Minutes of the 52nd ARG Meeting

18-19 October 2014

Portland, Oregon, USA

Attendees: Steve Baird, Randy Brukardt, Gary Dismukes, Erhard Ploedereder, Tucker Taft, Brad Moore, Jeff Cousins, John Barnes.

Observers: Steve Michell.

Meeting Summary

The meeting convened on Saturday, 18 October 2014 at 09:15 hours and adjourned at 17:50 hours on Sunday, 19 October 2014. The meeting was held at the Portland Marriott Downtown Waterfront. The meeting covered all of the normal AIs and some of the amendment AIs on the agenda.

AI Summary

The following AIs were approved with editorial changes:

AI12-0041-1/01 Type_Invariant'Class for interface types (6-0-2)
AI12-0090-1/04 Pre- and Postconditions and requeues (7-0-0)
AI12-0094-1/01 access_to_subprogram_definition should be a declarative region (7-0-1)
AI12-0103-1/02 Expression functions that are completions in package specifications (6-0-2)
AI12-0106-1/03 Write'Class aspect (7-0-1)
AI12-0109-1/01 Representation of untagged derived types (7-0-1)
AI12-0113-1/02 Class-wide preconditions and statically bound calls (5-0-3)
AI12-0117-1/02 Restriction No_Tasks_Unassigned_To_CPU (7-0-1)
AI12-0121-1/01 Stream-oriented aspects (7-0-1)
AI12-0124-1/02 Add Object'Image (8-0-0)
AI12-0130-1/00 All I/O packages should have Flush (7-0-1)
AI12-0131-1/01 Inherited Pre'Class when unspecified on initial subprograms (6-0-2)
AI12-0132-1/01 Freezing of renames-as-body (7-0-0)
AI12-0133-1/02 Type invariants and default initialized objects (8-0-0)
AI12-0134-1/01 Still more presentation errors in Ada 2012 (7-0-1)
AI12-0136-1/01 Language-defined packages and aspect Default_Storage_Pool (7-0-1)
AI12-0137-1/01 3.10.1 and class-wide types (7-0-1)

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

AI12-0003-1/05 Default storage pool for storage pools (7-0-0)
AI12-0038-1/05 Shared_Passive package restrictions (6-0-2)
AI12-0086-1/01 Aggregates and variant parts (7-1-0)
AI12-0129-1/01 Make protected objects more protecting (7-0-0)
AI12-0135-1/01 Enumeration types should be eligible for convention C (7-0-1)
AI12-0138-1/02 Iterators of formal derived types (7-0-1)

The following AIs were discussed and assigned to an editor:

AI12-0018-1/01 Entity versioning
AI12-0059-1/02 Object_Size attribute
AI12-0064-1/03 Aspect Nonblocking
AI12-0111-1/01 Tampering considered too expensive
AI12-0125-1/01 Add Object'Succ and 'Pred
AI12-0140-1/01 Access to unconstrained partial view when the full view is constrained

The following AIs were discussed and placed into hold status:

AI12-0063-1/01 No_Return functions (8-0-0)
AI12-0091-1/04 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions (8-0-0)
AI12-0123-1/01 Add 'Subtype attribute (8-0-0)

The following AI was discussed and voted No Action:

AI12-0108-1/01 Out-of-range static constants (8-0-0)

Detailed Minutes

Apologies

Tuillo Vardanega, Robert Dewar, and Alan Burns sent apologies for not being able to attend.

Previous Meeting Minutes

Randy had previously posted a second draft of the minutes with the various corrections that had been submitted.

Approve minutes: 8-0-0.

Date and Venue of the Next Meeting

Next meeting: June in Madrid, associated with the Ada-Europe conference. The conference organizers will give us the exact dates. [Appears to be June 26-28, 2015 – Editor.]

Do we need a winter meeting? Let's decide at the close of the meeting. IRTAW is too late (mid-April), too close to June. Many people want a Florida location if we have a meeting. On Sunday morning, Steve Michell notes that Greg Gicca (who is here) would be willing to help us set up a February meeting in St. Petersburg, if we decide to have a meeting.

On Sunday evening, we reconsidered the winter meeting question. We made so much progress here that we probably don't need one. So we'll just plan on a phone meeting or two.

We try to set a date for the first phone meeting. Tucker says that Tuesdays are bad for him. Preliminary date January 28, at noon eastern.

Thanks

Thanks to the editor (Randy Brukardt) for taking the minutes.

Thanks to SIGAda for the accommodations.

Thanks to the Rapporteur (Jeff Cousins) for running the meeting.

Amendment AIs to be added to the Corrigendum

Randy had noted that any Amendment AI that doesn't involve syntax could be included in the Corrigendum. We look at all of the Amendment AIs, and suggest that we consider the following AIs for inclusion:

AI12-0003-1, AI12-0041-1, AI12-0059-1, AI12-0086-1, AI12-0117-1, AI12-0124-1, AI12-0129-1, and AI12-0130-1.

These are all discussed individually later in the meeting (see below), along with several other Amendment AIs.

Old Action Items

Steve Baird has AI12-0016-1 (outline of paper) and AI12-0020-1 unfinished. Nothing on these have changed since last time, however AdaCore is planning to try to implement one of the AI12-0016-1 schemes.

Randy Brukardt notes that he didn't work on AI12-0112-1 as it is not intended for the Corrigendum and AdaCore decided not to fund his work on it at this time.

Ed Schonberg (AI12-0002-1, AI12-0111-1) isn't here to explain his sloth.

Tucker Taft has AI12-0058-1 (along with Ed and Van). Tucker forgot that he'd promised to work on wording changes for this AI. Tucker did AI12-0064-1 after the deadline, as Randy forgot to put it in his action item list.

John didn't do any work on the Corrigendum introduction, mainly because Randy didn't ask him.

Current Action Items

The combined unfinished old action items and new action items from the meeting are shown below. (* indicates an AI that we'd like to be able to put into the Corrigendum, ? indicates an AI that we'd put into the Corrigendum if it was finished in time, but that seems unlikely because of the amount of work remaining.)

Steve Baird:

- AI12-0016-1

- AI12-0020-1
- AI12-0129-1*
- AI12-0138-1*

John Barnes:

- Help Randy with the introduction to the upcoming Corrigendum.*

Randy Brukardt:

- AI12-0018-1
- AI12-0059-1?
- AI12-0112-1 (with Ed Schonberg)
- AI12-0125-1
- AI12-0140-1*
- Create AI to handle erroneous execution related to a bad Allocate_From_Subpool (see discussion of AI12-0136-1)*

Editorial changes only:

- AI12-0041-1*
- AI12-0090-1*
- AI12-0094-1*
- AI12-0103-1*
- AI12-0106-1*
- AI12-0108-1
- AI12-0109-1*
- AI12-0113-1*
- AI12-0117-1*
- AI12-0121-1*
- AI12-0124-1*
- AI12-0130-1*
- AI12-0131-1*
- AI12-0132-1*
- AI12-0133-1*
- AI12-0134-1*
- AI12-0136-1*
- AI12-0137-1*

Brad Moore:

- AI12-0003-1*

Erhard Ploedereder:

- Compile all of the examples in the Standard (see discussion of AI12-0134-1) and report on any difficulties.*
- If possible, acquire some examples where Ada containers are too slow compared to other containers (for AI12-0111-1?).

Ed Schonberg:

- AI12-0002-1?
- AI12-0058-1? (with Van Snyder and Tucker Taft)
- AI12-0112-1 (with Randy Brukardt)

Van Snyder:

- AI12-0058-1? (with Ed Schonberg and Tucker Taft)

Tucker Taft:

- AI12-0038-1*
- AI12-0058-1? (produce wording, with Ed Schonberg and Van Snyder)
- AI12-0064-1?
- AI12-0086-1*
- AI12-0111-1?
- AI12-0135-1*
- AI on class-wide type invariants and statically bound calls (see discussion of AI12-0113-1)?

Detailed Review

The minutes cover detailed review of Ada 2012 AIs (AI12s). The AI12s 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 working Ada 202x AARM, the number refers to the text in draft 4 of the Ada 202x AARM. Paragraph numbers in other drafts may vary. Other paragraph numbers come from the final Ada 2012 AARM; again the paragraph numbers in the many drafts may vary.

Detailed Review of Ada 2012 AIs

AI12-0003-1/05 Default storage pool for storage pools

The subject should say "Specifying the standard storage pool".

Since Standard could be an identifier, we need some preference rule. As this is an identifier specific to the pragma, it is not resolved.

Tucker would like there to be an indication of a preference for the identifier specific to the pragma.

Tucker says that another possibly would be to have no argument at all. Randy notes that would not work very well for the aspect Default_Storage_Pool.

Steve Baird suggests using <>. Someone objects that that is new syntax.

Tucker suggests Standard'Storage_Pool. Steve hates this, because it seems to denote a single object. Beyond that, it makes little sense to have an attribute that only allows a single package as a prefix (along with access types). That's the same resolution problem that we would have with using Standard.

Steve Baird says that the standard pool is a policy, not an object. He wants it to look like a policy, not like an individual object.

Straw vote:

Standard is specific to a pragma and no resolution; 4

Resolves to an object, else identifier specific to a pragma. 0

Resolves to package Standard; 3

Straw vote: standard'storage 3; standard as identifer 4

Tucker suggests that it is illegal if (without any context) if it (Standard) denotes anything other than package Standard. That's just a bare lookup – anything that is visible would trigger the error. Otherwise, it is treated as an identifier specific to the pragma/aspect.

Brad will attempt to write it up using this semantics. He also will add the wording needed for aspect Default_Storage_Pool.

Approve intent: 7-0-0.

AI12-0018-1/01 Entity versioning

Randy notes that he didn't propose the pragma Restriction identifier for Ada_95_Only and so on, which is necessary for this to make much sense.

Steve Baird thinks that this doesn't believe belong in the Standard, since we don't usually talk about Ada language version.

We agree that this doesn't belong in the Corrigendum, and maybe only as an implementation-defined thing.

Not in the corrigendum: 8-0-0.

AI12-0038-1/05 Shared_Passive package restrictions

Tucker explains his examples.

Erhard worries about transitive usage of this. For instance, put the record type in the Pure package. Looks like the wording doesn't work in that case.

Steve Baird worries about controlled types used in Pure and Shared_Passive packages. These have some global finalization effects. No one sees an obvious problem, but as this is a new capability in Ada 2012 (Controlled was preelaborated in Ada 2005), there is a possible issue.

We worry about a Pure package has limited private type with an access component, and a constructor. In that case, we'd have to break privacy to check.

Randy constructs the following example:

```
package P1
   with Pure is
   type Lim is limited private;
   function Construct (A : access Integer) return Lim;
   procedure Update (Obj : in out Lim; B : access Integer);
private
   type Pure_Acc is access all Integer
      with Storage_Size => 0;
   type Lim is record
      P : Pure_Acc;
   end record;
end P1;

package body P1 is
   function Construct (A : access Integer) return Lim is
   begin
      return (P => A);
   end Construct;

   procedure Update (Obj : in out Lim; B : access Integer) is
   begin
      Obj.P := B;
   end Update;
end P1;

package P2 is
   X : aliased Integer;
end P2;

with P1;
package P3
   with Shared_Passive is
   type Rec is record
      P : P1.Lim;
   end record;
   I : aliased Integer;
   V : Rec := (P => P1.Construct(I'Access));
end P3;

with P1, P2, P3;
```

```
package P4 is
   P1.Update(P3.V, P2.X'Access); -- ??
      -- Stores an access to an object declared in a normal package
      -- into an object declared in a Shared_Passive package.
end P4;
```

Tucker worries that an access value could be created earlier before any shared passive package is involved. Then the check is too late.

Tucker suggests a solution based on ghost types. That's too weird.

We need to ban access types that come from a pure package.

The conservative rule would make all private types illegal, which is too incompatible.

A privacy-breaking rule is suggested. Use of a type with buried access types is illegal.

Steve then asks about access-to-class-wide, for which extension components have this property. We think that's already illegal for shared passive packages.

The privacy breaking rule would make the component of type Rec in Tucker's example illegal.

Similarly in Randy's example (Lim has a hidden access type).

The AI goes back to Tucker for rewording.

Approve intent: 6-0-2.

AI12-0041-1/01 Type_Invariant'Class for interface types

In the proposal, change “combination” to “conjunction” to make it crystal clear how they combine.

7.3.2(1/3) needs to say “private type, private extension, or interface”.

Steve worries that a record extension that implements an interface would not check if the type is not in a package specification. That's OK, as type invariants don't make any sense on types that are visible – it doesn't make sense to just make checks at particular points since things can change anywhere.

Steve would prefer the “nice clean model” of always checking every invariant on every primitive operation of a descendant.

Tucker notes that an abstract private type with no components is essentially the same as an interface. So the checks should be done the same way.

Gary: Fix in example: Window.Get_Width and Window.Get_Height (only the current instance is visible in the invariant).

Change to Binding Interpretation.

Approve AI with changes: 6-0-2.

AI12-0059-1/02 Object_Size attribute

We have a brief discussion on this AI. Tucker suggests that Object_Size = 0 is defined to be the default, and it means that no size was specified. Randy will look at that as a possibility.

AI12-0063-1/01 No_Return functions

Such a function can be completed by an exception or by abort.

This seems to be much less important as we now have raise expressions; it's no longer necessary to write an exception raising function to get a particular exception to be raised by an expression evaluation.

Randy suggests voting this hold.

Hold AI: 8-0-0.

AI12-0064-1/03 Aspect Nonblocking

Not nonblocking would be an annoying double negative. We therefore want to use Potentially_Blocking as the name of the aspect.

Do we want this in the corrigendum? Randy says he does, but he didn't think we could get it done. Tucker notes that we need to put this on all of the language-defined subprograms. Randy notes that 9.5.1(18) already says which subprograms need Potentially_Blocking => True and which need Potentially_Blocking => False.

Steve Baird wonders if this is two valued (True or False) or three-valued (True, False, unspecified). Tucker says that he intended the first.

If this is compile-time enforcement, it's weird to put it into a Bounded Error section. Steve had only runtime checks.

We need to have it on generic formals. We want to statically check that the aspect doesn't lie. (We don't want to make such a check by default for 9.5.1(8), because that would be very incompatible. We should have a restriction that 9.5.1(8) is checked statically, that's even better than pragma Detect_Blocking).

Steve Baird asks about Finalization, storage pools, etc. These are not always known to the compiler. Requiring all extensions to be non-blocking would be rather incompatible.

The AI goes back to Tucker for a *real* wording proposal.

AI12-0086-1/01 Aggregates and variant parts

Tucker thinks the wording is insanely long for this change. Randy says that he tried to shorten Steve's wording but failed; everything Steve says seems to need to be there. Tucker is unconvinced.

Hand to Tucker to shorten wording. We'd like to have this in the Corrigendum if possible.

Approve intent of AI: 7-1-0.

John is against messing with the language definition for this problem.

AI12-0090-1/04 Pre- and Postconditions and requeues

Steve Baird notes that the goal is that the caller should be able to rely on the postconditions of an entry even if the implementation of the entry uses a requeue. Similarly, the implementation (body) of an entry should be able to rely on its preconditions no matter how it is invoked.

Typo in wording: Jeff “parmeter”, Erhard “the the”, Tucker “there shall be exists” should be “there shall exist”. 5/3 (4 th para): Missing comma after redundant text.

In para “If the requeue target”, “the requeue target is declared immediately with” (should be “within”).

The P1 and E1 and P2 and E2 are confusing. The wording is backwards compared to the question. Change all of the P1 to P2 and vice versa, as well as E1 to E2, etc.

Steve worries that the actual rule is backwards. E1 is the (original) entry, and E2 is the requeue target. So we want the rule to be reversed, and the numbers are right.

> For every [Redundant:specific or class-wide] postcondition expression P1 that applies to an entry E1, there shall exist a postcondition expression P2 that applies to the requeue target E2 of the innermost enclosing callable construct such that

The context here is missing, redo it again:

> If an accept_statement or entry_body for entry E1 contains a requeue, then for every [Redundant:specific or class-wide] postcondition expression P1 that applies to E1, there shall exist a postcondition expression P2 that applies to the requeue target E2 such that

Erhard suggests dropping the “enabled or not” (two places), as this is a Legality Rule, enabling is a dynamic construct.

Randy e-mails the wording changes to the group.

Steve Michell notes that one can use the entry family index in a postcondition. [This appears to be wrong, as an entry family index does not have a name in an entry_declaration – Editor.] Since you can internally requeue to a different entry family, that could cause the postcondition to be a lie. Do we care? (We're not trying to fix all such cases.)

Tucker suggests that the postcondition cannot depend on an entry index. So add that to the 'Old rule.

13.1.1(12/3) says that formal parameters are visible; that needs to include the entry family index. [Again, since it doesn't have a visible name, there's no way for it to be visible. 13.1.1(12/3) is OK as is. - Editor.]

> Given a requeue_statement where the innermost enclosing callable construct is for an entry E1, for every [Redundant:specific or class-wide] postcondition expression P1 that applies to E1, there shall exist a postcondition expression P2 that applies to the requeue target E2 such that

There is a discussion about whether the wording works if the target is parameterless. The conclusion is that it does because it starts with P2 (which may have no parameters), not P1.

> The requeue target shall not have an applicable specific or class-wide postcondition which includes an Old attribute_reference or a use of an entry index.

Approve AI with changes: 7-0-0.

AI12-0091-1/04 Add procedure Sin_Cos to Ada.Numerics.Generic_Elementary_Functions

Randy notes that this was originally billed as not requiring any wording changes, but now we have 10 paragraphs that are changed. It no longer seems so trivial.

Straw pool: For: 0; Against: 1; Abstain: 7.

Tucker notes that many C implementations have it, but it's never made it to the C standard.

Hold AI: 8-0-0.

AI12-0094-1/01 access_to_subprogram_definition should be a declarative region

Drop the 7.1 in front of the bullet, and make the editorial change noted in the editorial note.

Add this after 8.1(2), rather than after 8.1(6).

Actually, access_to_subprogram_definition is a stand-alone type declaration. So we don't need a rule for it. We need this to cover access_definition, not access_to_subprogram_definition.

That should be changed everywhere in the AI.

Approve AI with changes: 7-0-1.

AI12-0103-1/02 Expression functions that are completions in package specifications

Tucker would like this to say proper_body, body_stub, or entry_body.

3.f should drop the first sentence (since it's clear from the wording).

The renames-as-body wording (AI12-0132-1) should immediately follow this one.

- At the occurrence of an expression_function_declaration that is a completion, the expression of the expression function causes freezing.

Randy notes that he put this after 13.14(5/3) to keep all of the not-quite-bodies together. That is agreed on. (So in AI12-0132-1, move the change after 13.14(5/3)).

AARM 13.14(3.g):

> Note that the rule about [bodies]{proper bodies} being freezing only applies in declarative_parts. All of the kinds of bodies (see 3.11.1 - keep in mind the difference from "body"s) that are allowed in a package specification have their own freezing rules{, so they don't need to be covered by the above rule}.

Approve AI with changes: 6-0-2.

AI12-0106-1/03 Write'Class aspect

Tuck defined the term “class-wide aspect”, and the “unless specified otherwise”.

Randy notes that 13.1.1(36/3) already covers the “unless specified otherwise”. Tucker thinks it's good to be explicit (and no one else agrees with Randy that the text is unnecessary).

The cross-reference should be to AI12-0121-1.

Approve AI with change: 7-0-1.

AI12-0108-1/01 Out-of-range static constants

Tucker asks where this came from. It came out of a Steve Baird discussion, it wasn't from a user.

Tucker thinks we shouldn't force any compiler to change behavior here. Thus we don't want to decide the actual question. Classify this as a pathology, and we won't decide anything. Thus the wording stays the same.

Change classification to Pathology, then No Action: 8-0-0.

AI12-0109-1/01 Representation of untagged derived types

Tucker suggests dropping the second bullet, as it is incompatible, and while the case is dangerous, it is not worth handling. Besides, it provides a work-around for 13.1(10/3), and probably some people have found it.

So the addition to 13.1(10/3) is:

> Similarly, it is illegal to specify a nonconfirming type-related representation aspect for an untagged by-reference type after one or more types have been derived from it.

For question 2, Randy notes that the literal wording has the effect that we want, so he just recommends adding an AARM note that hand-stands are needed.

Dump the parenthetical remark (we believe...) from the AARM note for 15.1.b. Modify the remainder:

> If {an}[the] aspect was specified by an aspect_specification and the parent type has not yet been frozen, then {the} inherited aspect will not yet have been resolved and evaluated.

First paragraph of the discussion, insert “an” before “inherited primitive subprogram”.

Drop “Rather than pick a fight with them...”

There is clearly a possibility of anomolies given that the aspect has neither (spelling of anomalies).

Approve AI with changes: 7-0-1.

AI12-0111-1/01 Tampering considered too expensive

Tucker wants to know about the known overhead problem. Randy says that it mainly comes from A.18.2(148.17/3) and similar rules; the finalization overhead on a common operation is too much.

Tucker wonders if the overhead can be eliminated when one is in an iterator and tampering is already set. Randy notes that it certainly can be eliminated if the life of the reference object is short enough that no calls are made during its lifetime.

Tucker and Randy go into a long discussion about possible changes to the rules. Randy notes that tampering with cursors (as an iterator uses) and tampering with elements (which Reference, etc. use) are different. Tucker suggests that the rule would just be tampering with cursors for Reference/Constant_Reference other than in indefinite containers. That already applies during the entire loop, so only a single state change would be needed, eliminating most of the expense. [Unfortunately, I didn't record why he thought that eliminating the extra protection of “tampering with elements” would be OK for a loop – Editor.] **Of** iterators in particular can be converted automatically because the uses of Reference are implicit in the iterator.

Tucker will try to write up this idea. He also will try to convince Ed (or someone else) to do some performance tests and/or implementation prototyping.

Erhard says that he had a student project that was 10 times faster on a JVM versus Ada. We would like to see some such examples so we can see if we can increase the performance.

Keep alive: 8-0-0.

AI12-0113-1/02 Class-wide preconditions and statically bound calls

Tucker explains that the model has been changed to a formal derived type model.

6.1.1(18/3):

> ...if it {is} not ab{s}tract and the...

The fix to the original bug is that we use the "controlling tag" to determine which body is called for any primitive operations.

Tucker is wondering what happens if you reference Obj'Tag in Pre'Class. This attribute only works on class-wide objects (which these aren't), and converting to T'Class puts us into the dynamic realm. So all is OK.

Tucker says type invariants don't have this problem. Randy refers to 7.3.2(5/3) which says that Type_Invariant'Class has T'Class. Tucker is not happy, but is convinced. He will take the Type_Invariant case in a new AI (split from this one).

Randy will take a stab at adding wording for class-wide postconditions. Here it is:

Modify 6.1.1(37/3):

> For any subprogram or entry call {S} (including dispatching calls), the checks that are performed to verify specific precondition expressions and specific and class-wide postcondition expressions are determined by those for the subprogram or entry actually invoked. Note that the class-wide postcondition expressions verified by the postcondition check that is part of a call on a primitive subprogram of type *T* includes all class-wide postcondition expressions originating in any progenitor of *T*[AARM Redundant:, even if the primitive subprogram called is inherited from a type *T1* and some of the postcondition expressions do not apply to the corresponding primitive subprogram of *T1*]. {Any operations within a class-wide postcondition expression that were resolved as primitive operations of the (notional) formal derived type NT, are in the evaluation of the postcondition bound to the corresponding operations of the type identified by the controlling tag of the call on S. [AARM Redundant: This applies to both dispatching and non-dispatching calls on S.]}

In 6.1.1(7/3): "though it had a (notional) type NT that is a [visible] formal derived type whose ancestor type is T{, with directly visible primitive operations}"

Approve AI with changes: 5-0-3.

AI12-0117-1/02 Restriction No_Tasks_Unassigned_To_CPU

The "this ensures" paragraph only applies to Ravenscar, and as such should probably just be dropped.

Next paragraph: "compilation" to "complication".

We should have a dynamic check, so this is a complete solution.

So add to the end of the restriction:

> If aspect CPU is specified (dynamically) to the value Not_A_Specific_CPU, Program_Error is raised.
>
> AARM Ramification: If this restriction is used in a context for which restriction No_Dynamic_CPU_Assignment is in effect, then no runtime check is needed when specifying the CPU aspect.

What about Set_CPU? And Delay_Until_And_Set_CPU? Those also should be checked.

> If Set_CPU or Delay_Until_And_Set_CPU are called with the CPU parameter equal to Not_A_Specific_CPU, then Program_Error is raised.

Rewrite the entire thing:

> The CPU aspect is specified for the environment task. No CPU aspect is specified to be statically equal to Not_A_Specific_CPU. If aspect CPU is specified (dynamically) to the value Not_A_Specific_CPU, then Program_Error is raised. If Set_CPU or Delay_Until_And_Set_CPU are called with the CPU parameter equal to Not_A_Specific_CPU, then Program_Error is raised.
>
> AARM Ramification: If this restriction is used in a context for which restriction No_Dynamic_CPU_Assignment is in effect, then no runtime check is needed when specifying the CPU aspect. If the restriction is used with the Ravenscar profile, no runtime checks are needed.

Convert this to a Binding Interpretation.

Approve AI with changes: 7-0-1.

AI12-0121-1/01 Stream-oriented aspects

Steve Baird: Extra that: AARM Proof: 13.1.1 says that all operational attributes [that] can be specified with an aspect_specification.

Tucker notes that the old wording says “may” and the new wording says “can”. Both should be “may” in 13.13.2(38/2).

In the summary replace the second sentence, “Class-wide attributes for interfaces can be specified as any nonabstract subprogram.”

In the real wording, replace “stream attribute” with “stream-oriented attribute”.

Jeff notes that stream-oriented attributes aren't in the index.

Approve AI with changes: 7-0-1.

AI12-0123-1/01 Add 'Subtype attribute

The problem is not at all compelling. AI12-0124-1 directly addresses the example given in the problem statement, so it's even less compelling with the Corrigendum features in place.

Moreover, there are many pitfalls.

Hold AI: 8-0-0.

AI12-0124-1/02 Add Object'Image

The wording should start with “X'Image denotes...” to be consistent with the other attributes.

Put each after the matching subtype version.

> X'Image denotes the result of calling function S'Image with *Arg* being X, where S is the nominal subtype of X.

Change to Binding Interpretation.

Approve AI with changes: 8-0-0.

AI12-0125-1/01 Add Object'Succ and 'Pred

Erhard wonders whether we should make all of the attributes that might make sense using an object prefix. It only makes sense for attributes that have one argument of the subtype. For instance, Value doesn't work because the parameter (which would be the prefix) is String, not the result type. So it only makes sense for Pos, Succ, and Pred.

Tucker thinks that as procedures these should be Inc and Dec. And then as an amendment for future.

Not in corrigendum: 7-0-1.

AI12-0129-1/01 Make protected objects more protecting

Protected functions/procedures are not potentially blocking (formally), that's not changing here. All this does is revoke the permission to execute functions in parallel.

Steve wonders if callers need to know about this aspect (as in synchronized interfaces). The callee does all of the locking (at least officially).

Tucker suggests that we want a Not withstanding rule as an Implementation Requirement. He suggests that we might even consider putting it into Systems Programming Annex. Probably as C.8, “protected object locking”.

Steve Baird will take this AI and produce wording.

This is a Boolean aspect Exclusive_Functions. “Mutually_Exclusive_Functions” would be better, but it's awfully long. So just follow the proposal.

Can this be specified on a derived type? No, that sounds like work (we'd need a new body with different locking). So don't allow on a derived type.

Tucker suggests that it is an aspect of a protected unit. (Like Pure.)

Approve intent: 7-0-0.

AI12-0130-1/00 All I/O packages should have Flush

We spend a lot of time looking at the meaning of the various I/O packages, in particular if buffering is allowed. All of the Read and Write routines only talk about the internal file. So it appears that buffering is allowed (at least in the underlying OS). So it makes sense to have Flush for all packages.

Jeff suggests defining Flush in A.8.2 as it currently is in A.12.1, and then referring to the A.8.2 version in A.12.1. That makes a lot of sense.

The Text_IO version has to stay where it is as it refers to Current_Output.

Make this a Binding Interpretation, and trust the editor to get it right.

Approve AI with changes: 7-0-1.

AI12-0131-1/01 Inherited Pre'Class when unspecified on initial subprograms

Jeff: “corrsponding” in the summary. 3rd paragraph of question: “evaluared”.

In second line of the discussion, “the the”.

Pre'Class is not defined to be True if anything is inherited (see 6.1.1(3/3)). So we don't need the extra wording in the Dynamic Semantics.

> If a Pre'Class or Post'Class aspect is specified for a primitive subprogram of a tagged type T, {or such an aspect defaults to True}, then the associated expression also applies to the corresponding primitive subprogram of each descendant of T.

Typo in discussion: “substitutability” is missing a 't'.

Steve notes that the Legality Rule should have the generic boilerplate.

Drop the editor's note (we've covered that).

The “optional” legality rule is required. Otherwise, we would have an inconsistency rather than an incompatibility.

Drop the paragraph about possibly making it a suppressible error, and all of the wording about it being optional.

Update the ACATS test section to reflect the legality rule.

Discussion: Thus we say that [a] overriding a subprogram that has no Pre'Class specified {for any ancestor} causes the new subprogram to inherit a Pre'Class of True.

Approve AI with changes: 6-0-2.

AI12-0132-1/01 Freezing of renames-as-body

“a example” in the question.

The wording needs to start with the “At the place”. No, this is directly a freezing thing, and should go higher.

Put this after 13.14(3/3):

> A renames-as-body whose name denotes an expression function causes freezing of the expression of the expression function.

Note that 13.14(10.2/3) is also out of place, but we surely aren't going to change that in this AI.

Gary says it belongs in the list of bullets 13.14(4-7.2). Tucker says after 13.14(6). [Later, we moved it to after 13.14(5/3), see the discussion of AI12-0103-1 – this AI was discussed first.]

- At the occurrence of a renames-as-body whose name denotes an expression function, the expression of the expression function causes freezing.

In the discussion, change “may be” to something more positive (we think the problem didn't occur in Ada 95).

Drop the sentence about convention.

Approve AI with changes: 7-0-0.

AI12-0133-1/02 Type invariants and default initialized objects

Default initialization is not “visible” if the partial view has unknown discriminants, so a check is not needed nor expected. So we ought to except that from the wording.

So the proposed wording is:

> Modify 7.3.2(10/3):
> After successful [default] initialization of an object of type T that{ is initialized by default (see 3.3.1)}, unless the partial view of T has unknown discriminants, the check is performed on the new object;
>
> AARM Reason: If the type has unknown discriminants, no client of the package can declare a default-initialized object. As such, no invariant check should be needed, as all uses are inside the package.

We're not going to do anything for uses inside the package for types that don't have unknown discriminants, as such types can be default-initialized both inside and outside of the package, and they ought to work the same in both places. As such, in a correct program, the check inside the unit cannot fail.

Randy will rewrite the discussion.

There is dislike of the proposed wording. Try instead:

> After successful initialization of an object of type T, if the object is initialized by default (see 3.3.1) the check is performed on the new object unless the partial view of T has unknown discriminants,

Nope. Take 3:

> After successful initialization of an object of type T by default (see 3.3.1), the check is performed on the new object unless the partial view of T has unknown discriminants,
>
> AARM Reason: If the type has unknown discriminants, no client of the package can declare a default-initialized object. Therefore, no invariant check is needed, as all default initialized objects are necessarily inside the package.

There ought to be an ACATS test specifically to check that a type with unknown discriminants does not get an invariant check on default initialization.

Change the comma at the end of the wording to a semicolon.

Approve AI with changes: 8-0-0.

AI12-0134-1/01 Still more presentation errors in Ada 2012

The 'M' in More should be in lower case in the subject.

Tucker says the best fix is to allow arbitrary expressions of discriminants. Steve notes that the problem is that we don't want to evaluate these such that different objects get different answers for the same discriminant value (as in D+Func, where D is a discriminant and Func is a function call) [Or worse, the same object at different times even when the discriminant didn't change.] We'd need a fairly complex rule to only allow cases that are safe (something with complexity similar to “predicate static”. No one is that excited about that.

Erhard would like someone on the ARG to compile all of the examples. Seems like a good idea, he is given an action item to do that.

Change the subject to fix the example, since there is only one presentation error in this AI.

Approve AI with changes: 7-0-1.

AI12-0135-1/01 Enumeration types should be eligible for convention C

C doesn't define the size of enumerations. But that's OK, we certainly want the Ada compiler to match the target compiler.

Steve Baird wonders what the corresponding type in C is for a record type. (For instance.) We have rules for parameter passing but nothing to define corresponding. This idea doesn't get much traction.

B.1 should say that enumeration types are eligible for convention-C.

Should we allow other types? (Modular, fixed, float are discussed). Most of these have no direct counterpart on the C-side. Float is already handled in the package Interfaces.C. So don't go further.

This change seems to imply support for Fortran and Cobol as well. That's OK if there is no corresponding type, it doesn't mean anything, and otherwise, it should work.

What about enumeration representation clauses? C has something similar, so it makes sense to allow them. So no special wording is needed for that.

Tucker will take this AI and create wording.

Approve intent of AI: 7-0-1.

AI12-0136-1/01 Language-defined packages and aspect Default_Storage_Pool

Randy explains that the problem is what (if anything) the user can rely on.

Tucker would like it to be implementation-defined whether a language-defined generic unit does allocations from the default storage pool.

So add after A(4):

“might or might not”

Tucker suddenly thinks that is weird. So he suggests putting it after 13.11(20).

> It is implementation-defined whether a language-defined unit uses the default storage pool for ...

Steve Baird says that it makes sense at the end of 13.11.3(5/3):

> The effect of specifying this aspect on an instance of a language-defined generic unit is implementation-defined.

Tucker believes that there is no new problem for use of a pool in a language-defined package; 13.11(21) seems sufficient.

Randy is directed to create a new AI to handle the erroneous execution issue for Allocate_From_Subpool. Probably should just say that the same requirements as outlined in 13.11 need to be followed, otherwise execution is erroneous.

Randy will rewrite the discussion to match this conclusion.

Approve AI with changes: 7-0-1.

AI12-0137-1/01 3.10.1 and class-wide types

“ apply to” in the summary “applies to”.

Tucker suggests “given an access type whose designated type is T or T'Class, where T is an incomplete view...” in paragraph 2.2. Also “...has {an}[this] incomplete view except when:”.

Then at the end paragraph 2.6 “view of T or T'Class”.

Randy notes that there is no incomplete view of T'Class.

Tucker thinks that we need some rule after paragraph 2.1 instead of the above.

Steve Baird suggests “For an incomplete tagged type, the class-wide type is also incomplete.”

Tucker suggests adding before the redundant part of 2.1:

“If T denotes a tagged incomplete view, then T'Class denotes a tagged incomplete view.”

In the question, “a class-wide type has no {explicit} declaration...”

The editor will redo the discussion.

Approve AI with changes: 7-0-1.

AI12-0138-1/01 Iterators of formal derived types

Tucker would prefer that the term was “immutable”.

We don't want to make inheriting discriminants illegal. We shouldn't have to say anything, because the discriminant still exists.

Wording: “the the”.

> The aspects Implicit_Dereference, Constant_Indexing, and Variable_Indexing are similarly immutable.

The Boolean aspects should not be involved, they don't have this problem (they already have rules to prevent it). Leave 13.1.1(34/3) alone.

> In some cases (see below), an aspect of a type is defined to be *immutable*. If an aspect of the parent type in a derived_type_definition is immutable, then
> - that aspect of the derived type is immutable; and
> - any specification of that aspect for the derived type shall be confirming.
>
> In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.
>
> Similarly for the aspects Implicit_Dereference, Constant_Indexing, Variable_Indexing, and Default_Iterator is immutable.

Ugh. Steve will try to reword this overnight. We definitely have to be careful so that the aspect can be specified for a non-root type if there is no specified value beforehand.

AI12-0138-1/02 Iterators of formal derived types

Steve sent the following wording overnight:

> Insert after 13.1.1(34/3):
> Certain type-related aspects are defined to be *immutable*.
>
> If an immutable aspect is directly specified for a type *T*, then that aspect shall not be directly specified for any other descendant of *T*.
>
> [Alternatively:
> If an immutable aspect is directly specified for a type *T*, then any explicit specification of that aspect for any other descendant of *T* shall be confirming.]
>
> In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.
>
> The Default_Iterator. Implicit_Dereference, Constant_Indexing, and Variable_Indexing aspects are defined to be immutable.
>
> ----
>
> In 4.1.6, Replace
>
> The Constant_Indexing or Variable_Indexing aspect shall not be specified:
> - on a derived type if the parent type has the corresponding aspect specified or inherited; or
> - on a full_type_declaration if the type has a tagged partial view.
>
> with
>
> The Constant_Indexing or Variable_Indexing aspect shall not be specified on a full_type_declaration if the type has a tagged partial view.
>
> ----
>
> Add at the end of the Legality Rules section of 5.5.1:

> The Default_Iterator or Iterator_Element aspect shall not be specified on a full_type_declaration if the type has a tagged partial view.
>
> ---
>
> Add to the (currently nonexistent) Legality Rules section of 4.1.5:
>
> The Implicit_Dereference aspect shall not be specified on a full_type_declaration if the type has a discriminated partial view.

"confirming" is not defined for operational aspects. So we don't want to use that in wording, but something like that would be nice.

Iterator_Element should be included in the list of aspects.

The list of aspects is weird, that should be mentioned for each aspect at their place of declaration. The existing rule should be left but put in square brackets.

So add immediately ahead of the text that we're already adding about partial view:

"The <blah> aspect is immutable (see 13.1.1)." (or the plural form, "The <blah> and <bletch> aspects are immutable.").

In the 13.1.1(34/3) redundant list, drop "defined to be", just use "are immutable".

Typo: period in that list rather than comma.

Very last paragraph of this wording, "Implict_Dereference" aspect.

4.1.5 Legality Rules:

Needs to say "discriminated or tagged", as it would be possible to add discriminants to an extension and then specify Implicit_Dereference. The type then could have two implicit_dereferences in some visibility locations, which would clearly be madness.

The main rule is:

> If an immutable aspect is directly specified for a type *T*, then any explicit specification of that aspect for any other descendant of *T* shall be the same name or denote the corresponding entity for the descendant type.

All of these aspects are already defined to inherit things. So we could use that:

> If an immutable aspect is directly specified for a type *T*, then any explicit specification of that aspect for any other descendant of *T* shall denote the same entities as the inherited aspect.

That makes better sense, it's more general. But it won't work for values, we won't worry about that as we don't have any such aspects now.

So the total change:

> Insert after 13.1.1(34/3):
> Certain type-related aspects are defined to be *immutable*.
>
> If an immutable aspect is directly specified for a type *T*, then any explicit specification of that aspect for any other descendant of *T* shall denote the same entities as the inherited aspect.
>
> In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.
>
> The Default_Iterator, Iterator_Element, Implicit_Dereference, Constant_Indexing, and Variable_Indexing aspects are defined to be immutable.
>
> For 4.1.6:

The Constant_Indexing and Variable_Indexing aspects are immutable (see 13.1.1). The Constant_Indexing or Variable_Indexing aspect shall not be specified on a full_type_declaration if the type has a tagged partial view.

For 4.1.5:

The Implicit_Dereference aspect is immutable (see 13.1.1). The Implicit_Dereference aspect shall not be specified on a full_type_declaration if the type has a discriminated or tagged partial view.

For 5.5.1:

The Default_Iterator and Iterator_Element aspects are immutable (see 13.1.1). The Default_Iterator or Iterator_Element aspect shall not be specified on a full_type_declaration if the type has a tagged partial view.

Changes to the above:

13.1.1: "...then an[y] explicit specification..."

13.1.1: "Redundant[The Default_Iterator, Iterator_Element, Implicit_Dereference, Constant_Indexing, and Variable_Indexing aspects are defined to be immutable.]"

Problem:

```
      type Parent is tagged null record;
      type Child (D : access Integer) is new Parent with null record
          with Implicit_Dereference => D;

      type Priv is Parent with private;
   private
      type Priv is Child with null record;
         -- Priv has hidden Implicit_Dereference.
```

Priv does not visibly have Implicit_Dereference.

This happens for all of the cases, so we have to replace all of the wording.

The full declaration of the type shall not have a <blah> aspect if it has tagged partial view.

Give this back to Steve to redo this wording.

Approve intent of AI: 7-0-1.

AI12-0140-1/01 Access to unconstrained partial view when the full view is constrained

The subject is backwards. Randy needs to explain why the RM wording doesn't work; he ran out of time to reconstruct that description.

We'll defer this one until he does that.