Minutes of the 49th ARG Meeting

14-16 June 2013

Berlin, Germany

Attendees: Steve Baird, Randy Brukardt, Jeff Cousins, Brad Moore, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg (except Sunday), Tucker Taft, Tullio Vardanega (except late Saturday and Sunday).

Observers: Steve Michell.

Meeting Summary

The meeting convened on Friday, 14 June 2013 at 14:00 hours and adjourned at 12:50 hours on Sunday, 16 June 2013. The meeting was held at the Seminaris conference center in Dahlem, Berlin. The meeting covered most of the agenda.

AI Summary

The following AI was approved:

AI12-0067-1/01 Accessibility level of explicitly aliased parameters of non-functions (7-0-2)

The following AIs were approved with editorial changes:

AI12-0001-1/04 Independence and representation clauses for atomic objects (6-0-3)
AI12-0030-1/04 Formal derived types and stream attribute availability (8-0-0)
AI12-0032-1/06 Questions on 'Old (7-0-0)
AI12-0033-1/06 Sets of CPUs when defining dispatching domains (7-0-0)
AI12-0034-1/03 Remote stream attribute calls (5-0-2)
AI12-0035-1/03 Accessibility checks for indefinite elements of containers (9-0-0)
AI12-0044-1/03 Calling visible functions from type invariant expressions (9-0-0)
AI12-0050-1/03 Conformance of quantified expressions (7-0-0)
AI12-0054-2/03 Aspect Predicate_Failure (8-0-0)
AI12-0056-1/04 Presentation Errors in Ada 2012 (9-0-0)
AI12-0062-1/01 Raise exception with failing string expression (9-0-0)
AI12-0069-1/01 Inconsistency in Tree container definitions (9-0-0)
AI12-0070-1/01 9.3(2) does not work for anonymous access types (6-0-3)
AI12-0072-1/03 Missing rules for Discard_Names aspect (7-0-0)
AI12-0073-1/01 Synchronous Barriers are not allowed with Ravenscar (9-0-0)
AI12-0076-1/02 Variable state in pure packages (7-0-0)
AI12-0077-1/01 Has_Same_Storage on objects of size zero (9-0-0)
AI12-0078-1/01 Definition of node for tree container is confusing (9-0-0)

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

AI12-0016-1/02 Model of dynamic accessibility checking (9-0-0)
AI12-0031-1/02 All_Calls_Remote and indirect calls (5-0-2)
AI12-0055-1/02 All properties of a profile are defined by pragmas (7-0-2)
AI12-0061-1/01 Index parameters in array aggregates (9-0-0)
AI12-0068-1/00 Predicates and the current instance of a subtype (9-0-0)
AI12-0071-1/02 Order of evaluation when multiple predicates apply (7-0-0)
AI12-0074-1/02 View conversions and out parameters (6-0-2)

The following AIs were discussed and assigned to an editor:

AI12-0042-1/03 Type invariants cannot be inherited by non-private extensions
AI12-0058-1/01 The Fortran Annex needs updating to support Fortran 2008
AI12-0066-1/00 If it ain't broke...

The following AI was discussed and voted No Action:

AI12-0054-1/04 A raise_expression does not cause membership failure (8-0-0)

Detailed Minutes

Previous Meeting Minutes

Approve minutes. 7-0-1.

Date and Venue of the Next Meeting

The next meeting will follow the HILT conference in Pittsburgh. That will be the afternoon of Friday, November 15th, all day on November 16th, and the morning of November 17th. (WG 9 has Friday morning.) If there won't be any conference sessions on Thursday afternoon, November 14th (which is not decided at this time), then the ARG should have a session then as well.

Thanks

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

Thanks to Erhard and Ada-Europe for the excellent arrangements.

Thanks to the (new) Rapporteur for his patience herding our group of cats.

ASIS

As discussed in "Old Action Items" (below), we didn't get the AdaCore information until very late in the game. As such it wasn't sensible to ask anyone to get their homework done. We have that information now (thanks to Jean-Pierre), so there will be no such excuse for the next meeting.

Jeff wonders if we could have an earlier deadline for ASIS homework in order to jumpstart that work. Randy points out that, historically, not much ARG gets done during the summer. Moreover, ACATS homework is due at the end of September, which will probably occupy a lot of ARG member's energy during the summer. A late October deadline is suggested (October 28th?), but that would be only two weeks before the deadline for the fall meeting, and as such would be pointless. So it is best to have it due for the next meeting, with Randy providing the normal (virtual) whipping for people who procrastinate.

Randy wonders whether AdaCore is committed to funding an update to the ASIS Standard. He has been getting mixed messages from AdaCore people as to how valuable they think it is. If the ASIS vendors don't want and support a new standard, then it’s essentially impossible to get done (it's not just Randy's direct funding, but also the support for other ARG members that work at AdaCore or other vendors for spending some of their time working on ASIS topics). What WG 9 wants is irrelevant to that reality. Tucker suggests that AdaCore is supporting Ada standardization activities generally, and it not choosing which activities to do; that's done by WG 9. Even so, he thinks it’s important to ensure that AdaCore is in fact supporting the ASIS work; he volunteers to take the point and verify that.

Multiprocessing

Canada asked WG 9 to consider starting an activity specifically to support massively parallel programming. They also proposed to have SC 22 consider trying to find a common mechanism for all programming languages so that interoperability between the solutions is possible.

It's unclear what we could do that would support interoperability; the obvious ideas for Ada (like supporting a parallel loop construct) are going to be Ada-specific.

C++ is focusing on OpenMP and Silk and trying to add those to the standard. They have set up a subgroup for this purpose. C is also starting a project.

Tucker, Steve Michell, and Brad will monitor what those groups are doing and pass information to the ARG.

ARG Procedures

As noted in e-mail, we need to update the ARG procedures to add the two new statuses we agreed on during the last meeting. Jeff read the procedures closely when he was asked to take over as Rapporteur, and he also suggested a number of changes.

The approval date is wrong, needs to be corrected to June 14th.

Jeff would like subheadings in Section 2: 2.1 Initial processing of Comments, 2.2 Development of Commentaries, 2.3 Approval of Commentaries.

Tucker would like to change the term “poor” as used in the Hold status, to “thought to not be a good candidate”.

Approve procedures with changes: 9-0-0.

On Saturday, Jeff asks that we create an informal description of the rules of running a meeting. Tucker gave him a summary and he will produce a draft for discussion, which we will add to the procedures as a non-normative section.

On Sunday, we consider Jeff's proposed meeting rules.

We need to add text to say that "Keep alive" needs to be assigned to an ARG member. "Approve as is" and "No Action" are an action on the editor to change the status, but that probably goes without saying.

“appears desirable” should be “warrants investigation”

Approve adding this with changes to the procedures: 7-0-0.

Old Action Items

Steve Baird did not create a new proposal for AI12-0020-1. He also didn't work on the paper underlying AI12-0016-1 (he did update that AI, but it mainly refers to the non-existent paper). He also did not propose a checked specification that a subprogram is non-blocking. Ed Schonberg didn't work on AI12-0002-1. All other Ada language action items were completed.

There is a question about whether Steve Baird's paper on accessibility should get on the agenda for HILT. The deadline is in a couple of weeks, which seems like insufficient time for such a large undertaking. Someone comments that the technology is probably not of interest to most Ada practitioners (it's really relevant only for compiler-writers). It is suggested that there is a more general session on accessibility at HILT. Tucker will work with Tucker and Randy on that.

The ASIS homework for most people was contingent on getting the current state of the ASIS-for-GNAT (which proposes features for most of Ada 2012). However, making that available to the ARG took a long time. Randy tried to get various ARG members to do that, and either was rebuffed or never got any response. Finally, Tucker was nice enough to send him the entire unfiltered specification in late April, but by then Randy didn't have time to extract the Ada 2012 information. Jean-Pierre beat him to that by posting the Ada 2012 routines on May 22nd. That was way too late to avoid an unmanageable crush of homework at the last minute, so Randy decided to suspend any work on ASIS for this meeting. (For more on this topic, see ASIS, above.)

No one has finished their ACATS assignments yet, but as they aren't due until September 30th, that's not surprising.

Current Action Items

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

Steve Baird:

- AI12-0016-1
- AI12-0020-1
- AI12-0042-1
- AI12-0061-1
- AI12-0071-1 (with help from Tucker Taft)
- AI12-0074-1
- 2 ACATS tests (see separate assignments).

John Barnes:

- ASIS recommendations (see separate assignments).
- 4 ACATS tests (see separate assignments).

Geert Bosch:

- ASIS recommendations (see separate assignments).
- 4 ACATS tests (please ask Randy for assignments).

Randy Brukardt:

- AI12-0066-1
- Create AI to fix wording of all real-time aspects (see AI12-0033-1/05).
- ASIS recommendations (see separate assignments).
- 4 ACATS tests (see separate assignments).

Editorial changes only:

- AI12-0001-1
- AI12-0030-1
- AI12-0032-1
- AI12-0033-1
- AI12-0034-1
- AI12-0035-1
- AI12-0044-1
- AI12-0050-1
- AI12-0054-2
- AI12-0056-1
- AI12-0062-1
- AI12-0069-1
- AI12-0070-1
- AI12-0072-1
- AI12-0073-1
- AI12-0076-1
- AI12-0077-1
- AI12-0078-1

Alan Burns:

- 4 ACATS tests (see separate assignments).

Jeff Cousins:

- ASIS recommendations (see separate assignments).
- 4 ACATS tests (see separate assignments).

Gary Dismukes:

- 2 ACATS tests (see separate assignments).

Bob Duff:

- 3 ACATS tests (see separate assignments).

Greg Gicca:

- ASIS recommendations (see separate assignments).

Steve Michell:

- Monitor other language's efforts for parallel programming, and report the applicability (or not) for Ada (with Brad Moore and Tucker Taft).

Brad Moore:

- AI12-0031-1
- Monitor other language's efforts for parallel programming, and report the applicability (or not) for Ada (with Steve Michell and Tucker Taft).
- 2 ACATS tests (see separate assignments).

Erhard Ploedereder:

- 4 ACATS tests (see separate assignments).

Jean-Pierre Rosen:

- 2 ACATS tests (see separate assignments).

Ed Schonberg:

- AI12-0002-1
- AI12-0058-1 (with Van Snyder and Tucker Taft)

- ASIS recommendations (see separate assignments).
- 4 ACATS tests (see separate assignments).

Van Snyder:

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

Tucker Taft:

- AI12-0058-1 (with Ed Schonberg and Van Snyder)
- AI12-0068-1
- AI12-0071-1 (assist Steve Baird)
- SI99-0062-2 (unwinding the introduction; this part will have a new SI number)
- SI99-0066-1
- ASIS recommendations (see separate assignments).
- 3 ACATS tests (see separate assignments).
- Check on AdaCore's commitment to supporting ASIS standardization work (as opposed to work on 8652).
- Monitor other language's efforts for parallel programming, and report the applicability (or not) for Ada (with Brad Moore and Steve Michell).

Bill Thomas:

- ASIS recommendations (see separate assignments).

Tullio Vardanega:

- AI12-0055-1
- 2 ACATS tests (see separate assignments).

Detailed Review

The minutes for the detailed review of AIs and SIs are divided into ASIS Issues (SIs) and Ada 2012 AIs (AI12s), but no SIs were considered at this meeting. The AIs and SIs are presented in numeric order, which is not necessarily the order in which they were discussed. Votes are recorded as “for”-“against”-“abstentions”. For instance, a vote of 6-1-2 would have had six votes for, one vote against, and two abstentions.

If a paragraph number is identified as coming from the working Ada 202x AARM, the number refers to the text in draft 1 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

AI12-0001-1/04 Independence and representation clauses for atomic objects

The wording for 13.2(7/3) has too many “or”s. Tucker wonders why “volatile” is in this paragraph; nothing about it requires independent addressability or anything else incompatible with packing.

So, change it to:

> For a packed type that has a component that is of a by-reference type, that is specified as independently addressable, or that contains an aliased part, then the component shall be aligned according to the alignment of its subtype. Any other component may be aligned arbitrarily.
>
> AARM Note: "Atomic" implies "specified as independently addressable", so we don't need to mention atomic here.

The last sentence is weird, as this a requirement, we don't need this extra sentence at all. Just drop it.

Tucker notes that this has to apply to all components of a type -- if there is more than one atomic component, it has to apply to all of them. The previous wording doesn't do that. So reword it to say:

> Any component of a packed type that is of a by-reference type, that is specified as independently addressable, or that contains an aliased part, shall be aligned according to the alignment of its subtype.

> AARM Note: "Atomic" implies "specified as independently addressable", so we don't need to mention atomic here.

As previously discussed, this does not include "volatile" components; we don't require independent addressability for volatile, so it shouldn't have any change on Pack.

Randy had inserted some thoughts into the AI; he asks that we decide on those.

The group agrees that the change to C.6(13.2/3) gives the wrong impression. So make no change to this paragraph, as none is needed.

The group agrees that the Language Design Principle is a good idea. In particular, programs (and programmers) should not be relying on Pack to provide a particular representation to match external requirements. (The e-mail example of using Pack rather than Component_Size = 1 to get an array of bits is precisely what should be avoided.) The editor can craft the Language Design Principle (that's the typical penalty for any good idea within the ARG, so he can't complain about the extra work).

Erhard would like to note that Pack cannot silently destroy *specified* independently addressability. But it can clobber independent addressability if not specified. This idea doesn't result in any wording.

Erhard would like to see C.6(21/3) as a user note, because he thinks this is important.

So add:

> Note: Specifying the Pack aspect cannot override the effect of specifying an Atomic or Atomic_Components aspect.

Approve with changes: 6-0-3.

AI12-0016-1/02 Model of dynamic accessibility checking

The AARM notes were modified, no other wording. Those notes refer to a paper not yet written, so we can hardly approve this.

This will remain with Randy and Steve Baird to create the paper that's not yet written.

Approve intent: 9-0-0.

AI12-0030-1/03 Formal derived types and stream attribute availability

The summary should mention “formal derived type”, which is the case that we are worrying about.

The question is whether we want to have these reemerge; having the ancestor stream attribute reemerge would make it impossible to properly stream anything with user-defined attributes in generics. OTOH, many other things reemerge in generics, so it would be consistent to do so. It is noted that just because Ada 83 did some things wrong does not mean that we have to make the same mistake again.

We eventually agree that reemergence is not what we want, so the summary and recommendation are completely bogus.

Steve Baird will redo the summary, recommendation, and question (and maybe discussion).

Approve intent: 6-0-3.

AI12-0030-1/04 Formal derived types and stream attribute availability

Steve Baird sent AI updates during the break.

Drop the “should it be? (yes.)” as that's redundant.

Change !recommendation to (See summary.)

Approve AI with changes: 8-0-0.

AI12-0031-1/02 All_Calls_Remote and indirect calls

We can't figure out who wrote this. Eventually, we find that Brad was assigned this one. He claims to not remember it at all.

This matters for calls outside of the package, but in the same partition.

Why is “local” deleted? This word is ambiguous (local here means “in the same partition”). The text is making that clear, rather than using an unclear term.

We have a hard time figuring out what is going on here. We already decided most of this in Stockholm. Tucker insists on working it all out again.

Remote-access-to-subprogram objects are fat. So the partition id needs a bit or something to tell between All_Calls_Remote and the normal case. That can always be determined at the point where the 'Access is taken, so one extra bit is the worst case. And, an implementation can always make all calls remote through remote-access types and thus not even need any extra overhead. So the answer to the question is really yes. All this aspect does is prevent the optimization of using a local call when the actual subprogram turns out to be in the same partition.

The questioners other concern about normal access-to-subprogram is a red herring, because such a type cannot be visible outside of the declarative region of the ACR package. Thus all such calls can be local calls.

Thus there is no issue here at all, and this is just plain wrong.

Randy should take the E.2.3(16.a/3) note, and add that the AARM AI (that is, make this change, but drop “direct”).

No action: 7-0-1.

Sunday morning, Brad asks for reconsideration of this AI. He believes (as described in a message he sent overnight) that there is indeed a case that we failed to handle.

In particular, normal access-to-subprogram calls are a problem. For such calls, the designated subprogram is always local to the partition (not necessarily to the package).

The 'Access in Normal_Package in Brad's e-mailed example is illegal. [Editor's note: Why it is illegal was not recorded, unfortunately.] But the other part is legal so there is a smallish problem here (as the existing wording implies that this somehow is a remote call) – clearly we could have a legal 'Access somewhere.

Jean-Pierre proposes that the model should be that All_Calls_Remote says that all calls that *could* be remote have to be remote. But local calls that can't be remote don't suddenly try to go through the PCS.

E.4 says there are only three ways to make a remote procedure call. A normal access-to-subprogram is not on this list, so there is no way it could ever be a remote call. Tucker notes that such a call would be across a boundary which could not be separated into different partitions (without changing the categorization of the packages, in which case the access-to-subprogram would have to have become a remote type).

So we decide that All_Calls_Remote can only affect calls that might be remote.

Tucker says that a minimum we need to change E.2.3(19/3) from “the implementation shall route any call” to “the implementation shall route any remote call”.

Give this back to Brad to write wording and discussion to this effect. (And make sure to include the AARM note that the ARG wanted to keep originally.)

Keep alive: 5-0-2.

AI12-0032-1/06 Questions on 'Old

Erhard wonders why we complicate the wording by mentioning “postcondition expression that is enabled”. Steve Baird says that is necessary to prevent possible undesirable side-effects when the postconditions are disabled; such postconditions ought to have no effect.

An example would be:

```
Post => ... X(F(...))'Old ...
```

Tucker notes that Large_Array'Old takes a lot of storage, we don't want that in our real-time program when we've explicitly “Ignored” postconditions.

It clearly has to be undefined when ignored. Steve Baird points out that this is only the dynamic semantics of the attribute; it has no effect on static meaning.

Typo: “anoynmous” in the wording.

Potentially unevaluated needs to cover quantified expressions, else cases like `(for I in 1 .. 10 => A(I)'Old = A(I))` would be allowed. That is illegal by 6.1.1(27). But that rule doesn't prevent `(for I in 1 .. F(...) => A(F(N)'Old))` which also should be illegal.

So add the predicate of quantified_expressions to potentially unevaluated.

Tucker suggests:

- If X is of an anonymous access defined by an access_definition *A* then

```
X'Old : constant A := X;
```

to replace the first three subbullets of 6.1.1(26/3).

Erhard suggests that we drop the sentence about the nominal subtype, because that follows from the analogy.

Change: "Prepend [(append?)] to {AARM} 6.1.1(27.d(3)):"

There is no ordering rule for finalization for a task entry, that seems to be missing. Change the last sentence of the new text for 6.1.1(35/3):

> [For a call to a subprogram or protected entry, the]{The} postcondition check {for any call} is performed before the finalization of any implicitly-declared constants associated (as described above) with Old attribute_references but after the finalization of any other entities whose accessibility level is [the] that of the execution of the {callable construct}[body of the callable entity].

Make this a separate paragraph, otherwise the paragraph is way too large.

Question (3) should be answered that the nominal subtype follows from the definition of the constants. So the answer is (Yes and it is.). (It is defined as the consequence of other rules.)

The wording change for 4.1.4(9/3) should be first in the !wording section. We're happy with it as it is; drop the question suggesting a change.

Tucker groans that 4.1.4(9/3) requires explicit text to define the nominal subtype. But we're doing that implicitly.

Add back the nominal subtype wording as follows:

> The nominal subtype of X'Old is as implied by the above definitions.

Simplify the last subbullet of 6.1.1(26/3):

- Otherwise:

```
X'Old : constant S := X;
```

  where *S* is the nominal subtype of X. Redundant[This includes the case where the type of *S* is an anonymous array type or a universal type.]

We believe this is OK as nominal subtype is well-defined for all expressions.

Approve AI with changes: 7-0-0.

AI12-0033-1/04 Sets of CPUs when defining dispatching domains

Tucker thinks that checking for empty domains is a problem. These are limited types and it isn't practical to make the initializations conditional, especially as these can only be created during library-level elaboration. Thus we need to allow empty domains (in case runtime parameters, like the number of CPUs on the target, are involved).

Therefore, we should make it raise an exception if someone attempts to assign a task to a domain with no processors.

So we should drop the wording about {, or if the set or range contains no processors}.

Assign_Task would need a check for a domain with no processors (paragraph 26). Similarly, paragraph 20 would need a rule to that effect (to cover the case of the Dispatching_Domain aspect).

Tucker will take this and create wording.

Approve intent: 7-0-2.

AI12-0033-1/05 Sets of CPUs when defining dispatching domains

Tucker sent an update overnight.

Tucker explains that he cleaned up the D.16.1(20/3), since these are aspects of task types. Erhard notes that this never says **when** this evaluation occurs. Randy notes that this wording is a copy of Priority and CPU. And those don't really say, either.

These clearly have to happen before the activation of the task.

“The expression specified for the dispatching domain aspect of a task type is evaluated each time an object of the type is created. If ... is raised; otherwise, the newly created task is assigned to the domain identified by the value of the expression.”

Randy will create an AI to make similar changes to all of the other similar Annex D aspects. He notes that this would fit under the “if it ain't broke, don't fix it” AI. Erhard says he wants it fixed.

D.16.1(24/3): “...but the returned values are otherwise unspecified”.

Steve Baird asks what happens if type CPU contains only 1 value. That seems reasonable on a mono-processor. But Tucker's wording is impossible in that case (there is no value one less than Get_CPU_First). Preferably, Get_CPU_Last should return zero in the empty case. That requires changing this to return CPU_Range. That seems better than “unspecified”. Tucker will do this during the upcoming break.

In D.16.1(26/3), “Dispatching_Domain” is a type, but this is talking about the concept. It should say “dispatching domain”.

AI12-0033-1/06 Sets of CPUs when defining dispatching domains

After the coffee break, a new version of the AI has appeared in our inboxes.

Randy will handle the “Dispatching_Domain” to “dispatching domain” in a variety of places, in a presentation AI. Erhard will send a reminder to the ARG list (which he did immediately). [Note: The fix in D.16.1(26/3) is easier to do in this AI, so it was included here; the others – if any - will be done in a presentation AI - Randy.]

Change D.16.1(26):

> Assigning a task already assigned to System_Dispatching_Domain to that domain has no effect.

Approve with changes: 7-0-0.

AI12-0034-1/02 Remote stream attribute calls

The added rule belongs to E.2.2, and probably should go after paragraph 14.

The question example uses “Stream” without defining it.

Ed will try to contact AdaCore people to determine whether this is too incompatible with their users and implementation. (We don't want to require something wildly different than existing practice.)

Keep alive: 8-0-0.

AI12-0034-1/03 Remote stream attribute calls

Randy rewrote this Saturday night based on the e-mail conversation with Thomas Quinot.

Legal? In the example of the question should simply say (No.), and remove “-- [1]”.

“...such a stream attribute {call} by definition...”

Approve AI with changes: 5-0-2.

AI12-0035-1/03 Accessibility checks for indefinite elements of containers

Steve Baird explains this one for Gary (who isn't here).

Erhard does not like the use of “allocation” (in "perform indefinite allocation") here as it may not be implemented that way. After much discussion, we think “retains” is a better term. “retain indefinite objects” is proposed. But that doesn't sound like a property of the subprogram (it sounds more like a property of the container as a whole).

"Perform indefinite insertion” is suggested as the term. That meets general approval. It words in part because a copy of the object is going to be returned from any function.

There are no checks unless the object type has access discriminants or is class-wide.

In the !summary, change “an access type associated with the instance” to “ a notional access type with the designated type of the element type of the instance”.

Changes for the wording:

Modify the second paragraph of A.18(5/3): “...[no]{the} container is {not} modified by the call.”

The first bullet following the above should be changed to “value of the qualified_expression is that of the parameter; and” rather the “the initial value is the value of the parameter; and”.

All of these “Dynamic Semantics” sections should just be an additional bullet at the end of the Static Semantics sections, as language-defined packages are (almost) always defined in their entirety in Static Semantics.

[The new wording after A.18(5/3) contains several instances of "section 4.8" and "section 12.3". These haven't been called "sections" since Ada 83, they now should be "subclauses". See AARM 1.1.2(1.a/3) for the details. These were fixed in the final AI -- Editor.]

[For Containers.Indefinite_Holders, it makes more sense to put this text into each subprogram's definition than to have a separate rule, especially as only two subprograms are listed as performing indefinite allocation. Moreover, Holders is different than the other indefinite containers in that it is completely defined here rather than as differences from a "normal" container. This change was made in the final AI – Editor.]

[The text "This call should presumably fail" in the examples (two places) should be "Raises Program_Error because of new rules", since the AI specifies what happens for these calls. Consider this an "editorial review" change – Editor.]

Approve AI with changes: 9-0-0.

AI12-0042-1/02 Type invariants cannot be inherited by non-private extensions

If one inherits a private operation such that it becomes a public operation, it then becomes subject to the type invariant. Tucker proposes that such operations have to be overridden; then the fact that it is now public and now checked is more obvious.

Edit the first paragraph of the wording: “If a Type_Invariant'Class expression applies to any ancestor of a type extension, the extension shall be a private extension or a record extension that is the completion of a private type or {private} extension that has a Type_Invariant'Class that applies. 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.”

Randy says that he thought we had decided to get rid of this first rule. It is really a methodological issue. We should allow type invariants so long as there are any private “pieces”.

It's not clear that everyone agrees with the last sentiment given above.

Straw poll - keep first paragraph: 2-2-5.

That's inconclusive. We discuss this a bit.

Repeat the straw poll – keep first paragraph: 3-2-4.

Still inconclusive; continue to discuss this.

Repeat the straw poll again – keep first paragraph: 6-0-3.

Do we want the second paragraph as proposed by Tucker? Straw poll: 9-0-0.

Steve Baird will take the AI.

The discussion continues on whether we should be banning Type_Invariant'Class on types derived from visible record types. That's the other part of the methodological restriction. Tucker worries about a mix-in generic instantiated with a regular record type; it could not introduce a type invariant on just its components.

We do not have conclusion on this restriction; we'll leave it for Steve Baird to decide and justify his choice.

Keep alive: 9-0-0.

AI12-0042-1/03 Type invariants cannot be inherited by non-private extensions

The term "dynamically callable" is necessary because the definition is recursive. That needs to be explicit in the wording. Steve Baird will reword this.

AI12-0044-1/02 Calling visible functions for type invariant expressions

Tucker says that he thinks the default checking rules should not cause recursion. He believes it is too hard to remember to use a special aspect on functions that are intended to be called from an invariant expression – in that case, forgetting to use the aspect would cause recursion. It is pointed out that a friendly compiler would warn the user in that case. Tucker says that's true for calls directly from the invariant expression, but not necessarily for calls from in the body of functions that are called from the invariant expression.

An objection is raised: a function body could call a function with an **in out** parameter or (horrors) a procedure, and trigger recursion that way. Tucker agrees, but argues that those are unlikely to occur in real query functions (the kind typically used in assertion expressions). Hopefully, we'll have global-in/global-out annotations in the next version of Ada, and those could be used to detect problematic functions in assertion expressions.

Aside: the new SPARK is providing the root of a solution for the global-in/global-out problem. It wouldn't be close to a complete solution, since SPARK doesn't do dynamic memory allocation, but it could be used as a starting point. There should be an AI12 about this topic [A mostly empty AI, AI12-0079-1, has been created to hold this].

Jeff thought that the expression function rule was better. Tucker replies that he does not want to break the equivalence between regular functions and expression functions.

We do not want compilers to insert additional checks of assertions of any kind, including invariants, under the normal policies. (An implementation-defined policy could do that, of course). Doing that could cause a portability problem, either because of additional failures occurring in "working" code, or because of poorly behaved assertion expressions.

Steve Baird notes that entries aren't covered by this wording. We need to say “procedure or entry”.

Tucker hates this wording after this change, he would like to reword it and bring it back later.

Approve intent: 7-0-2.

AI12-0044-1/03 Calling visible functions from type invariant expressions

Tuck sends new wording in an e-mail:

- has a result with a part of type *T*, or one or more {**out** or **in out**} parameters with a part of type *T*, or an access [to variable] parameter whose designated type has a part of type *T*[.]{;
- is a procedure or entry and has an **in** parameter with a part of type *T*.}

Randy notes that access-to-subprogram parameters don't have a designated type, so say “access-to-object”. (That was implicit in "access-to-variable".)

Add an AARM note to explain that we don't check for functions with **in** parameters to avoid infinite recursion for public function calls appearing in type invariant expressions.

Approve AI with changes: 9-0-0.

AI12-0050-1/02 Conformance of quantified expressions

Erhard asks that we drop the final 's' in the summary. This is singular, so the summary is correct as is. (These things are mutually exclusive.)

There doesn't seem to be any requirement that both expressions are quantified expressions. The previous bullet requires that they are the same syntactic category, so the wording doesn't need to mention that.

Tucker tries to change the wording, as he doesn't believe that `(for I in 1..10 => True)` and `(for J in 1..10 => True)` fail to conform by this wording. He wants a bullet before this that each defining_identifier is the same.

Our desire for dinner exceeds our desire to keep wordsmithing this to finish it now. So we give it to Tucker to make a wording proposal for consideration tomorrow.

Approve intent: 6-0-1.

AI12-0050-1/03 Conformance of quantified expressions

Tucker explains his wording updates. He added a new bullet.

“shall be” should be “are” in 6.3.1(20).

Approve AI with changes: 7-0-0.

AI12-0054-1/04 A raise_expression does not cause membership failure

We're going to use the other alternative to this AI, so this one is dead.

No Action: 8-0-0.

AI12-0054-2/01 Aspect Predicate_Failure

Steve Baird notes that there should be a rule that aspect Predicate_Failure is only allowed when there is a predicate on the subtype.

Steve Baird suggests that semantics be that it evaluates the expression, and then raises Assertion_Error. (Usually it does not get that far.)

Jean-Pierre is uncomfortable with discarding the result of the function. The suggestion is that it is a string expression, which is used as the parameter to the Assertion_Error message. That way, the result is not discarded (which would be a new weirdness).

The ordering of evaluation is in AI12-0071-1.

It has to be reworded that so that Predicate_Failure clearly applies to the current predicate (only); other predicates use their own Predicate_Failure aspects.

Should we have something similar for preconditions? Randy notes that that would bring up issues for having multiple preconditions on a subprogram, because you can't just declare an extra subtype to add a separate exception. And preconditions are explicitly unordered, that would make it hard to determine which failure message is going to occur for a particular call. Usually you need determinism in the order that checks are made.

Approve intent: 8-0-1.

AI12-0054-2/02 Aspect Predicate_Failure

Jean-Pierre provided a new version for our consideration on Saturday.

Ed would like the last sentence to be phrased more like a question. Tucker thinks it is OK, as it is worded as a question. “there is” should be “would be” in the last sentence of the question.

The wording should say something about an exception being raised.

Tucker suggests,

> This aspect specifies an expression that will be evaluated if a predicate check for the subtype fails; {if it does not raise an exception, }it defines the Message associated to the occurrence of Assertion_Error {which}[wich] is raised by the failure of the condition.

Tucker thinks that we need to specify that the exception raised by the expression is propagated.

We need to associate this directly with the predicate expression that is False.

> This aspect specifies an expression that will be evaluated only if a predicate check fails because some predicate aspect of this subtype evaluates to False; if the evaluation of Predicate_Failure expression propagates an exception occurrence, then that occurrence is propagated, otherwise, Assertion_Error is raised with the message defined by the value of the Predicate_Failure expression.

It would misleading to allow Predicate_Failure on any subtype that doesn't have any predicate expressions. Steve Baird notes that this has to participate in static matching. By preventing Predicate_Failure from being specified on subtypes without a predicate expression, the existing static matching rule is sufficient. (If the predicate expression

came from the same declaration, then the Predicate_Failure expression also comes from the same declaration and thus is necessarily the same.)

Tucker will take this and produce wording for later in the meeting.

Jean-Pierre notes that the Show_Window example in AI12-0022-1 (already approved by WG 9) is a bad idea. We shouldn't modify that AI, but this AI ought to give the example "correctly" and suggest that other example is incorrect.

Approve intent: 7-0-2.

AI12-0054-2/03 Aspect Predicate_Failure

Add "the" in "If any of predicate checks fail" in the text after 3.2.4(31/3).

Add a user note in 3.2.4: Predicate_Failure expressions are never evaluated during the evaluation of a membership test or Valid attribute.

Add a second user note: A Predicate_Failure expression can be a raise expression.

Drop "a *string*_expression", replace by "an expression", because it is not mentioned in the syntax.

Randy suggests adding some examples to the end of the clause.

We should add the examples from the AI to the end of the clause; add a forward reference to Text_IO to the second one.

The string in the Open_File_Type is wrong, it should be "File not open".

Approve AI with changes: 8-0-0.

AI12-0055-1/02 All properties of a profile are defined by pragmas

Tucker hates "No_Not_A_Specific_CPU". He suggests "No_Use_Of_Not_A_Specific_CPU". That's hardly any better. We try "No_Unassigned_Tasks", but that's vague as assigned to what. So we settle on. "No_Tasks_Unassigned_To_CPU".

Tullio will take this AI and provide wording.

Approve intent: 7-0-2.

AI12-0056-1/04 Presentation Errors in Ada 2012

For the second note for 6.2, change:

> A formal parameter of mode **out** {might}[may] be [partially or completely] uninitialized at the start of the subprogram_body (see 6.4.1).

Approve AI with changes: 9-0-0.

AI12-0058-1/01 The Fortran Annex needs updating to support Fortran 2008

Van Snyder has provided a list of problems with the Fortran annex. Jeff says that he showed that to his company's Fortran expert, and was told that Van was correct that these things are problems with the Annex.

The AI has no wording, Van didn't feel competent to provide it.

How are we going come up with wording for this? We don't seem to have anyone that knows both languages well-enough to do that.

Tucker and Ed both volunteer to work with Van and produce wording for these changes.

Keep alive: 9-0-0.

AI12-0061-1/01 Index parameters in array aggregates

Do we need wording to make this a constant? Tucker suggests that we just call it a loop parameter, then the existing wording will handle that.

Conformance rules still need to be filled in. Something like the wording used in AI12-0050-1, but we haven't discussed that one yet. (We did later, of course, see above.)

Steve Baird will take this one to finish the wording.

Erhard is worried about the way that this is declared. If it says "parameterized_array_component_association declares" or similar, we could have a problem. We need to make sure that the declarative region for the loop parameter is correct, and that the loop parameter isn't hidden from all visibility. He's specifically talking about 8.3(16); this means that the declaration must not be the entire association. So the thing in front of the arrow should have its own name such that it is the declaration, the region is the entire association.

Approve intent: 9-0-0.

AI12-0062-1/01 Raise exception with failing string expression

The references to AI12-0054-1 at the end of !response and in the wording should be deleted. The associated wording goes away as well.

Erhard worries that this would cover bugs:

```
raise Bomb_Exploding with F(A); -- F(A) raises Constraint_Error.
```

The “wrong” handler would be triggered in this case.

Tucker notes that this does the same thing:

```
Raise_Exception (Bomb_Exploding'Identity, F(A));
```

You have to evaluate the parameters first, for which F(A) raises an exception. We want these to execute the same way, as switching between the forms shouldn't change the semantics.

Code reviewers (both human and automated) need to be aware that messages can fail and check for that.

Erhard would like a user note to this effect; turn the first AARM note into a user note.

Approve AI with changes: 9-0-0.

AI12-0066-1/00 If it ain't broke...

The group agrees that we shouldn't try to fix these. Randy wants to leave this open for future topics, so we don't take a vote.

AI12-0067-1/01 Accessibility level of explicitly aliased parameters of non-functions

It proves hard to explain the exact problem to the group. Once that's done however, there is no objection to the fix.

Approve AI: 7-0-2.

AI12-0068-1/00 Predicates and the current instance of a subtype

Tucker suggests that the current instance of a subtype be defined as a constant view of an object within a predicate. Randy notes that one can pass scalar literals as **in** parameters which would cause a scalar predicate to be evaluated. Those are values, not objects.

Steve Baird says that if these are objects, then the predicate can query the properties of the object. He thinks that might be useful; he gave an example that made its way into the question of the AI.

Tucker thinks it is weird to treat these as objects, as it is strange to allow 'Size and the like. Even for composite types (which probably will be passed by reference), you might be looking at temporaries. We don't want to be making promises that you can read the actual object in a predicate.

“In an aspect specification for a type or subtype, the current instance represents a constant value of the type. (It is not an object.)”

So 'Size, 'Address, 'Constrained are illegal.

Steve Baird says that it's important that the **in** parameter implementation model works. Tucker says that's fine with this wording; there might be a need for an extra Legality Check, but the runtime should not be affected.

What about 'Length and 'First for arrays? Those work on array values (the wording is “prefix of an array type”) so these would be allowed.

Tucker will take this one and produce wording.

Approve intent: 9-0-0.

AI12-0069-1/01 Inconsistency in Tree container definitions

Tucker is confused by the definition of node for the tree containers, found at the start of A.18.10. He would like them clarified. He is directed to make a proposal as a new AI (which became AI12-0078-1).

“...starting {from}[with] the root node...” in all of these as the root node is not returned. Do the same with the subtree versions (they can be called with the root node, of course).

Approve AI with changes: 9-0-0.

AI12-0070-1/01 9.3(2) does not work for anonymous access types

Summary, “irregardless” should be “regardless”.

There are two alternative 1s in this AI, the second one should be labeled 2.

Randy tries to explain that the existing otherwise bullet should be appropriate for all other cases.

Tucker comes out in favor of the minimum change, which is just to add “named” (the second alternative #1). He says that there is value to repeating the rules for the two most common cases, but if there is anything the least bit unusual, we should just drop into the "otherwise". There appears to be agreement with this plan.

Second "vaguely-related" issue. Randy explains that Adam noted that the only accessibility level not defined in 3.10.2 was the one he was interested in. That seems like a mistake, they all should be in 3.10.2. So we add a rule in 3.10.2 to cover this case. A wording improvement is suggested:

> The accessibility level of the anonymous access subtype defined by a return_subtype_indication {that is an access_definition (see 6.5)}[in an extended_return_statement] is that of the result subtype of the enclosing function.

Put this after 3.10.2(13.1/3).

Approve AI with changes: 6-0-3.

AI12-0071-1/01 Order of evaluation when multiple predicates apply

Steve Baird worries about multiple constraints on a subtype -- compilers do not check that original constraints hold. (If we have ranges of 1..10 and then a further subtype of 2..9, only 2..9 would get checked for a constraint check). He would prefer that all constraints/exclusions are checked before any predicates are checked.

Steve Baird will take this AI and craft wording to this effect.

Approve intent: 9-0-0.

AI12-0071-1/02 Order of evaluation when multiple predicates apply

Steve Baird sent three messages overnight with improved wording. The second message Steve sent should be ignored.

“anded together” should be replaced by “combined”.

“If all the these tests” should be “If all of these tests”.

Make this part into an AARM implementation note:

> This test need not be performed in the case of a predicate test for one subtype that is part (as described below) of a predicate test for another subtype, [If S2 is a subtype of S1 and we have already established that the constraints and null exclusions of S2 are satisfied, testing those of S1 would be redundant.]

“unspecified order” should be “arbitrary order”.

Italicize “satisfies the predicate”.

The progenitors are tested in an arbitrary order. For the following:

```
type T is new A and B and C ...
subtype S is T;
```

The order is constraints and null exclusions of S, predicates of A and B and C in some order, then predicates of T, then predicates of S.

“satisfies the predicate” is the term that makes it recursive.

The large redundant stuff at the end should be an AARM note.

Randy wonders about the progentitors of a task type (or protected type). Those aren't covered by the “derived type” wording. Steve Baird complains that 3.2.4(3-5/3) says derived type declaration. Yes, that's wrong too. This should just say “type declaration” instead of “derived type declaration” in all of these places.

Tucker suggests that where we say “unspecified order”, we move that to the end.

Perhaps we need a permission to avoid evaluating predicate expressions multiple times.

Tucker volunteers to help Steve Baird with the wording to integrate AI12-0054-2.

Approve intent: 7-0-0.

AI12-0072-1/01 Missing rules for Discard_Names aspect

Randy explains that the aspect is not defined or indexed properly. It seemed better to define it like all of the other aspects rather than worrying about whether the "blanket" rules in chapter 13 cover everything.

The configuration pragma case is not well-defined (this is an old bug). A configuration pragma applies to compilation units, not to the contents. We need to say that explicitly. Randy will try to craft wording for that.

Approve intent: 6-0-3.

AI12-0072-1/02 Missing rules for Discard_Names aspect

Randy couldn't find a term for these entities, so he inserted a placeholder, "muggywomp". Steve Baird says he would have preferred Randy use "Frobisher" for this, since he's partial to little-known Arctic explorers. A groan arises.

Steve Baird now makes a serious (?) suggestion of “reified name entity”. The groan continues. Someone asks what is the commonality between these entities. Randy answers that they all work with aspect Discard_Names, otherwise, they seem random to him. The groan gets louder. Someone suggests that these all entities of some name (identifier) that shows up at runtime. Finally, the groan disappears. Tucker suggests “runtime name strings” for the term. Steve Baird throws out “dynamic name text”. Tucker then suggests “runtime name text”. The problem with all of these is that an exception or enumeration type isn't "name text", of any kind. The mouthful “entities with runtime name text” is suggested.

We try adding that to the introductory text:

"...used for reducing the storage for entities with runtime name text."

Randy will rewrite the AI using “entities with runtime name text”. It's not just a drop in, so we'll have to see this AI again.

Approve intent: 8-0-0.

AI12-0072-1/03 Missing rules for Discard_Names aspect

Randy rewrote this before dinner Saturday night.

The subclause title should be “Aspect Discard_Names”.

C.5(1): “Specifying the aspect Discard_Names can be used ...”

Approve AI with changes: 7-0-0.

AI12-0073-1/01 Synchronous Barriers are not allowed with Ravenscar

Correct the wording: No_Depenence should be No_Dependence.

The question is rather bogus – but Randy notes that it is the question that was asked. We shouldn't change it too much.

Tullio says that Ravenscar is based on the idea that each waiting place can only cause a single task to wait. That means that each event releases precisely one task; this is important for predicability. Barriers violate this; therefore, barriers are incompatible with Ravenscar.

Drop all of the text after and including “Other options were considered:”.

Approve AI with changes: 9-0-0.

AI12-0074-1/01 View conversions and scalar out parameters passed by copy

Steve Baird proposes that the passing in the Default_Value in the case that the (untagged) view conversion doesn't fit. That would work, but several people object to this working differently from the normal case.

Jean-Pierre would prefer that an **out** parameter is always initialized to the Default_Value. He doesn't get much support for that idea. For access types, we already pass the original value. Similarly, record types that have components with defaults have their existing values passed (not the defaults). It seems odd for scalars to work differently than this.

It is proposed to make such untagged view conversions illegal. Tucker suggests that this would be the case if they do not have a common ancestor (there is no problem if they have a common ancestor, as the representations have to be the same in that case, because of the infamous 13.1(10)). It's only checked if the target has Default_Value. View conversions that would trigger this rule have to be written explicitly (implicit view conversions exist in calls to inherited subprograms, but such types are necessarily related), and such conversions could easily be replaced by temporary objects if necessary. This rule is also compatible with Ada 2005 and earlier, as Default_Value is new in Ada 2012.

There is a generic contract issue because we don't know of a formal parameter type has Default_Value (or not). This can be handled by raising Program_Error (and assuming that formal types do not have Default_Value inside the generic). Tucker suggests that we raise that when the generic is instantiated; it would not be checked at each view conversion. That's cheaper and better. We think we already have cases like that. But when we do some research, we discover that we gave up on that idea for predicate checks. So probably we should simply do these at each view conversion (there aren't likely to be very many).

Steve Baird objects to the example at the end of the discussion; it doesn't have a view conversion.

Steve Baird will take this AI and provide wording.

Approve intent: 8-0-1.

AI12-0074-1/02 View conversions and out parameters

The check would fail in a generic body if the operand is a formal type (which we don't know about Default_Value) and the target type (which is also the formal parameter type) has a Default_Value. That's intended.

Approve AI: 6-0-2.

[Note: This AI was approved without any changes although it (1) has a summary of "** TBD"; (2) answers the question with "Not Sure"; (3) does not explain the chosen solution in the !discussion section; (4) fails to address the conflicting text in 4.6(56); (5) fails to address (even in the discussion) the similar problem for access types. What were we thinking? As such, it was left as a Work Item (still assigned to Steve Baird) until those deficiencies were addressed.]

AI12-0076-1/01 Variable state in pure packages

Steve Baird explains that there is a similar problem for “known to be constrained”. Steve would like to make all of these cases erroneous. Randy objects, as we just eliminated that erroneousness. A long discussion ensues.

Tucker thinks that it should be erroneous to change the value of a discriminant of a constant, other than in initialize and finalize. For pure packages, we need any modification to be erroneous, other than in initialize and finalize. (Controlled is pure, so controlled types can occur in pure packages.) The same is true for remote types packages.

Tullio wonders about the wording in the !discussion. The intent is that there is no mechanism to keep state in sync for pure and remote types packages – they can be replicated without overhead.

Tucker will take this AI and work out perfect solutions.

Keep alive: 9-0-0.

AI12-0076-1/02 Variable state in pure packages

Tucker provided an AI update via e-mail.

“attempts to modify” should be “modifies” in the wording (twice).

Approve AI with changes: 7-0-0.

AI12-0077-1/01 Has_Same_Storage on objects of size zero

“included” should be in “including” in the wording. Randy notes he was not happy with this wording.

Erhard suggests that the wording should be:

> The actual parameter shall be a name that denotes an object. The object denoted by the actual parameter can be of any type. This function evaluates the names of the objects involved and returns True if the representation of the object denoted by the actual parameter occupies at least one bit and exactly the same bits as the representation of the object denoted by X; otherwise, it returns False.

Tucker suggests an improvement:

> The actual parameter shall be a name that denotes an object. The object denoted by the actual parameter can be of any type. This function evaluates the names of the objects involved and returns True if the representation of the object denoted by the actual parameter occupies at least one bit and all the bits are exactly the same bits as the representation of the object denoted by X; otherwise, it returns False.

This sounds too much like the value is involved.

Reword as follows:

> The actual parameter shall be a name that denotes an object. The object denoted by the actual parameter can be of any type. This function evaluates the names of the objects involved{. It}[and] returns True if the representation of the object denoted by the actual parameter occupies exactly the same bits as the representation of the object denoted by X {and the objects occupy at least one bit}; otherwise, it returns False.

Approve AI with changes: 9-0-0.

AI12-0078-1/01 Definition of node for tree container is confusing

Tucker rewrote this introduction to make the definition more clearly, and sent the result via e-mail.

!wording

Modify paragraphs A.18.10(2,3):

2/3
A multiway tree container object manages a tree of {*nodes*, comprising of a *root node*, and a set of *internal*} [internal] *nodes* each of which contains an element and pointers to the parent, first child, last child, next (successor) sibling, and previous (predecessor) sibling internal nodes. A cursor designates a particular node within a tree (and by extension the element contained in that node, if any). A cursor keeps designating the same node (and element) as long as the node is part of the container, even if the node is moved within the container.

3/3
A subtree is a particular node (which roots the subtree) and all of its child nodes (including all of the children of the child nodes, recursively). [There is a special node, the root, which] {The root node is always present and has neither an associated element value nor any parent node{; it has pointers to its first child and its last child, if any}. The root node provides a place to add nodes to an otherwise empty tree and represents the base of the tree.

Italics in the second paragraph are preserved (they're missing in Tucker's version).

Approve AI with changes: 9-0-0.