolons, add "and" to the end of first sub-bullet. The start text is talking about ready queues, and the end is talking about protected objects. The end should be talking about ready queues. Better, use "placed on the ready queue for the highest priority R, if any, such that" and "If no such priority R exists, then…" Also, name the task by writing "When the task T…" and replace "unblocked task" with "task T" in the sub-bullets.

In the third bullet, drop the parenthesis. Steve suggests merging with the previous bullet. Others think that would make it ugly. Pascal suggests "When the setting of the base priority of a running task takes effect, and the new base priority is in the range Low .. High, the task is added…".

In the fourth bullet, replace "determined by" with "of". Why is the second rule there? It seems to be obvious. Alan says that this overrides any rules from Ceiling_Locking.

The fifth bullet should not be a bullet. Change the second "a task" to "any task". Merge this with the next paragraph.

Erhard wonders what happens at a task dispatching point; it isn't clear which ready queue it goes on. It seems that we need a bullet about putting back in the ready queues. Similarly, what if the priority changed for a non-running task?

It seems like the three rules of bullet 2 should be described independently of the event that triggers it, then have a list of events that trigger those rules. Also, we don't seem to cover the case where a task has its deadline changed. It would seem that we should sort the ready queue in this case.

In the last paragraph, drop "and types" as there are no types in this package.

In the first note, change "are allocated" to "are specified to have". Remove the last sentence, we're not talking about other policies.

In the third note, change "may" to "can". Tucker suggests that this is should be an AARM note; it is not aimed at the user of the language.

Approve intent of AI: 7-0-2.

AI-359-01/03 Deferring Freezing of a Generic Instantiation
AI-359-02/02 Deferring Freezing of a Generic Instantiation
AI-359-03/01 Delayed completion of a generic instantiation

Pascal thinks that alternative 3 is too complex. Tucker notes that it needs to be a partial view, not a limited view. That certainly doesn't make it easier. The "customer", (Bob) didn't like either alternative 2 or alternative 3 as written.

Tucker asks to defer this to Saturday so he can present his improvement of alternative 3, as the mail discussion on it was private and not recorded.

At the crack of dawn on Saturday (OK, really the start of the meeting at 9 AM) Tucker explains his alternative proposal. He suggests piggybacking on the formal package rules:

```
generic
    type T1 (<>) is private;
    type T2 is new T1 with private;
    with package P is new G (T1, others => <>);
package G is ... end G;

package Inst is new G (T1 => A1, T2 => A2, P => AP);
```

The formal part is similar to the visible part of a package. The "completions" are the actuals. For a package, they are completed in the private part.

```
package P is
    type T1(<>) is private;
    type T2 is new T1 with private;
    with package P2 is new G (T1, others => <>);
private
    type T1 (...) is ...
    type T2...
    package P2 is new G (A, X, 3);
```

Types in the package would be "private". Items could be used in constructs that don't cause freezing.

Steve comments that this would introduce "deferred variables". Yes, but we already have them. They would be like a reference to a variable in the default expression when the declaration is separated from a representation clause on the object.

This idea is promising enough that Tucker is asked to write up a complete proposal and present it on Sunday (see the next item).

After discussing that proposal on Sunday, we all agree that the other alternatives are not interesting.

No action (on alternatives 1 to 3): 9-0-0.

AI-359-04/01 Abstract generic instantiations

On Sunday, Tucker presents his complete proposal. He says that a partial instance needs to be allowed in a private part (of course, it would need to be completed there). That's necessary to allow recursive data types.

Tucker is worried that his proposal doesn't scale well. He shows an example:

```
package P is
    type T is private;
    with package T_Sets is new Sets(T);
    with package T_Maps is new Maps(T);
private
    type T is record
        S : T_Sets.Set;
        M : T_Maps.Maps;
    end record;

    package T_Sets is new Sets(T);
        -- T freezes here, but T_Maps isn't full yet.
    package T_Maps is new Maps(T);
end P;
```

Tucker thought about deferring the freezing to fix this, but that brings up the problems we previously had. Pascal agrees that it doesn't seem worth pursuing if it doesn't compose.

Randy notes that his proposal (alternative 3) didn't have this problem because it required the use of **access**, and **access** doesn't freeze the designated type. So this does work in that case.

Pascal realizes that this example is in a state of sin, because it is depending on the implementation of Sets and Maps having a level of indirection. That is privacy breaking, of course; code should not make such assumptions. So users shouldn't do this; they should explicitly use accesses.

Pascal doesn't like the **with** in the syntax. He suggests that it is partial if it has a box. Randy objects, because you may only have a few parameters and you may need to give them all; the client of the package containing the partial instantiation may need that interface.

Randy notes that **with** is a noise word added to formal parts to avoid syntactic ambiguity. He often forgets to put it in (until the compiler complains). Many agree with this sentiment.

Randy's alternative used **limited**, but that would be inappropriate here because the view is not a limited view (it's a partial view).

We start listing various syntax ideas:

```
with package P is new G(T);
begin package P is new G(T);
package P is private new G(T);
package P is new private G(T);
interface package I is new G(T);
package I is interface G(T);
package I is interface new G(T);
package interface I is new G(T);
package P interface is new G(T);
package P is abstract new G(T);
package P is new G(T) with <>;
package P is new G(T, others => <>);
          -- Allowing others => <> not to match anything.
package P is new G(T); -- If T incomplete.
```

Randy objects to the last. Tucker notes that it would be odd to have to identical declarations that mean something different depending on invisible properties of the actuals and their location in the source code.

Erhard's suggestion of:

```
package P is abstract new G(T);
```

seems good, as the obvious terminology seems natural. The keywords should be in this order to match abstract type derivation (it would be confusing for it to be different). We take a straw vote to use this syntax: 8-0-1.

The wording would change "partial instance" to "abstract instance".

Pascal says that he couldn't find words in the AI to say that everything exported from the instance is a partial view. Steve asks if a type is declared in the specification (but not depending on formal parameters) could be static. This would seem to introduce a great deal of complexity, so it's better to say that everything is nonstatic (and noncompleted), heck, even named numbers.

Erhard asks about this example:

```
package P is
    type T is private;
    package T_Sets is abstract new Sets(T);
    package T_Maps is abstract new Maps(T_Sets.Sets);
private
    type T is record
        M : access T_Maps.Maps;
    end record;

    package T_Maps is new Maps(T_Sets.Sets);
        -- Bad, but don't do this, put it after the instance T_Sets.
    package T_Sets is new Sets(T);

end P;
```

Pascal says that a call is a freezing operation; but freezing a subprogram doesn't require a body. So, what about a call to a subprogram in the instance? That would be an access-before-elaboration error. Pascal would like this to be detected statically; there seems to be no reason to allow it. To do that, we would need some words.

Tucker suggests that we need at least: "The completion of the instantiation shall occur before any entity defined by the instantiation is frozen."

Pascal wonders if this works. He writes an example:

```
generic
package G is
    function F return Boolean;
    B : Boolean := F;
end G;

package I is abstract new G;
```

The declaration of I.B would freeze F. We certainly don't want that to happen. We need words like:

> An abstract instantiation does not cause freezing. Using the name of an abstract instantiation does not cause freezing.

Steve is thinking about generic children. Assuming that a generic child is withed, does the nested child generic exist in the abstract instance? The most logical answer is yes, but it would have to instantiated as an abstract instance. That also would be true for a nested generic unit. More words are needed to capture that.

Pascal asks to look at the example in the AI. Outside the package, you can reference the boxy parameters, but you don't know the values. That seems weird. Tucker says that these should have the same rules as formal packages; we don't want different rules. Erhard says that it seems odd that you can lose names when you give the actuals. This is not a new oddity, it was discussed in the context of formal packages (AI-317). We're not going to repeat that discussion.

Gary says that this is likely to require work and introduce bugs. He's not convinced it provides enough capability to be worth that work.

We all agree that the other alternatives are junk; this is the only one that we might want to pursue.

So is the problem worth solving? Randy says it is a lot of work. But users are going to run into it when they are using the container libraries. Erhard thinks that it might not be as bad to implement as described, because the rules are fairly clean (not a lot of special cases).

Pascal and Steve discuss their implementation. Randy describes his ideas (instance at the point of the abstract, code at the full). But Randy's implementation doesn't work for <>. Pascal thinks it could be done similarly to formal packages. Gary still thinks it's hard.

Steve asks if you have to generate any code at the point of abstract. The wording says that the actuals are evaluated. Tucker says no code is needed, because all of the actuals are static; there cannot be function calls, etc. as actuals because of the conformance rules.

Pascal points out that you don't want to freeze a deferred constant, which you would have to do to evaluate it. So this should say that the parameters aren't evaluated.

Approve intent of AI: 8-0-1.

AI-366/03 More liberal rule for Pure units

Erhard thinks that E.2.2(17/1) conflicts, as it says that Storage_Size is not defined. It probably should say Storage_Size = 0. This seems harmless, no one can be using Storage_Size on such a type.

Erhard does not think that "part" applies to a type, only to an object. We discuss this for a while. Tucker says that component probably has the same problem. "The terms part, component, and subcomponent can be applied to types meaning the part, component, or subcomponent of values or objects of the type." should be added to 3.2(6).

3.8(9) has "component" in italics; that's not a definition of "component". Fix that.

E.2.3(14.1) should say "available" instead of "user-defined" for limited types (but "user-specified" is correct for access types). Make these rules bullets. "Part" of access type includes the type itself, so get rid of the extra phrase (two places). Tucker would like to define "remote streamable". Pascal worries that it would need to be defined in 10.2.1, which is far away from the usages. Tucker suggests "externally streamable" to get away from Annex E connotations.

John suggests revising the title. At a minimum, there is more than one rule.

Erhard will update the wording.

AI-370/03 Environment variables

The AI was updated as discussed in Palma.

For Clear, the wording should say "all existing environment variables".

There is a typo in the Bounded Error text: "exist" should be "exists".

The Erroneous Execution wording should be rewritten as Nick Roberts suggested in his e-mail (see the Appendix of the AI), as Iterate is not generic.

Several people dislike Constraint_Error if there is no environment. Program_Error seems more appropriate if an environment is not supported.

Tucker would like the Bounded-Error to say that “one of the values is returned and that same value is returned in subsequent calls in the absence of changes to the environment”. We don’t want some random value to be returned, which the current wording allows. The group agrees with this change.

Steve would like Implementation Advice saying that this is not intended to buffer the OSes environment (changes should be reflected immediately). We also need a statement that it is implementation-defined what happens if the environment is accessed via other mechanisms (e.g., direct system calls).

The Implementation Permission seems to say that if you don’t have variables, you don’t have variables. Replace “does not support” by “supports”. The (existing) IA should be included in the permission. Tucker says that an implementation should either simulate an environment well or not at all.

Gary notes that “non-empty” should be “nonempty” to match other uses in the Standard. [Editor’s note: there are 4 such uses.]

“Concept of” seems weird. The text “concept of subprocesses” should be “supports subprocesses” and “concept of environment variables” should be “supports environment variables”.

John notes that the last sentence of the first paragraph of the discussion is wrong. Remove it. Erhard notes that the penultimate paragraph of the discussion talks about the Implementation Advice, change to Implementation Permission as we’ve moved all of the advice to the permission.

John notes that the last paragraph says “environment variable” should be “environment variables”. The second paragraph of the discussion is talking about the position stuff, which was eliminated. Get rid of it.

Approve Intent of AI: 8-0-1.

Pascal will update the AI.

AI-382/01 Current instance rule and anonymous access types

This allows the current type name to represent the type, not the current instance, in access_definitions. Otherwise, self-referencing anonymous access components can’t be declared.

Should this be allowed in access-to-subprogram? That seems OK. Must be careful not to allow too much, because this also occurs in task bodies, and we don’t want to go too far (that could be very incompatible). Wording of “a subtype_mark in an access_definition” is suggested. That is not good, it allows it in a conversion of a default expression. Steve says that would make T'(T) legal in some cases!

We eventually agree on the long-winded wording in the AI. Replace “general access type” by “access-to-object type”; add “a parameter of an access-to-subprogram” as another place it is allowed to be a type. Rewrite the text into bullets; possibly redo the outer bullets as well to make more sense.

The type should have at least the amount of info that you get from a limited view (or an incomplete type); so it should be known to be tagged. That means we need even more words here. (We definitely want 'Class to work in this case.) Try the wording “within the subtype_mark” (which allows T'Class as a subtype_mark).

What about discriminants? The type is not visible there, and we want it this way. (How could you declare such an object for a non-null access type? Where would you get the first object from?)

Approve Intent of AI: 9-0-0.

Pascal will update the AI.

Detailed Review of Regular AIs

AI-51/12 Size and Alignment clauses

13.3(25): “as at least as” → “at least as”.

Erhard wonders why this doesn’t say how small Size can be. That is covered by the existing standard text — in 13.3(55).

Erhard wonders what the largest Alignment of any record or array type is. That is the largest Alignment for any conceivable record or array type. That makes sense for Alignment, but that it would be goofy for Size. Drop record and array types from 13.3(56). [Editor’s Note: in the end we did not decide what the recommended level of support should be for record and array types; presumably this will have to be resolved by email.]

Tucker thinks that protected types and task types should be required to support sizes. Pascal and Randy can’t figure out why: size clauses exist for interfacing to hardware and (foreign) software; how many protected types are you going to find in hardware?

Approve AI with changes: 8-0-1.

AI-133/04 Controlling bit order

There is a typo in !question: “is is intended to solve” should be “is it intended to solve”.

Pascal explains that he did not do two things suggested at the Palma meeting that he felt didn’t make any sense. He didn’t change “machine scalar” to “machine size”, because it is a container, not a size. Also, he didn’t change the definition of the position attributes, because that only made sense when the attributes were specified (the semantics were in essence “return what the component clause says”, but what if there is no component clause?)

The position attributes depend on the existence of a representation clause. That’s ugly, but it doesn’t seem to be avoidable.

Steve asks if alignment needs to be included in the definition of machine scalars. Pascal replies that it doesn’t interact here.

Tucker would like a specification of R.C'Address in terms of R.C'Position when it is specified. That seems incompatible, and possibly it would point at the wrong byte for the hardware.

We need to allow sizes larger than the largest machine scalar for composite types. In that case, the first bit shall be 0 and the last bit must be a multiple of 8 - 1.

Tucker would like to use the words “The set of distinctly sized machine scalars is implementation-defined.” Insert that after 13.3(8).

Gary notes that 13.5.1(13) “rounding up the largest size” should say what it is rounding up to. The size of the “smallest machine scalar that is larger than last bit” is what it is rounding to.

Gary notes that in 13.5.1(17), “floating-point” should be “floating point”, and needs a comma following it.

In 13.5.2(3-4), “Otherwise” should be in lower case.

Kiyoshi notes a typo in the summary — “are interpreted are” should be “are interpreted as”.

Pascal will make the changes in time for editorial review.

Approve AI with changes: 8-0-1.

AI-158/03 Renamings of primitive of a class-wide generic actual type

Tucker thinks that the wording section is weird, since there is no problem with the renaming; it’s a weird instantiation. So, drop all of the 8.5.4 text except the last sentence.

Tucker claims that instantiating a generic formal derived type with a class-wide actual type was fully intended. The intent was that using the primitive operations would be dispatching. Steve says that the wording of 12.5(8) says primitive operations are matched, and there are no such operations for a class-wide. And that is the core of the problem — since we don’t know what the operations are, we certainly don’t know what they do. We probably need words to say that in this case, the inherited routines are dispatching.

Tucker would like to clean up this AI, and have a new one to discuss whether this capability should be allowed at all, and what the model would be. 12.5.1(21/1) would need to have a wrapper model that the calls would be dispatching for *all* of the inherited functions.

There is no reason for a separate AI; the meaning of the renames follows naturally once we know how the inherited subprograms work. Moreover, the discussion and possible solutions are strongly related.

Reclassify as not a pathology (the rename might be a pathology, but passing in a class-wide type isn’t) and assign to Tucker to try to fix the dynamic semantics.

AI-269/04 Generic formal objects can be static in the instance

Gary explains the changes.

4.9(38) should be plural (there are multiple rules). Perhaps it should be stated in terms of the “Implementation Requirements”. Tucker suggests that this is an expansion of the last sentence 4.9(38); it should just be part of it. That would make it less open-ended.

Steve suggests that we simply say that an expression in a generic body that depends on a formal is not static. We discussed that a lot in the past, and decided on these rules. Steve persists — why? We go back the minutes of previous meetings to see.

Gary describes the other issue. The wording for 4.9(35) was borrowed from elsewhere — 8.6(27). Using that wording avoids the need to define the base range for “any integer type” and universal_integer. We don’t want to depend on that wording, it’s scary.

Pascal suggests: “…and the expression is expected to be of a single specific type…”. Randy worries that the technical meaning of “single” applies here. We need something like “single” to prevent the rule from triggering in cases where “any integer type” is expected. The use of “single” does not worry others, so use that.

Steve notes another problem in 4.9(38); he would prefer “…the corresponding expression in the generic body is not static”, as the expression in the generic body is not the same as the one in the instance.

Approve AI with changes: 9-0-0.

AI-279/05 Tag read by T'Class'Input

Tucker would prefer that tags aren’t “elaborated”, but that they are “created”. So he thinks the term should be changed. Pascal and Randy disagree but lose out to many examples from the current standard.

Tucker notes that “identifed” is spelled wrong in 13.13.2(34).

Gary says that in the first paragraph of the summary, “in” should be deleted in “ a library level type in which has not yet”.

Pascal comments that in the AARM note in the wording the use of Tag_Read should be removed, as we killed it yesterday.

Erhard would like to promote the AARM Note into an official Note (for users) that erroneous execution can be avoided by using a prefix type that is library level. Word carefully.

Tucker asks that in Erroneous Execution, “provided to” be changed to “returned by Descendant_Tag”, because we don’t have to worry about Tag_Read.

The permission of 3.9(26) should extend to the returned tag from Internal_Tag and Descendant_Tag.

Approve AI with changes: 7-0-2.

AI-291/03 By-reference types and the recommended level of support for representation clauses

In 13.1(18.1/1), drop “the value for”.

Erhard asked about 13.1(24). This is a list of bullets, which should be noted. The third new bullet should not be a bullet, it goes after the end of the list of bullets. The fourth new bullet should be moved in front of the first new bullet.

Erhard wonders about 13.2(6). This gives permission to ignore alignment requirements when packing. Erhard still wants specified alignment to not be overriddable. 13.3(38) says this is intended. Steve notes that this would mean that giving a confirming representation clause has an effect; we’ve worked hard for this never to be true.

Tucker wants to drop “object of”. Steve disagrees; he says you don’t “allocate” a component of a type. Pascal suggests moving “object of” after the second “component”. Steve and Pascal are happy with that change. Tucker notes that “aliased” is redundant with “aliased part”; “is not aliased,” should be removed.

John notes that this AI talks about “representation clauses”, he thinks that we dropped that in favor of “aspect clauses”. There is disagreement on this point. But we should fix the subject; it should be “representation items” because we talked about pragma Pack (which is not a representation clause!). Similar changes need to be made in the question, summary, and discussion (but not in the wording).

Erhard wonders why 13.2(8) is deleted. Steve thinks it is just redundant. Pascal says the “need not” is very important. Do not delete this paragraph.

Erhard wonders about 13.3(18). It is replaced by text in 13.1(24). This should be reflected in the text (“Delete 13.3(18) as it is moved to 13.3(24)”).

Avoid “i.e.”; use “that is” instead. Needs to be a comma after “e.g.”; but use “for example,” instead. Gary says “non-zero” should be “nonzero” (there are other occurrences in the Standard). “Size_Clause” should be written “Size clause” (without the underscore). “a Pack pragma” should be “pragma Pack”.

13.3(25) should be written in the singular. Erhard would like Alignment to be in lower case; but that is the (ugly) style of existing text. Drop “type’s”, it’s redundant and not elegant.

Tucker comments that volatile is not necessary; volatile is either by-reference or by-copy. And by-copy isn’t an issue. Drop the “volatile”.

Someone wonders why examples are given in 13.3(28), there are none currently. Delete the parenthesized text. These rules are still odd. We really want the original rule, with just “is specified” dropped.

The last paragraph of S'Alignment has an example that is unnecessary. There is a Note covering this, anyway. Delete the parenthesized text.

Tucker doesn’t understand the purpose of the rule in 13.3(43). He claims that there isn’t any problem for composite types (any extra space is junk). Pascal points out 13.1(7), which says that all bits are all read/written. Tucker wonders why by-reference matters. Pascal says that the problem is real for elementary types. Change the wording to “aliased elementary objects”.

Tucker's reasoning applies to 13.3(56), as well. What about making it smaller? Erhard notes that 13.3(43) covers it for objects. But that doesn't apply to 13.3(56). You have view conversions in both directions. You could copy, but not if it is limited.

We turn to 13.3(72). This again is mainly about view conversions; the types don't necessarily have common ancestors. Conversions between unrelated types are covered by AI-246. So we only care about this on derived types. But this is a type-related attribute, and those can't be changed by 13.1(10) for "by-reference". Thus, "by-reference or" can be deleted.

Tucker wonders if saying something like 13.1(23) as a blanket rule would be a good idea.

Tucker should rewrite the AI with this in mind.

Approve Intent of AI: 2-0-7 (Tucker's intent.)

Since that is surely not a clear consensus, we continue to discuss the issue.

Pascal notes that he was happy to limit the requirements on implementers, and let the marketplace decide. John concurs with Pascal's summary; we're forcing too many requirements. Different sizes are different types; we shouldn't try to require larger objects.

Straw vote: Steve's model, which is to impose minimum requirements on the implementation: 6; Tucker's model: more clauses are portable, more requirements: 2; Abstain: 1.

Approve AI with changes: 4-2-3 (Tucker and Erhard prefer Tucker's model).

We still have no consensus. Steve and Tucker will work on this off-line in the hopes of coming up with something. We'll try to limit discussion on this one in Atlanta.

AI-333/04 Other locking policies can be used with FIFO_Within_Priorities

The permission should also apply to Non_Preemptive and to Round_Robin.

Pascal comments that the use of "Locking_Policy" is a typo. Randy says that this refers to the pragma, not the "locking policy", because we didn't define them this way. Several others disagree, and think using "locking policy" is good enough. Change to "locking policy" in the wording.

Approve AI with changes: 8-0-1.

AI-334/02 Is overriding of abstract equality required?

Pascal worries about the effect on AI-228, but there seems to be none. He eventually agrees.

Approve AI: 8-0-0.

AI-377/01 Naming of child packages

Steve explains how the problem could occur. You can get two I1.G2 in some units.

Tucker would prefer that withing both items in the same unit be illegal. That seems much better from a user perspective; there is no clear reason for preferring one over the other. All of the examples would be illegal.

Approve intent of AI: 8-0-0.

Steve will update the AI.

AI-384/01 Conversion rules should be symmetric

Tucker explains the AI by showing the example in the AI. Conversions are allowed only one way by 4.6(16), but 4.6(50) checks it both ways at runtime. There doesn't seem to be a reason for this difference.

John remarks that it is not a pathology, because he has an example in his book.

Erhard wonders why this does not allow non-statically matching conversions, which also could be checked at runtime? Ada 95 tried to make many runtime checks at compile-time, and introduced some incompatibilities in doing so. It wouldn't make sense for us to undo that. Tucker notes that in the case of constrained to constrained conversions, the check either must always fail or always succeed. That suggests that deferring the checks to the conversions is strange.

Approve AI: 9-0-0.

[Editor's note: the !corrigendum and !ACATS Test sections were missing, thus the AI needed to be updated anyway.]